IsarMathLib

Sławomir Kołodyński, Daniel de la Concepción Sáez

February 22, 2021

Abstract

This is the proof document of the IsarMathLib project version 1.15.1. IsarMathLib is a library of formalized mathematics for Isabelle2021 (ZF logic).

Contents

1 Introduction to the IsarMathLib project 10
1.1 How to read IsarMathLib proofs - a tutorial 10
1.2 Overview of the project 12

2 First Order Logic 16
2.1 Notions and lemmas in FOL 16

3 ZF set theory basics 19
3.1 Lemmas in Zermelo-Fraenkel set theory 19

4 Natural numbers in IsarMathLib 24
4.1 Induction 25
4.2 Intervals 30

5 Order relations - introduction 31
5.1 Definitions 31
5.2 Intervals 35
5.3 Bounded sets 36

6 More on order relations 43
6.1 Definitions and basic properties 43
6.2 Properties of (strict) total orders 45

7 Even more on order relations 47
7.1 Maximum and minimum of a set 48
7.2 Supremum and Infimum 55
7.3 Strict versions of order relations 62

8 Functions - introduction 66
8.1 Properties of functions, function spaces and (inverse) images. 66
8.2 Functions restricted to a set 80
8.3 Constant functions 82
8.4 Injections, surjections, bijections etc. 83
8.5 Functions of two variables 91

9 Semilattices and Lattices 95
9.1 Semilattices 95

10 Order on natural numbers 100
10.1 Order on natural numbers 101

11 Binary operations 102
11.1 Lifting operations to a function space 102
11.2 Associative and commutative operations 104
11.3 Restricting operations 107
11.4 Compositions 109
11.5 Identity function 110
11.6 Lifting to subsets 112
11.7 Distributive operations 117

12 More on functions 118
12.1 Functions and order 118
12.2 Functions in cartesian products 122
12.3 Induced relations and order isomorphisms 124

13 Finite sets - introduction 131
13.1 Definition and basic properties of finite powerset 132

14 Finite sets 141
14.1 Finite powerset 141
14.2 Finite range functions 148

15 Finite sets 1 149
15.1 Finite vs. bounded sets 150

16 Finite sets and order relations 153
16.1 Finite vs. bounded sets 153
16.2 Order isomorphisms of finite sets 155

17 Equivalence relations 161
17.1 Congruent functions and projections on the quotient 161
17.2 Projecting commutative, associative and distributive operations. 167

17.3 Saturated sets 169

18 Finite sequences 172
18.1 Lists as finite sequences 172
18.2 Lists and cartesian products 186

19 Inductive sequences 188
19.1 Sequences defined by induction 189
19.2 Images of inductive sequences 196
19.3 Subsets generated by a binary operation 197
19.4 Inductive sequences with changing generating function 199

20 Folding in ZF 203
20.1 Folding in ZF 203

21 Partitions of sets 207
21.1 Bisections 208
21.2 Partitions 210

22 Enumerations 212
22.1 Enumerations: definition and notation 212
22.2 Properties of enumerations 213

23 Semigroups 216
23.1 Products of sequences of semigroup elements 216
23.2 Products over sets of indices 220
23.3 Commutative semigroups 223

24 Commutative Semigroups 235
24.1 Sum of a function over a set 235

25 Monoids 239
25.1 Definition and basic properties 239

26 Groups - introduction 243
26.1 Definition and basic properties of groups 244
26.2 Subgroups 256

27 Groups 1 263
27.1 Translations 263
27.2 Odd functions 271
27.3 Subgroups and interval arithmetic 271

28 Groups - and alternative definition 276
28.1 An alternative definition of group 277

29 Abelian Group 279
29.1 Rearrangement formulae 279

30 Groups 2 291
30.1 Lifting groups to function spaces 291
30.2 Equivalence relations on groups 296
30.3 Normal subgroups and quotient groups 299
30.4 Function spaces as monoids 304

31 Groups 3 305
31.1 Group valued finite range functions 305
31.2 Almost homomorphisms 307
31.3 The classes of almost homomorphisms 316
31.4 Compositions of almost homomorphisms 318
31.5 Shifting almost homomorphisms 327

32 Direct product 328
32.1 Definition 328
32.2 Associative and commutative operations 329

33 Ordered groups - introduction 330
33.1 Ordered groups 330
33.2 Inequalities 337
33.3 The set of positive elements 348
33.4 Intervals and bounded sets 355

34 More on ordered groups 361
34.1 Absolute value and the triangle inequality 361
34.2 Maximum absolute value of a set 373
34.3 Alternative definitions 375
34.4 Odd Extensions 379
34.5 Functions with infinite limits 381

35 Rings - introduction 384
35.1 Definition and basic properties 384
35.2 Rearrangement lemmas 392

36 More on rings 395
36.1 The ring of classes of almost homomorphisms 395

37 Ordered rings 398
37.1 Definition and notation 398
37.2 Absolute value for ordered rings 406
37.3 Positivity in ordered rings 408

38 Cardinal numbers 415
38.1 Some new ideas on cardinals . . . 415
38.2 Main result on cardinals (without the Axiom of Choice) . . . 419
38.3 Choice axioms . . . 422

39 Groups 4 426
39.1 Conjugation of subgroups . . . 426
39.2 Finite groups . . . 431
39.3 Subgroups generated by sets . . . 435
39.4 Homomorphisms . . . 436
39.5 First isomorphism theorem . . . 442

40 Fields - introduction 449
40.1 Definition and basic properties . . . 449
40.2 Equations and identities . . . 452
40.3 1/0=0 . . . 454

41 Ordered fields 455
41.1 Definition and basic properties . . . 455
41.2 Inequalities . . . 459
41.3 Definition of real numbers . . . 462

42 Integers - introduction 462
42.1 Addition and multiplication as ZF-functions. . . . 462
42.2 Integers as an ordered group . . . 469
42.3 Induction on integers. . . . 482
42.4 Bounded vs. finite subsets of integers . . . 485

43 Integers 1 488
43.1 Integers as a ring . . . 488
43.2 Rearrangement lemmas . . . 491
43.3 Integers as an ordered ring . . . 497
43.4 Maximum and minimum of a set of integers . . . 508
43.5 The set of nonnegative integers . . . 512
43.6 Functions with infinite limits . . . 518
43.7 Miscelaneous . . . 523

44 Division on integers 525
44.1 Quotient and reminder . . . 525

45 Integers 2 527
45.1 Slopes . . . 527
45.2 Composing slopes . . . 549

46 Integers 3 555
46.1 Positive slopes 555
46.2 Inverting slopes 566
46.3 Completeness 574

47 Construction real numbers - the generic part 580
47.1 The definition of real numbers 581

48 Construction of real numbers 588
48.1 Definitions and notation 588
48.2 Multiplication of real numbers 591
48.3 The order on reals 594
48.4 Inverting reals 603
48.5 Completeness 607

49 Complex numbers 627
49.1 From complete ordered fields to complex numbers 627
49.2 Axioms of complex numbers 631

50 Topology - introduction 643
50.1 Basic definitions and properties 644
50.2 Interior of a set 648
50.3 Closed sets, closure, boundary. 649

51 Topology 1 654
51.1 Separation axioms. 654
51.2 Bases and subbases. 656
51.3 Product topology 660

52 Topology 1b 665
52.1 Compact sets are closed - no need for AC 665

53 Topology 2 667
53.1 Continuous functions. 667
53.2 Homeomorphisms 672
53.3 Topologies induced by mappings 674
53.4 Partial functions and continuity 676
53.5 Product topology and continuity 678
53.6 Pasting lemma 681

54 Topology 3 684
54.1 The base of the product topology 684
54.2 Finite product of topologies 687

55 Topology 4 695
55.1 Nets 695
55.2 Filters 698
55.3 Relation between nets and filters 704

56 Topology and neighborhoods 714
56.1 Neighborhood systems 715
56.2 Topology from neighborhood systems 716
56.3 Neighborhood system from topology 717

57 Topology - examples 719
57.1 CoCardinal Topology 719
57.2 Total set, Closed sets, Interior, Closure and Boundary 721
57.3 Excluded Set Topology 727
57.4 Total set, closed sets, interior, closure and boundary 728
57.5 Special cases and subspaces 731
57.6 Included Set Topology 733
57.7 Basic topological notions in included set topology 734
57.8 Special cases and subspaces 737

58 More examples in topology 738
58.1 New ideas using a base for a topology 739
58.2 The topology of a base 739
58.3 Dual Base for Closed Sets 743
58.4 Partition topology 745
58.5 Partition topology is a topology. 746
58.6 Total set, Closed sets, Interior, Closure and Boundary 747
58.7 Special cases and subspaces 752
58.8 Order topologies 754
58.9 Order topology is a topology 754
58.10 Total set 765
58.11 Right order and Left order topologies. 766
58.11.1 Right and Left Order topologies are topologies 766
58.11.2 Total set 768
58.12 Union of Topologies 768

59 Properties in Topology 770
59.1 Properties of compactness 770
59.2 Properties of numerability 774
59.3 Relations between numerability properties and choice principles 776
59.4 Relation between numerability and compactness 781

60 Topology 5 794
60.1 Some results for separation axioms 794
60.2 Hereditability 810
60.3 Spectrum and anti-properties 813

61 Topology 6 842
61.1 Image filter 842
61.2 Continuous at a point vs. globally continuous 844
61.3 Continuous functions and filters 845

62 Topology 7 847
62.1 Connection Properties 847

63 Topology 8 878
63.1 Definition of quotient topology 878
63.2 Quotient topologies from equivalence relations 880

64 Topology 9 887
64.1 Group of homeomorphisms 888
64.2 Examples computed 890
64.3 Properties preserved by functions 901

65 Topology 10 905
65.1 Closure and closed sets in product space 905
65.2 Separation properties in product space 908
65.3 Connection properties in product space 912

66 Topology 11 916
66.1 Order topologies 916
66.2 Separation properties 916
66.3 Connectedness properties 919
66.4 Numerability axioms 938

67 Properties in topology 2 946
67.1 Local properties. 947
67.2 First examples 947
67.3 Local compactness 948
67.4 Compactification by one point 956
67.5 Hereditary properties and local properties 966

68 Properties in Topology 3 998
68.1 More anti-properties 998
68.2 First examples 998
68.3 Structural results 999
68.4 More Separation properties 1009

68.5 Definitions 1009
68.6 First results 1010
68.7 Counter-examples 1016
68.8 Other types of properties 1054
68.9 Definitions 1054
68.10 First examples 1055
68.11 Structural results 1059

69 Metric spaces 1062
69.1 Pseudometric - definition and basic properties 1062

70 Uniform spaces 1066
70.1 Definition and motivation 1067

71 More on uniform spaces 1071
71.1 Uniformly continuous functions 1071

72 Topological groups - introduction 1072
72.1 Topological group: definition and notation 1072
72.2 Interval arithmetic, translations and inverse of set 1078
72.3 Neighborhoods of zero 1081
72.4 Closure in topological groups 1086
72.5 Sums of sequences of elements and subsets 1087

73 Topological groups 1 1090
73.1 Separation properties of topological groups 1090
73.2 Existence of nice neighbourhoods. 1093
73.3 Rest of separation axioms 1094
73.4 Local properties 1099

74 Topological groups - uniformity 1100
74.1 Natural uniformities in topological groups: definitions and notation 1100

75 Topological groups 2 1122
75.1 Quotients of topological groups 1122

76 Topological groups 3 1127
76.1 Subgroups topologies 1128

77 Metamath introduction 1137
77.1 Importing from Metamath - how is it done 1137
77.2 The context for Metamath theorems 1138

78 Logic and sets in Metamatah 1141
78.1 Basic Metamath theorems 1141

79 Complex numbers in Metamatah - introduction 1192

80 Metamath examples 1310

81 Metamath interface 1315
81.1 MMisar0 and complex0 contexts. 1315

82 Metamath sampler 1322
82.1 Extended reals and order 1322
82.2 Natural real numbers 1326
82.3 Infimum and supremum in real numbers 1328

1 Introduction to the IsarMathLib project

theory Introduction imports ZF.equalities

begin

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

1.1 How to read IsarMathLib proofs - a tutorial

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

The first thing that mathematicians typically do is to define notions. In Isar this is done with the definition keyword. In our case we define a notion of two sets being disjoint. We will use the infix notation, i.e. the string –is disjoint

with″ put between two sets to denote our notion of disjointness. The left side of the ≡ symbol is the notion being defined, the right side says how we define it. In Isabelle/ZF 0 is used to denote both zero (of natural numbers) and the empty set, which is not surprising as those two things are the same in set theory.

```
definition
  AreDisjoint (infix –is disjoint with″ 90) where
  A –is disjoint with″ B ≡ A ∩ B = 0
```

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

```
theorem disjointness˙symmetric:
  shows A –is disjoint with″ B ⟷ B –is disjoint with″ A
proof -
  have A –is disjoint with″ B ⟶ B –is disjoint with″ A
  proof -
    { assume A –is disjoint with″ B
      then have A ∩ B = 0 using AreDisjoint˙def by simp
      hence B ∩ A = 0 by auto
      then have  B –is disjoint with″ A
        using AreDisjoint˙def by simp
```

```
    } thus thesis by simp
  qed
  moreover have B –is disjoint with″ A ⟶ A –is disjoint with″ B
  proof -
    { assume B –is disjoint with″ A
      then have B ∩ A = 0 using AreDisjoint˙def by simp
      hence A ∩ B = 0 by auto
      then have  A –is disjoint with″ B
        using AreDisjoint˙def by simp
    } thus thesis by simp
  qed
  ultimately show thesis by blast
qed
```

1.2 Overview of the project

The Fol1, ZF1 and Nat˙ZF˙IML theory files contain some background material that is needed for the remaining theories.

Order˙ZF and Order˙ZF˙1a reformulate material from standard Isabelle's Order theory in terms of non-strict (less-or-equal) order relations. Order˙ZF˙1 on the other hand directly continues the Order theory file using strict order relations (less and not equal). This is useful for translating theorems from Metamath.

In NatOrder˙ZF we prove that the usual order on natural numbers is linear.

The func1 theory provides basic facts about functions. func˙ZF continues this development with more advanced topics that relate to algebraic properties of binary operations, like lifting a binary operation to a function space, associative, commutative and distributive operations and properties of functions related to order relations. func˙ZF˙1 is about properties of functions related to order relations.

The standard Isabelle's Finite theory defines the finite powerset of a set as a certain "datatype" (?) with some recursive properties. IsarMathLib's Finite1 and Finite˙ZF˙1 theories develop more facts about this notion. These two theories are obsolete now. They will be gradually replaced by an approach based on set theory rather than tools specific to Isabelle. This approach is presented in Finite˙ZF theory file.

In FinOrd˙ZF we talk about ordered finite sets.

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

FiniteSeq˙ZF discusses the notion of finite sequences (a.k.a. lists).

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

Fold˙ZF shows how the familiar from functional programming notion of fold

can be interpreted in set theory.

Partitions˙ZF is about splitting a set into non-overlapping subsets. This is a common trick in proofs.

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

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

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

Group˙ZF, Group˙ZF˙1, Group˙ZF˙1b and Group˙ZF˙2 provide basic facts of the group theory. Group˙ZF˙3 considers the notion of almost homomorphisms that is nedeed for the real numbers construction in Real˙ZF.

The TopologicalGroup connects the Topology˙ZF and Group˙ZF series and starts the subject of topological groups with some basic definitions and facts.

In DirectProduct˙ZF we define direct product of groups and show some its basic properties.

The OrderedGroup˙ZF theory treats ordered groups. This is a suprisingly large theory for such relatively obscure topic.

Ring˙ZF defines rings. Ring˙ZF˙1 covers the properties of rings that are specific to the real numbers construction in Real˙ZF.

The OrderedRing˙ZF theory looks at the consequences of adding a linear order to the ring algebraic structure.

Field˙ZF and OrderedField˙ZF contain basic facts about (you guessed it) fields and ordered fields.

Int˙ZF˙IML theory considers the integers as a monoid (multiplication) and an abelian ordered group (addition). In Int˙ZF˙1 we show that integers form a commutative ring. Int˙ZF˙2 contains some facts about slopes (almost homomorphisms on integers) needed for real numbers construction, used in Real˙ZF˙1.

In the IntDiv˙ZF˙IML theory we translate some properties of the integer quotient and reminder functions studied in the standard Isabelle's IntDiv˙ZF theory to the notation used in IsarMathLib.

The Real˙ZF and Real˙ZF˙1 theories contain the construction of real numbers based on the paper [2] by R. D. Arthan (not Cauchy sequences, not Dedekind sections). The heavy lifting is done mostly in Group˙ZF˙3, Ring˙ZF˙1 and Int˙ZF˙2. Real˙ZF contains the part of the construction that can be done starting from generic abelian groups (rather than additive group of integers). This allows to show that real numbers form a ring. Real˙ZF˙1 continues the construction using properties specific to the integers and showing that real

numbers constructed this way form a complete ordered field.

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

Group˙ZF˙4 considers conjugate of subgroup and defines simple groups. A nice theorem here is that endomorphisms of an abelian group form a ring. The first isomorphism theorem (a group homomorphism h induces an isomorphism between the group divided by the kernel of h and the image of h) is proven.

Turns out given a property of a topological space one can define a local version of a property in general. This is studied in the the Topology˙ZF˙properties˙2 theory and applied to local versions of the property of being finite or compact or Hausdorff (i.e. locally finite, locally compact, locally Hausdorff). There are a couple of nice applications, like one-point compactification that allows to show that every locally compact Hausdorff space is regular. Also there are some results on the interplay between hereditability of a property and local properties.

For a given surjection $f : X \to Y$, where X is a topological space one can consider the weakest topology on Y which makes f continuous, let's call it a quotient topology generated by f. The quotient topology generated by an equivalence relation r on X is actually a special case of this setup, where f is the natural projection of X on the quotient X/r. The properties of these two ways of getting new topologies are studied in Topology˙ZF˙8 theory. The main result is that any quotient topology generated by a function is homeomorphic to a topology given by an equivalence relation, so these two approaches to quotient topologies are kind of equivalent.

As we all know, automorphisms of a topological space form a group. This fact is proven in Topology˙ZF˙9 and the automorphism groups for co-cardinal, included-set, and excluded-set topologies are identified. For order topologies it is shown that order isomorphisms are homeomorphisms of the topology induced by the order. Properties preserved by continuous functions are studied and as an application it is shown for example that quotient topological spaces of compact (or connected) spaces are compact (or connected, resp.)

The Topology˙ZF˙10 theory is about products of two topological spaces. It is proven that if two spaces are T_0 (or T_1, T_2, regular, connected) then their product is as well.

Given a total order on a set one can define a natural topology on it generated by taking the rays and intervals as the base. The Topology˙ZF˙11 theory studies relations between the order and various properties of generated topology. For example one can show that if the order topology is connected, then the order is complete (in the sense that for each set bounded from above the set of upper bounds has a minimum). For a given cardinal κ we can consider generalized notion of $\kappa - separability$. Turns out κ-separability is related to (order) density of sets of cardinality κ for order topologies.

Being a topological group imposes additional structure on the topology of the group, in particular its separation properties. In Topological˙Group˙ZF˙1.thy theory it is shown that if a topology is T_0, then it must be T_3 , and that the topology in a topological group is always regular.

For a given normal subgroup of a topological group we can define a topology on the quotient group in a natural way. At the end of the Topological˙Group˙ZF˙2.thy theory it is shown that such topology on the quotient group makes it a topological group.

The Topological˙Group˙ZF˙3.thy theory studies the topologies on subgroups of a topological group. A couple of nice basic properties are shown, like that the closure of a subgroup is a subgroup, closure of a normal subgroup is normal and, a bit more surprising (to me) property that every locally-compact subgroup of a T_0 group is closed.

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

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

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

The theories MMI˙logic˙and˙sets, MMI˙Complex, MMI˙Complex˙1 and MMI˙Complex˙2 contain the theorems imported from the Metamath's set.mm database. As the translated proofs are rather verbose these theories are not printed in this proof document. The full list of translated facts can be found in the Metamath˙theorems.txt file included in the IsarMathLib distribution. The MMI˙examples provides some theorems imported from Metamath that are printed in this proof document as examples of how translated proofs look like.

end

2 First Order Logic

theory Fol1 imports ZF.Trancl

begin

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

2.1 Notions and lemmas in FOL

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

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

```
lemma Fol1˙L2: assumes
  A1: ∀ x y z. ⟨x, y⟩ ∈ r ∧ ⟨y, z⟩ ∈ r ⟶ ⟨x, z⟩ ∈ r
  shows trans(r)
proof -
  from A1 have
    ∀ x y z. ⟨x, y⟩ ∈ r ⟶ ⟨y, z⟩ ∈ r ⟶ ⟨x, z⟩ ∈ r
    using imp˙conj by blast
  then show thesis unfolding trans˙def by blast
qed
```

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

```
lemma Fol1˙L3: assumes A1: trans(r) and A2: ⟨ a,b⟩ ∈ r  ∧ ⟨ b,c⟩ ∈ r
  shows ⟨ a,c⟩ ∈ r
proof -
  from A1 have  ∀x y z. ⟨x, y⟩ ∈ r ⟶ ⟨y, z⟩ ∈ r ⟶ ⟨x, z⟩ ∈ r
   unfolding trans˙def by blast
  with A2 show thesis using imp˙conj by fast
qed
```

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

```
lemma Fol1˙L4:
  assumes A1: antisym(r) and A2: ⟨ a,b⟩ ∈ r   ⟨ b,a⟩ ∈ r
  shows a=b
```

```
proof -
  from A1 have ∀ x y. ⟨ x,y⟩ ∈ r ⟶ ⟨ y,x⟩ ∈ r ⟶ x=y
    unfolding antisym˙def by blast
  with A2 show a=b using imp˙conj by fast
qed
```

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

```
definition
  Exactly˙1˙of˙3˙holds(p,q,r) ≡
  (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
```

The next lemma allows to prove statements of the form Exactly˙1˙of˙3˙holds(p,q,r).

```
lemma Fol1˙L5:
  assumes p∨q∨r
  and p ⟶ ¬q ∧ ¬r
  and q ⟶ ¬p ∧ ¬r
  and r ⟶ ¬p ∧ ¬q
  shows Exactly˙1˙of˙3˙holds(p,q,r)
proof -
  from assms have
    (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
    by blast
  then show Exactly˙1˙of˙3˙holds (p,q,r)
    unfolding Exactly˙1˙of˙3˙holds˙def by fast
qed
```

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

```
lemma Fol1˙L6:
  assumes A1: ¬p and A2: Exactly˙1˙of˙3˙holds(p,q,r)
  shows q∨r
proof -
  from A2 have
    (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
    unfolding Exactly˙1˙of˙3˙holds˙def by fast
  hence p ∨ q ∨ r by blast
  with A1 show q ∨ r by simp
qed
```

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

```
lemma Fol1˙L7:
  assumes A1: q and A2: Exactly˙1˙of˙3˙holds(p,q,r)
  shows ¬r
proof -
   from A2 have
    (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
    unfolding Exactly˙1˙of˙3˙holds˙def by fast
  with A1 show ¬r by blast
```

qed

The next lemma demonstrates an elegant form of the Exactly˙1˙of˙3˙holds(p,q,r) predicate.

```
lemma Fol1˙L8:
  shows Exactly˙1˙of˙3˙holds(p,q,r) ⟷ (p⟷q⟷r) ∧ ¬(p∧q∧r)
proof
  assume Exactly˙1˙of˙3˙holds(p,q,r)
  then have
    (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
    unfolding Exactly˙1˙of˙3˙holds˙def by fast
  thus (p⟷q⟷r) ∧ ¬(p∧q∧r) by blast
next assume (p⟷q⟷r) ∧ ¬(p∧q∧r)
  hence
    (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
    by auto
  then show Exactly˙1˙of˙3˙holds(p,q,r)
    unfolding Exactly˙1˙of˙3˙holds˙def by fast
qed
```

A property of the Exactly˙1˙of˙3˙holds predicate.

```
lemma Fol1˙L8A: assumes A1: Exactly˙1˙of˙3˙holds(p,q,r)
  shows p ⟷ ¬(q ∨ r)
proof -
  from A1 have (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
    unfolding Exactly˙1˙of˙3˙holds˙def by fast
  then show p ⟷ ¬(q ∨ r) by blast
qed
```

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

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

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

```
lemma Fol1˙L9: shows p Xor q ⟷ ¬(p⟷q)
  using Xor˙def by auto
```

Equivalence relations are symmetric.

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

qed

end

3 ZF set theory basics

theory ZF1 imports ZF.equalities

begin

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

3.1 Lemmas in Zermelo-Fraenkel set theory

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

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

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

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 assms by blast
```

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

```
lemma ZF1˙1˙L2: assumes A1:C≠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
```

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

```
lemma ZF1˙1˙L4: assumes A1: ∀x∈X.∀y∈Y. a(x,y) = b(x,y)
  shows –a(x,y). ⟨x,y⟩ ∈ X×Y″ = –b(x,y). ⟨x,y⟩ ∈ X×Y″
proof
  show –a(x, y). ⟨x,y⟩ ∈ X × Y″ ⊆ –b(x, y). ⟨x,y⟩ ∈ X × Y″
  proof
    fix z assume z ∈ –a(x, y) . ⟨x,y⟩ ∈ X × Y″
    with A1 show  z ∈ –b(x,y).⟨x,y⟩ ∈ X×Y″ by auto
  qed
  show –b(x, y). ⟨x,y⟩ ∈ X × Y″ ⊆ –a(x, y). ⟨x,y⟩ ∈ X × Y″
  proof
    fix z assume z ∈ –b(x, y). ⟨x,y⟩ ∈ X × Y″
    with A1 show z ∈ –a(x,y).⟨x,y⟩ ∈ X×Y″ by auto
  qed
qed
```

If two meta-functions are the same on a cartesian product, then the subsets defined by them are the same. This is similar to ZF1˙1˙L4, except that the set definition varies over p∈X×Y rather than ⟨ x,y⟩∈X×Y.

```
lemma ZF1˙1˙L4A: assumes A1: ∀x∈X.∀y∈Y. a(⟨ x,y⟩) = b(x,y)
  shows –a(p). p ∈ X×Y″ = –b(x,y). ⟨x,y⟩ ∈ X×Y″
proof
  { fix z assume z ∈ –a(p). p∈X×Y″
    then obtain p where D1: z=a(p) p∈X×Y by auto
    let x = fst(p) let y = snd(p)
    from A1 D1 have z ∈ –b(x,y). ⟨x,y⟩ ∈ X×Y″ by auto
  } then show –a(p). p ∈ X×Y″ ⊆ –b(x,y). ⟨x,y⟩ ∈ X×Y″ by blast
next
  { fix z assume z ∈ –b(x,y). ⟨x,y⟩ ∈ X×Y″
    then obtain x y where D1: ⟨x,y⟩ ∈ X×Y z=b(x,y) by auto
    let p = ⟨ x,y⟩
    from A1 D1 have p∈X×Y z = a(p) by auto
    then have z ∈ –a(p). p ∈ X×Y″ by auto
  } then show –b(x,y). ⟨x,y⟩ ∈ X×Y″ ⊆ –a(p). p ∈ X×Y″ by blast
qed
```

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

```
lemma prod˙subset: assumes U×V≠0 U×V ⊆ X×Y shows U⊆X and V⊆Y
  using assms by auto
```

A technical lemma about sections in cartesian products.

```
lemma section˙proj: assumes A ⊆ X×Y and U×V ⊆ A and x ∈ U  y ∈ V
  shows U ⊆ –t∈X. ⟨t,y⟩ ∈ A″ and V ⊆ –t∈Y. ⟨x,t⟩ ∈ A″
  using assms by auto
```

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

```
lemma ZF1˙1˙L4B: assumes ∀x∈X. a(x) = b(x)
```

shows –a(x). x∈X″ = –b(x). x∈X″
using assms by simp

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

lemma ZF1˙1˙L5: assumes X≠0 and ∀x∈X. b(x) = c
shows –b(x). x∈X″ = –c″ using assms by blast

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

lemma subset˙with˙property: assumes Y = –x∈X. b(x)″
shows Y ⊆ X
using assms by auto

We can choose an element from a nonempty set.

lemma nonempty˙has˙element: assumes X≠0 shows ∃x. x∈X
using assms by auto

In Isabelle/ZF the intersection of an empty family is empty. This is exactly lemma Inter˙0 from Isabelle's equalities theory. We repeat this lemma here as it is very difficult to find. This is one reason we need comments before every theorem: so that we can search for keywords.

lemma inter˙empty˙empty: shows ⋂0 = 0 by (rule Inter˙0)

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

lemma inter˙nempty˙nempty: assumes ⋂A ≠ 0 shows A≠0
using assms by auto

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

definition
ProductCollection(T,S) ≡ ⋃U∈T.–U×V. V∈S″

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

lemma ZF1˙1˙L6: shows ⋃ ProductCollection(S,T) = ⋃S × ⋃T
using ProductCollection˙def by auto

An intersection of subsets is a subset.

lemma ZF1˙1˙L7: assumes A1: I≠0 and A2: ∀i∈I. P(i) ⊆ X
shows (⋂i∈I. P(i)) ⊆ X
proof -
from A1 obtain i_0 where i_0 ∈ I by auto
with A2 have (⋂i∈I. P(i)) ⊆ P(i_0) and P(i_0) ⊆ X
by auto

```
  thus ( ⋂i∈I. P(i) ) ⊆ X by auto
qed
```

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

```
lemma ZF1˙1˙L8: shows ⋃ –x″ = x by auto
```

Some properties of singletons.

```
lemma ZF1˙1˙L9: assumes A1: ∃! x. x∈A ∧ φ(x)
  shows
  ∃a. –x∈A. φ(x)″ = –a″
  ⋃ –x∈A. φ(x)″ ∈ A
  φ(⋃ –x∈A. φ(x)″)
proof -
  from A1 show ∃a. –x∈A. φ(x)″ = –a″ by auto
  then obtain a where I: –x∈A. φ(x)″ = –a″ by auto
  then have ⋃ –x∈A. φ(x)″ = a by auto
  moreover
  from I have a ∈ –x∈A. φ(x)″ by simp
  hence a∈A and φ(a) by auto
  ultimately show ⋃ –x∈A. φ(x)″ ∈ A and φ(⋃ –x∈A. φ(x)″)
    by auto
qed
```

A simple version of ZF1˙1˙L9.

```
corollary singleton˙extract: assumes  ∃! x. x∈A
  shows (⋃ A) ∈ A
proof -
  from assms have ∃! x. x∈A ∧ True by simp
  then have ⋃ –x∈A. True″ ∈ A by (rule ZF1˙1˙L9)
  thus (⋃ A) ∈ A by simp
qed
```

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

```
lemma singleton˙comprehension:
  assumes A1: y∈X and A2: ∀x∈X. ∀y∈X. P(x) = P(y)
  shows (⋃–P(x). x∈X″) = P(y)
proof -
  let A = –P(x). x∈X″
  have ∃! c. c ∈ A
  proof
    from A1 show ∃c. c ∈ A by auto
  next
    fix a b assume a ∈ A and b ∈ A
    then obtain x t where
      x ∈ X a = P(x) and t ∈ X b = P(t)
      by auto
```

```
    with A2 show a=b by blast
  qed
  then have (⋃A) ∈ A by (rule singleton˙extract)
  then obtain x where x ∈ X and (⋃A) = P(x)
    by auto
  from A1 A2 ‹x ∈ X› have P(x) = P(y)
    by blast
  with ‹(⋃A) = P(x)› show (⋃A) = P(y) by simp
qed
```

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

```
lemma set˙elem˙add: assumes x∈X shows X ∪ –x″ = X using assms
  by auto
```

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

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

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

```
lemma union˙restrict:
  shows ⋃(M –restricted to″ X) = (⋃M) ∩ X
  using RestrictedTo˙def by auto
```

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

```
lemma ZF1˙1˙L10: assumes A1: ∀U∈C. ∃A∈B. U = ⋃A
  shows ⋃⋃ –⋃–A∈B. U = ⋃A″. U∈C″ = ⋃C
proof
  show ⋃(⋃U∈C. ⋃–A ∈ B . U = ⋃A″) ⊆ ⋃C by blast
  show ⋃C ⊆ ⋃(⋃U∈C. ⋃–A ∈ B . U = ⋃A″)
  proof
    fix x assume x ∈ ⋃C
    show x ∈ ⋃(⋃U∈C. ⋃–A ∈ B . U = ⋃A″)
    proof -
      from ‹x ∈ ⋃C› obtain U where U∈C ∧ x∈U by auto
      with A1 obtain A where A∈B ∧ U = ⋃A by auto
      from ‹U∈C ∧ x∈U› ‹A∈B ∧ U = ⋃A› show x∈ ⋃(⋃U∈C. ⋃–A ∈ B . U =
⋃A″)
 by auto
    qed
  qed
qed
```

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

```
lemma consdef: shows cons(a,A) = A ∪ –a″
```

using cons˙def by auto

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

lemma singl˙diff˙empty: assumes A - –x″ = 0
shows A = 0 ∨ A = –x″
using assms by auto

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

lemma singl˙diff˙eq: assumes A1: A - –x″ = A
shows x ∉ A
proof -
have x ∉ A - –x″ by auto
with A1 show x ∉ A by simp
qed

A basic property of sets defined by comprehension.

lemma comprehension: assumes a ∈ –x∈X. p(x)″
shows a∈X and p(a) using assms by auto

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

lemma image˙rel˙mono: assumes r⊆s shows r(A) ⊆ s(A)
using assms by auto

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

lemma image˙greater˙rel:
assumes x ∈ U–x″ and U–x″ ⊆ C
shows (U ∪ C×C)–x″ = C
using assms image˙Un˙left by blast

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

lemma set˙mem˙eq: assumes x∈A A=B shows x∈B using assms by simp

end

4 Natural numbers in IsarMathLib

theory Nat˙ZF˙IML imports ZF.Arith

begin

The ZF set theory constructs natural numbers from the empty set and the notion of a one-element set. Namely, zero of natural numbers is defined as the empty set. For each natural number n the next natural number is

defined as $n \cup \{n\}$. With this definition for every non-zero natural number we get the identity $n = \{0, 1, 2, .., n-1\}$. It is good to remember that when we see an expression like $f : n \to X$. Also, with this definition the relation "less or equal than" becomes "$\subseteq$" and the relation "less than" becomes "$\in$".

4.1 Induction

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

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

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

A nonzero natural number has a predecessor.

```
lemma Nat˙ZF˙1˙L3: assumes A1: n ∈ nat and A2: n≠0
  shows ∃k∈nat. n = succ(k)
proof -
  from A1 have n ∈ –0″ ∪ –succ(k). k∈nat″
    using nat˙unfold by simp
  with A2 show thesis by simp
qed
```

What is succ, anyway?

```
lemma succ˙explained: shows succ(n) = n ∪ –n″
  using succ˙iff by auto
```

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

```
lemma empty˙in˙every˙succ: assumes A1: n ∈ nat
  shows 0 ∈ succ(n)
proof -
  note A1
```

```
  moreover have 0 ∈ succ(0) by simp
  moreover
  { fix k assume k ∈ nat and A2: 0 ∈ succ(k)
    then have succ(k) ⊆ succ(succ(k)) by auto
    with A2 have 0 ∈ succ(succ(k)) by auto
  } then have ∀k ∈ nat. 0 ∈ succ(k) ⟶ 0 ∈ succ(succ(k))
    by simp
  ultimately show 0 ∈ succ(n) by (rule ind˙on˙nat)
qed
```

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

```
lemma succ˙ineq: assumes A1: n ∈ nat
  shows ∀i ∈ n. succ(i) ∈ succ(n)
proof -
  note A1
  moreover have ∀k ∈ 0. succ(k) ∈ succ(0) by simp
  moreover
  { fix k assume A2: ∀i∈k. succ(i) ∈ succ(k)
    { fix i assume i ∈ succ(k)
      then have i ∈ k ∨ i = k by auto
      moreover
      { assume i∈k
 with A2 have succ(i) ∈ succ(k) by simp
 hence succ(i) ∈ succ(succ(k)) by auto }
      moreover
      { assume i = k
 then have succ(i) ∈ succ(succ(k)) by auto }
      ultimately have succ(i) ∈ succ(succ(k)) by auto
    } then have ∀i ∈ succ(k). succ(i) ∈ succ(succ(k))
      by simp
  } then have ∀k ∈ nat.
      ( (∀i∈k. succ(i) ∈ succ(k)) ⟶ (∀i ∈ succ(k). succ(i) ∈ succ(succ(k))) )
    by simp
  ultimately show ∀i ∈ n. succ(i) ∈ succ(n) by (rule ind˙on˙nat)
qed
```

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

```
lemma succ˙subset: assumes A1: k ∈ nat  n ∈ nat and A2: k⊆n
  shows succ(k) ⊆ succ(n)
proof -
  from A1 have T: Ord(k) and Ord(n)
    using nat˙into˙Ord by auto
  with A2 have succ(k) ≤ succ(n)
    using subset˙imp˙le by simp
  then show succ(k) ⊆ succ(n) using le˙imp˙subset
    by simp
qed
```

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

```
lemma nat˙incl˙total: assumes A1: i ∈ nat  j ∈ nat
  shows i ⊆ j ∨ j ⊆ i
proof -
  from A1 have T: Ord(i)   Ord(j)
    using nat˙into˙Ord by auto
  then have i∈j ∨ i=j ∨ j∈i using Ord˙linear
    by simp
  moreover
  { assume i∈j
    with T have i⊆j ∨ j⊆i
      using lt˙def leI le˙imp˙subset by simp }
  moreover
  { assume i=j
    then have i⊆j ∨ j⊆i by simp }
  moreover
  { assume j∈i
    with T have i⊆j ∨ j⊆i
      using lt˙def leI  le˙imp˙subset by simp }
  ultimately show i ⊆ j ∨ j ⊆ i by auto
qed
```

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

```
lemma nat˙union˙succ: shows nat = (⋃n ∈ nat. succ(n))
proof
  show nat ⊆ (⋃n ∈ nat. succ(n)) by auto
next
  { fix k assume A2: k ∈ (⋃n ∈ nat. succ(n))
    then obtain n where T: n ∈ nat and I: k ∈ succ(n)
      by auto
    then have k ≤ n using nat˙into˙Ord lt˙def
      by simp
    with T have k ∈ nat using le˙in˙nat by simp
  } then show  (⋃n ∈ nat. succ(n)) ⊆ nat by auto
qed
```

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

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

Element of a natural number is a natural number.

```
lemma elem˙nat˙is˙nat: assumes A1: n ∈ nat  and A2: k∈n
  shows k ¡ n  k ∈ nat  k ≤ n  ⟨k,n⟩ ∈ Le
proof -
  from A1 A2 show k ¡ n using nat˙into˙Ord lt˙def by simp
```

```
  with A1 show k ∈ nat using lt˙nat˙in˙nat by simp
  from ‹k ¡ n› show k ≤ n using leI by simp
  with A1 ‹k ∈ nat› show ⟨k,n⟩ ∈ Le using Le˙def
    by simp
qed
```

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

```
lemma nat˙union˙nat: shows nat = ⋃ nat
  using elem˙nat˙is˙nat by blast
```

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

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

Adding natural numbers does not decrease what we add to.

```
lemma add˙nat˙le: assumes A1: n ∈ nat  and A2: k ∈ nat
  shows
  n ≤ n #+ k
  n ⊆ n #+ k
  n ⊆ k #+ n
proof -
  from A1 A2 have n ≤ n  0 ≤ k  n ∈ nat  k ∈ nat
    using nat˙le˙refl nat˙0˙le by auto
  then have n #+ 0 ≤ n #+ k by (rule add˙le˙mono)
  with A1 show n ≤ n #+ k using add˙0˙right by simp
  then show n ⊆ n #+ k using le˙imp˙subset by simp
  then show n ⊆ k #+ n using add˙commute by simp
qed
```

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

```
lemma add˙lt˙mono:
  assumes k ∈ nat and j∈k
  shows
  (n #+ j) ¡ (n #+ k)
  (n #+ j) ∈ (n #+ k)
proof -
  from assms have j ¡ k using elem˙nat˙is˙nat by blast
  moreover note ‹k ∈ nat›
  ultimately show (n #+ j) ¡ (n #+ k)   (n #+ j) ∈ (n #+ k)
    using add˙lt˙mono2 ltD by auto
qed
```

A technical lemma about a decomposition of a sum of two natural numbers: if a number i is from $m + n$ then it is either from m or can be written as a sum of m and a number from n. The proof by induction w.r.t. to m seems

to be a bit heavy-handed, but I could not figure out how to do this directly from results from standard Isabelle/ZF.

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

A variant of induction useful for finite sequences.

```
lemma fin˙nat˙ind: assumes A1: n ∈ nat and A2: k ∈ succ(n)
  and A3: P(0) and A4: ∀j∈n. P(j)  ⟶ P(succ(j))
  shows P(k)
proof -
  from A2 have k ∈ n ∨ k=n by auto
  with A1 have k ∈ nat using elem˙nat˙is˙nat by blast
  moreover from A3 have 0 ∈ succ(n) ⟶ P(0) by simp
  moreover from A1 A4 have
    ∀k ∈ nat. (k ∈ succ(n) ⟶ P(k)) ⟶ (succ(k) ∈ succ(n) ⟶ P(succ(k)))
    using nat˙into˙Ord Ord˙succ˙mem˙iff by auto
  ultimately have k ∈ succ(n) ⟶ P(k)
    by (rule ind˙on˙nat)
  with A2 show P(k) by simp
qed
```

Some properties of positive natural numbers.

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

4.2 Intervals

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

The interval is determined by two parameters: starting point and length. Recall that in standard Isabelle's Arith.thy the symbol #+ is defined as the sum of natural numbers.

```
definition

  NatInterval(n,k) ≡ –n #+ j. j∈k″
```

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

```
lemma inter˙diff˙in˙len:
  assumes A1: k ∈ nat and A2: i ∈ NatInterval(n,k)
  shows i #- n ∈ k
proof -
  from A2 obtain j where I: i = n #+ j and II: j ∈ k
    using NatInterval˙def by auto
  from A1 II have j ∈ nat using elem˙nat˙is˙nat by blast
  moreover from I have i #- n = natify(j) using diff˙add˙inverse
    by simp
  ultimately have i #- n = j by simp
  with II show thesis by simp
qed
```

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

```
lemma length˙start˙decomp: assumes  A1: n ∈ nat  k ∈ nat
  shows
  n ∩ NatInterval(n,k) = 0
  n ∪ NatInterval(n,k) = n #+ k
proof -
  { fix i assume A2: i ∈ n and i ∈ NatInterval(n,k)
    then obtain j where I: i = n #+ j and II: j ∈ k
      using NatInterval˙def by auto
    from A1 have k ∈ nat using elem˙nat˙is˙nat by blast
```

```
    with II have j ∈ nat using elem˙nat˙is˙nat by blast
    with A1 I have n ≤ i using add˙nat˙le by simp
    moreover from A1 A2 have i ¡ n using elem˙nat˙is˙nat by blast
    ultimately have False using le˙imp˙not˙lt by blast
  } thus n ∩ NatInterval(n,k) = 0 by auto
  from A1 have n ⊆ n #+ k using add˙nat˙le by simp
  moreover
  { fix i assume i ∈ NatInterval(n,k)
    then obtain j where III: i = n #+ j and IV: j ∈ k
      using NatInterval˙def by auto
    with A1 have j ¡ k using elem˙nat˙is˙nat by blast
    with A1 III have i ∈ n #+ k using add˙lt˙mono2 ltD
      by simp }
  ultimately have n ∪ NatInterval(n,k) ⊆ n #+ k by auto
  moreover from A1 have n #+ k ⊆ n ∪ NatInterval(n,k)
    using nat˙sum˙decomp NatInterval˙def by auto
  ultimately show n ∪ NatInterval(n,k) = n #+ k by auto
qed
```

Sme properties of three adjacent intervals.

```
lemma adjacent˙intervals3: assumes n ∈ nat  k ∈ nat  m ∈ nat
  shows
  n #+ k #+ m = (n #+ k) ∪ NatInterval(n #+ k,m)
  n #+ k #+ m = n ∪ NatInterval(n,k #+  m)
  n #+ k #+ m = n ∪ NatInterval(n,k) ∪ NatInterval(n #+ k,m)
  using assms add˙assoc length˙start˙decomp by auto

end
```

5 Order relations - introduction

```
theory Order˙ZF imports Fol1

begin
```

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

5.1 Definitions

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

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

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

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

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

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

definition
IsLinOrder(X,r) ≡ (antisym(r) ∧ trans(r) ∧ (r –is total on″ X))

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

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

We define sets bounded below analogously.

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

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

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

The notation for the definition of an interval may be mysterious for some readers, see lemma Order˙ZF˙2˙L1 for more intuitive notation.

definition
Interval(r,a,b) ≡ r–a″ ∩ r-–b″

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

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

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

definition

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

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

definition
HasAmaximum(r,A) ≡ ∃M∈A.∀x∈A. ⟨x,M⟩ ∈ r

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

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

Definition of the maximum of a set.

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

Definition of a minimum of a set.

definition
Minimum(r,A) ≡ THE m. m∈A ∧ (∀x∈A. ⟨m,x⟩ ∈ r)

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

definition
Supremum(r,A) ≡ Minimum(r,⋂a∈A. r–a″)

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

definition
HasAsupremum(r,A) ≡ HasAminimum(r,⋂a∈A. r–a″)

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

definition
HasAnInfimum(r,A) ≡ HasAmaximum(r,⋂a∈A. r-–a″)

Infimum is defined analogously.

definition
Infimum(r,A) ≡ Maximum(r,⋂a∈A. r-–a″)

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

definition
IsComplete (˙ –is complete″) where
r –is complete″ ≡
∀ A. IsBoundedAbove(A,r) ∧ A≠0 ⟶ HasAminimum(r,⋂a∈A. r–a″)

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

lemma Order˙ZF˙1˙L1: assumes r –is total on″ X and a∈X
shows ⟨a,a⟩ ∈ r using assms IsTotal˙def by auto

A total relation is reflexive.

lemma total˙is˙refl:
assumes r –is total on″ X
shows refl(X,r) using assms Order˙ZF˙1˙L1 refl˙def by simp

A linear order is partial order.

lemma Order˙ZF˙1˙L2: assumes IsLinOrder(X,r)
shows IsPartOrder(X,r)
using assms IsLinOrder˙def IsPartOrder˙def refl˙def Order˙ZF˙1˙L1
by auto

Partial order that is total is linear.

lemma Order˙ZF˙1˙L3:
assumes IsPartOrder(X,r) and r –is total on″ X
shows IsLinOrder(X,r)
using assms IsPartOrder˙def IsLinOrder˙def
by simp

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

lemma Order˙ZF˙1˙L4: assumes r –is total on″ X and A⊆X
shows r –is total on″ A
using assms IsTotal˙def by auto

A linear relation is linear on any subset.

lemma ord˙linear˙subset: assumes IsLinOrder(X,r) and A⊆X
shows IsLinOrder(A,r)
using assms IsLinOrder˙def Order˙ZF˙1˙L4 by blast

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

lemma Order˙ZF˙1˙L5:
assumes r –is total on″ X and A⊆X and a∈X
shows A = –x∈A. ⟨x,a⟩ ∈ r″ ∪ –x∈A. ⟨a,x⟩ ∈ r″
using assms IsTotal˙def by auto

A technical fact about reflexive relations.

```
lemma refl˙add˙point:
  assumes refl(X,r) and A ⊆ B ∪ −x″ and B ⊆ X and
  x ∈ X and ∀y∈B. ⟨y,x⟩ ∈ r
  shows ∀a∈A. ⟨a,x⟩ ∈ r
  using assms refl˙def by auto
```

5.2 Intervals

In this section we discuss intervals.

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

```
lemma Order˙ZF˙2˙L1:
  shows x ∈ Interval(r,a,b) ⟷ ⟨ a,x⟩ ∈ r ∧ ⟨ x,b⟩ ∈ r
  using Interval˙def by auto
```

Since there are some problems with applying the above lemma (seems that simp and auto don't handle equivalence very well), we split Order˙ZF˙2˙L1 into two lemmas.

```
lemma Order˙ZF˙2˙L1A: assumes x ∈ Interval(r,a,b)
  shows ⟨ a,x⟩ ∈ r  ⟨ x,b⟩ ∈ r
  using assms  Order˙ZF˙2˙L1 by auto
```

Order˙ZF˙2˙L1, implication from right to left.

```
lemma Order˙ZF˙2˙L1B: assumes ⟨ a,x⟩ ∈ r  ⟨ x,b⟩ ∈ r
  shows x ∈ Interval(r,a,b)
  using assms Order˙ZF˙2˙L1 by simp
```

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

```
lemma Order˙ZF˙2˙L2: assumes refl(X,r)
  and a∈X  b∈X and ⟨ a,b⟩ ∈ r
  shows
  a ∈ Interval(r,a,b)
  b ∈ Interval(r,a,b)
  using assms refl˙def Order˙ZF˙2˙L1 by auto
```

Under the assumptions of Order˙ZF˙2˙L2, the interval is nonempty.

```
lemma Order˙ZF˙2˙L2A: assumes refl(X,r)
  and a∈X  b∈X and ⟨ a,b⟩ ∈ r
  shows Interval(r,a,b) ≠ 0
proof -
  from assms have a ∈ Interval(r,a,b)
    using Order˙ZF˙2˙L2 by simp
  then show Interval(r,a,b) ≠ 0 by auto
qed
```

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

```
lemma Order˙ZF˙2˙L3:
  assumes A1: trans(r) and A2:⟨ a,b⟩∈r  ⟨ b,c⟩∈r  ⟨ c,d⟩∈r
shows Interval(r,b,c) ⊆ Interval(r,a,d)
proof
  fix x assume A3: x ∈ Interval(r, b, c)
  note A1
  moreover from A2 A3 have ⟨ a,b⟩ ∈ r ∧ ⟨ b,x⟩ ∈ r using Order˙ZF˙2˙L1A
    by simp
  ultimately have T1: ⟨ a,x⟩ ∈ r by (rule Fol1˙L3)
  note A1
  moreover from A2 A3 have ⟨ x,c⟩ ∈ r ∧ ⟨ c,d⟩ ∈ r using Order˙ZF˙2˙L1A
    by simp
  ultimately have ⟨ x,d⟩ ∈ r by (rule Fol1˙L3)
  with T1 show x ∈ Interval(r,a,d) using Order˙ZF˙2˙L1B
    by simp
qed
```

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

```
lemma Order˙ZF˙2˙L4:
  assumes A1: refl(X,r) and A2: antisym(r) and A3: a∈X
  shows Interval(r,a,a) = −a″
proof
  from A1 A3 have ⟨ a,a⟩ ∈ r using refl˙def by simp
  with A1 A3 show −a″ ⊆ Interval(r,a,a) using Order˙ZF˙2˙L2 by simp
  from A2 show Interval(r,a,a) ⊆ −a″ using Order˙ZF˙2˙L1A Fol1˙L4
    by fast
qed
```

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

```
lemma Order˙ZF˙2˙L5: assumes A1: trans(r) and A2: ⟨ a,b⟩ ∉ r
  shows Interval(r,a,b) = 0
proof -
  { assume Interval(r,a,b)≠0 then obtain x where x ∈ Interval(r,a,b)
    by auto
  with A1 A2 have False using Order˙ZF˙2˙L1A Fol1˙L3 by fast
  } thus thesis by auto
qed
```

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

```
lemma Order˙ZF˙2˙L6: assumes A1: r ⊆ X×X
  shows Interval(r,a,b) ⊆ X
  using assms Interval˙def by auto
```

5.3 Bounded sets

In this section we consider properties of bounded sets.

For reflexive relations singletons are bounded.

```
lemma Order˙ZF˙3˙L1: assumes refl(X,r) and a∈X
  shows IsBounded(–a″,r)
  using assms refl˙def IsBoundedAbove˙def IsBoundedBelow˙def
    IsBounded˙def by auto
```

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

```
lemma Order˙ZF˙3˙L1A: assumes r ⊆ X×X
  and IsBoundedAbove(A,r)
  shows A⊆X using assms IsBoundedAbove˙def by auto
```

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

```
lemma Order˙ZF˙3˙L1B: assumes r ⊆ X×X
  and IsBoundedBelow(A,r)
  shows A⊆X using assms IsBoundedBelow˙def by auto
```

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

```
lemma Order˙ZF˙3˙L2: assumes r –is total on″ X
  and x∈X y∈X
  shows
  ⟨x,GreaterOf(r,x,y)⟩ ∈ r
  ⟨y,GreaterOf(r,x,y)⟩ ∈ r
  ⟨SmallerOf(r,x,y),x⟩ ∈ r
  ⟨SmallerOf(r,x,y),y⟩ ∈ r
  using assms IsTotal˙def Order˙ZF˙1˙L1 GreaterOf˙def SmallerOf˙def
  by auto
```

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

```
lemma Order˙ZF˙3˙L2B:
  assumes A1: r –is total on″ X and A2: trans(r)
  and A3: u∈X w∈X
  and A4: ∀x∈A. ⟨ x,u⟩ ∈ r ∀x∈B. ⟨ x,w⟩ ∈ r
  shows ∀x∈A∪B. ⟨x,GreaterOf(r,u,w)⟩ ∈ r
proof
  let v = GreaterOf(r,u,w)
  from A1 A3 have T1: ⟨ u,v⟩ ∈ r and T2: ⟨ w,v⟩ ∈ r
    using Order˙ZF˙3˙L2 by auto
  fix x assume A5: x∈A∪B show ⟨x,v⟩ ∈ r
  proof -
    { assume x∈A
    with A4 T1 have ⟨ x,u⟩ ∈ r ∧ ⟨ u,v⟩ ∈ r by simp
    with A2 have ⟨x,v⟩ ∈ r by (rule Fol1˙L3) }
  moreover
  { assume x∉A
    with A5 A4 T2 have ⟨ x,w⟩ ∈ r ∧ ⟨ w,v⟩ ∈ r by simp
    with A2 have ⟨x,v⟩ ∈ r by (rule Fol1˙L3) }
```

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

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

```
lemma Order˙ZF˙3˙L3:
  assumes A1: r –is total on˝ X and A2: trans(r)
  and A3: IsBoundedAbove(A,r) IsBoundedAbove(B,r)
  and A4: r ⊆ X×X
  shows IsBoundedAbove(A∪B,r)
proof -
  { assume A=0 ∨ B=0
    with A3 have IsBoundedAbove(A∪B,r) by auto }
  moreover
  { assume ¬ (A = 0 ∨ B = 0)
    then have T1: A≠0 B≠0 by auto
    with A3 obtain u w where D1: ∀x∈A. ⟨ x,u⟩ ∈ r ∀x∈B. ⟨ x,w⟩ ∈ r
      using IsBoundedAbove˙def by auto
    let U = GreaterOf(r,u,w)
    from T1 A4 D1 have u∈X w∈X by auto
    with A1 A2 D1 have ∀x∈A∪B.⟨ x,U⟩ ∈ r
      using Order˙ZF˙3˙L2B by blast
    then have IsBoundedAbove(A∪B,r)
      using IsBoundedAbove˙def by auto }
  ultimately show thesis by auto
qed
```

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

```
lemma Order˙ZF˙3˙L4:
  assumes A1: r –is total on˝ X and A2: trans(r)
  and A3: IsBoundedAbove(A,r) and A4: a∈X and A5: r ⊆ X×X
  shows IsBoundedAbove(A∪–a˝,r)
proof -
  from A1 have refl(X,r)
    using total˙is˙refl by simp
  with assms show thesis using
    Order˙ZF˙3˙L1 IsBounded˙def Order˙ZF˙3˙L3 by simp
qed
```

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

```
lemma Order˙ZF˙3˙L5B:
  assumes A1: r –is total on˝ X and A2: trans(r)
  and A3: l∈X m∈X
  and A4: ∀x∈A. ⟨ l,x⟩ ∈ r ∀x∈B. ⟨ m,x⟩ ∈ r
  shows ∀x∈A∪B. ⟨SmallerOf(r,l,m),x⟩ ∈ r
```

```
proof
  let k = SmallerOf(r,l,m)
  from A1 A3 have T1: ⟨ k,l⟩ ∈ r and T2: ⟨ k,m⟩ ∈ r
    using Order˙ZF˙3˙L2 by auto
  fix x assume A5: x∈A∪B show ⟨k,x⟩ ∈ r
  proof -
    { assume x∈A
      with A4 T1 have ⟨ k,l⟩ ∈ r ∧ ⟨ l,x⟩ ∈ r by simp
      with A2 have ⟨k,x⟩ ∈ r by (rule Fol1˙L3) }
    moreover
    { assume x∉A
      with A5 A4 T2 have ⟨ k,m⟩ ∈ r ∧ ⟨ m,x⟩ ∈ r by simp
      with A2 have ⟨k,x⟩ ∈ r by (rule Fol1˙L3) }
    ultimately show thesis by auto
  qed
qed
```

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

```
lemma Order˙ZF˙3˙L6:
  assumes A1: r –is total on″ X and A2: trans(r)
  and A3: IsBoundedBelow(A,r) IsBoundedBelow(B,r)
  and A4: r ⊆ X×X
  shows IsBoundedBelow(A∪B,r)
proof -
  { assume A=0 ∨ B=0
    with A3 have thesis by auto }
  moreover
  { assume ¬ (A = 0 ∨ B = 0)
    then have T1: A≠0 B≠0 by auto
    with A3 obtain l m where D1: ∀x∈A. ⟨ l,x⟩ ∈ r ∀x∈B. ⟨ m,x⟩ ∈ r
      using IsBoundedBelow˙def by auto
    let L = SmallerOf(r,l,m)
    from T1 A4 D1 have T1: l∈X m∈X by auto
    with A1 A2 D1 have ∀x∈A∪B.⟨ L,x⟩ ∈ r
      using Order˙ZF˙3˙L5B by blast
    then have IsBoundedBelow(A∪B,r)
      using IsBoundedBelow˙def by auto }
  ultimately show thesis by auto
qed
```

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

```
lemma Order˙ZF˙3˙L7:
  assumes A1: r –is total on″ X and A2: trans(r)
  and A3: IsBoundedBelow(A,r) and A4: a∈X and A5: r ⊆ X×X
  shows IsBoundedBelow(A∪–a″,r)
proof -
  from A1 have refl(X,r)
```

```
    using total˙is˙refl by simp
  with assms show thesis using
    Order˙ZF˙3˙L1 IsBounded˙def Order˙ZF˙3˙L6 by simp
qed
```

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

```
theorem Order˙ZF˙3˙T1:
  assumes r –is total on″ X and trans(r)
  and IsBounded(A,r) IsBounded(B,r)
  and r ⊆ X×X
  shows IsBounded(A∪B,r)
  using assms Order˙ZF˙3˙L3 Order˙ZF˙3˙L6 Order˙ZF˙3˙L7 IsBounded˙def
  by simp
```

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

```
lemma Order˙ZF˙3˙L8:
  assumes r –is total on″ X and trans(r)
  and IsBounded(A,r) and a∈X and r ⊆ X×X
  shows IsBounded(A∪–a″,r)
  using assms total˙is˙refl Order˙ZF˙3˙L1 Order˙ZF˙3˙T1 by blast
```

A sufficient condition for a set to be bounded below.

```
lemma Order˙ZF˙3˙L9: assumes A1: ∀a∈A. ⟨l,a⟩ ∈ r
  shows IsBoundedBelow(A,r)
proof -
  from A1 have ∃l. ∀x∈A. ⟨l,x⟩ ∈ r
    by auto
  then show IsBoundedBelow(A,r)
    using IsBoundedBelow˙def by simp
qed
```

A sufficient condition for a set to be bounded above.

```
lemma Order˙ZF˙3˙L10: assumes A1: ∀a∈A. ⟨a,u⟩ ∈ r
  shows IsBoundedAbove(A,r)
proof -
  from A1 have ∃u. ∀x∈A. ⟨x,u⟩ ∈ r
    by auto
  then show IsBoundedAbove(A,r)
    using IsBoundedAbove˙def by simp
qed
```

Intervals are bounded.

```
lemma Order˙ZF˙3˙L11: shows
  IsBoundedAbove(Interval(r,a,b),r)
  IsBoundedBelow(Interval(r,a,b),r)
  IsBounded(Interval(r,a,b),r)
proof -
```

```
  { fix x assume x ∈ Interval(r,a,b)
    then have ⟨ x,b⟩ ∈ r  ⟨ a,x⟩ ∈ r
      using Order˙ZF˙2˙L1A by auto
  } then have
      ∃u. ∀x∈Interval(r,a,b). ⟨ x,u⟩ ∈ r
      ∃l. ∀x∈Interval(r,a,b). ⟨ l,x⟩ ∈ r
    by auto
  then show
    IsBoundedAbove(Interval(r,a,b),r)
    IsBoundedBelow(Interval(r,a,b),r)
    IsBounded(Interval(r,a,b),r)
    using IsBoundedAbove˙def IsBoundedBelow˙def IsBounded˙def
    by auto
qed
```

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

```
lemma Order˙ZF˙3˙L12: assumes A1: IsBoundedBelow(A,r) and A2: B⊆A
  shows IsBoundedBelow(B,r)
proof -
  { assume A = 0
    with assms have IsBoundedBelow(B,r)
      using IsBoundedBelow˙def by auto }
  moreover
  { assume A ≠ 0
    with A1 have ∃l. ∀x∈A. ⟨l,x⟩ ∈ r
      using IsBoundedBelow˙def by simp
    with A2 have ∃l.∀x∈B. ⟨l,x⟩ ∈ r by auto
    then have IsBoundedBelow(B,r) using IsBoundedBelow˙def
      by auto }
  ultimately show IsBoundedBelow(B,r) by auto
qed
```

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

```
lemma Order˙ZF˙3˙L13: assumes A1: IsBoundedAbove(A,r) and A2: B⊆A
  shows IsBoundedAbove(B,r)
proof -
  { assume A = 0
    with assms have IsBoundedAbove(B,r)
      using IsBoundedAbove˙def by auto }
  moreover
  { assume A ≠ 0
    with A1 have ∃u. ∀x∈A. ⟨x,u⟩ ∈ r
      using IsBoundedAbove˙def by simp
    with A2 have ∃u.∀x∈B. ⟨x,u⟩ ∈ r by auto
    then have IsBoundedAbove(B,r) using IsBoundedAbove˙def
      by auto }
  ultimately show IsBoundedAbove(B,r) by auto
qed
```

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

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

```
lemma Order˙ZF˙3˙L14:
  assumes A1: r –is total on″ X
  and A2: trans(r) and A3: antisym(r)
  and A4: r ⊆ X×X and A5: X≠0
  and A6: ∀x∈X. ∃a∈A. x≠a ∧ ⟨x,a⟩ ∈ r
  shows ¬IsBoundedAbove(A,r)
proof -
  { from A5 A6 have I: A≠0 by auto
    moreover assume IsBoundedAbove(A,r)
    ultimately obtain u where II: ∀x∈A. ⟨ x,u⟩ ∈ r
      using IsBounded˙def IsBoundedAbove˙def by auto
    with A4 I have u∈X by auto
    with A6 obtain b where b∈A and III: u≠b and ⟨u,b⟩ ∈ r
      by auto
    with II have ⟨b,u⟩ ∈ r  ⟨u,b⟩ ∈ r by auto
    with A3 have b=u by (rule Fol1˙L4)
    with III have False by simp
  } thus ¬IsBoundedAbove(A,r) by auto
qed
```

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

```
lemma Order˙ZF˙3˙L15: shows IsBoundedAbove(–x∈A. ⟨x,a⟩ ∈ r″,r)
  using IsBoundedAbove˙def by auto
```

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

```
lemma Order˙ZF˙3˙L16: assumes A1: IsBoundedBelow(A,r)
  shows IsBounded(–x∈A. ⟨x,a⟩ ∈ r″,r)
proof -
  { assume A=0
    then have IsBounded(–x∈A. ⟨x,a⟩ ∈ r″,r)
      using IsBoundedBelow˙def IsBoundedAbove˙def IsBounded˙def
      by auto }
  moreover
  { assume A≠0
    with A1 obtain l where I: ∀x∈A. ⟨l,x⟩ ∈ r
      using IsBoundedBelow˙def by auto
    then have ∀y∈–x∈A. ⟨x,a⟩ ∈ r″. ⟨l,y⟩ ∈ r by simp
    then have IsBoundedBelow(–x∈A. ⟨x,a⟩ ∈ r″,r)
      by (rule Order˙ZF˙3˙L9)
    then have IsBounded(–x∈A. ⟨x,a⟩ ∈ r″,r)
      using Order˙ZF˙3˙L15 IsBounded˙def by simp }
  ultimately show thesis by blast
qed

end
```

6 More on order relations

theory Order˙ZF˙1 imports ZF.Order ZF1

begin

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

6.1 Definitions and basic properties

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

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

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

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

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

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

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

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

```
lemma orders˙imp˙tot˙ord: assumes A1: R Orders A
  shows
  irrefl(A,R)
  trans[A](R)
  part˙ord(A,R)
  linear(A,R)
  tot˙ord(A,R)
proof -
```

```
  from A1 have I:
    ∀x y z. (x∈A ∧ y∈A ∧ z∈A) ⟶
    (⟨x,y⟩ ∈ R ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ R)) ∧
    (⟨x,y⟩ ∈ R ∧ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R)
    unfolding StrictOrder˙def by simp
  then have ∀x∈A. ⟨x,x⟩ ∉ R by blast
  then show irrefl(A,R) using irrefl˙def by simp
  moreover
  from I have
    ∀x∈A. ∀y∈A. ∀z∈A. ⟨x,y⟩ ∈ R ⟶ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R
    by blast
  then show trans[A](R) unfolding trans˙on˙def by blast
  ultimately show part˙ord(A,R) using part˙ord˙def
    by simp
  moreover
  from I have
    ∀x∈A. ∀y∈A. ⟨x,y⟩ ∈ R ∨ x=y ∨ ⟨y,x⟩ ∈ R
    by blast
  then show linear(A,R) unfolding linear˙def by blast
  ultimately show tot˙ord(A,R) using tot˙ord˙def
    by simp
qed
```

A converse of orders˙imp˙tot˙ord. Together with that theorem this shows that Metamath's notion of an order relation is equivalent to Isabelles tot˙ord predicate.

```
lemma tot˙ord˙imp˙orders: assumes A1: tot˙ord(A,R)
  shows R Orders A
proof -
  from A1 have
    I: linear(A,R) and
    II: irrefl(A,R) and
    III: trans[A](R) and
    IV: part˙ord(A,R)
    using tot˙ord˙def part˙ord˙def by auto
  from IV have asym(R ∩ A×A)
    using part˙ord˙Imp˙asym by simp
  then have V: ∀x y. ⟨x,y⟩ ∈ (R ∩ A×A) ⟶ ¬(⟨y,x⟩ ∈ (R ∩ A×A))
    unfolding asym˙def by blast
  from I have VI: ∀x∈A. ∀y∈A. ⟨x,y⟩ ∈ R ∨ x=y ∨ ⟨y,x⟩ ∈ R
    unfolding linear˙def by blast
  from III have VII:
    ∀x∈A. ∀y∈A. ∀z∈A. ⟨x,y⟩ ∈ R ⟶ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R
    unfolding trans˙on˙def by blast
  { fix x y z
    assume T: x∈A y∈A z∈A
    have ⟨x,y⟩ ∈ R ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ R)
    proof
      assume A2: ⟨x,y⟩ ∈ R
```

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

6.2 Properties of (strict) total orders

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

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

```
lemma cnvso: shows R Orders A ⟷ converse(R) Orders A
proof
  let r = converse(R)
  assume R Orders A
  then have tot˙ord(A,r) using orders˙imp˙tot˙ord tot˙ord˙converse
    by simp
  then show r Orders A using tot˙ord˙imp˙orders
    by simp
next
  let r = converse(R)
  assume r Orders A
  then have A2: ∀x y z. (x∈A ∧ y∈A ∧ z∈A) ⟶
    (⟨x,y⟩ ∈ r ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ r)) ∧
    (⟨x,y⟩ ∈ r ∧ ⟨y,z⟩ ∈ r ⟶ ⟨x,z⟩ ∈ r)
    using StrictOrder˙def by simp
  { fix x y z
    assume x∈A ∧ y∈A ∧ z∈A
```

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

Supremum is unique, if it exists.

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

Supremum has expected properties if it exists.

```
lemma sup˙props: assumes A1: R Orders A and
  A2: ∃x∈A. (∀y∈B. ⟨x,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,x⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R))
```

```
  shows
  Sup(B,A,R) ∈ A
  ∀y∈B. ⟨Sup(B,A,R),y⟩ ∉ R
  ∀y∈A. ⟨y,Sup(B,A,R)⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R )
proof -
  let S = –x∈A. (∀y∈B. ⟨x,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,x⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R ) )
"
  from A2 obtain x where
    x∈A and (∀y∈B. ⟨x,y⟩ ∉ R) and ∀y∈A. ⟨y,x⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R)
    by auto
  with A1 have I:
    ∃!x. x∈A∧(∀y∈B. ⟨x,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,x⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R))
    using supeu by simp
  then have ( ⋃S ) ∈ A by (rule ZF1˙1˙L9)
  then show Sup(B,A,R) ∈ A using Sup˙def by simp
  from I have II:
    (∀y∈B. ⟨⋃S ,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,⋃S⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R))
    by (rule ZF1˙1˙L9)
  hence ∀y∈B. ⟨⋃S,y⟩ ∉ R by blast
  moreover have III: (⋃S) = Sup(B,A,R) using Sup˙def by simp
  ultimately show ∀y∈B. ⟨Sup(B,A,R),y⟩ ∉ R by simp
  from II have IV: ∀y∈A. ⟨y,⋃S⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R)
    by blast
  { fix y assume A3: y∈A and ⟨y,Sup(B,A,R)⟩ ∈ R
    with III have ⟨y,⋃S⟩ ∈ R by simp
    with IV A3 have ∃z∈B. ⟨y,z⟩ ∈ R by blast
  } thus  ∀y∈A. ⟨y,Sup(B,A,R)⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R )
    by simp
qed
```

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

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

end
```

7 Even more on order relations

```
theory Order˙ZF˙1a imports Order˙ZF
```

begin

This theory is a continuation of Order˙ZF and talks about maximuma and minimum of a set, supremum and infimum and strict (not reflexive) versions of order relations.

7.1 Maximum and minimum of a set

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

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

```
lemma set˙max˙not˙empty: assumes HasAmaximum(r,A) shows A≠0
  using assms unfolding HasAmaximum˙def by auto
```

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

```
lemma set˙min˙not˙empty: assumes HasAminimum(r,A) shows A≠0
  using assms unfolding HasAminimum˙def by auto
```

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

```
lemma set˙sup˙not˙empty: assumes HasAsupremum(r,A) shows A≠0
proof -
  from assms have HasAminimum(r,⋂a∈A. r–a″) unfolding HasAsupremum˙def
    by simp
  then have (⋂a∈A. r–a″) ≠ 0 using set˙min˙not˙empty by simp
  then obtain x where x ∈ (⋂y∈A. r–y″) by blast
  thus thesis by auto
qed
```

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

```
lemma set˙inf˙not˙empty: assumes HasAnInfimum(r,A) shows A≠0
proof -
  from assms have HasAmaximum(r,⋂a∈A. r-–a″) unfolding HasAnInfimum˙def
    by simp
  then have (⋂a∈A. r-–a″) ≠ 0 using set˙max˙not˙empty by simp
  then obtain x where x ∈ (⋂y∈A. r-–y″) by blast
  thus thesis by auto
qed
```

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

```
lemma Order˙ZF˙4˙L1: assumes A1: antisym(r) and A2: HasAmaximum(r,A)
  shows ∃!M. M∈A ∧ (∀x∈A. ⟨ x,M⟩ ∈ r)
proof
  from A2 show ∃M. M ∈ A ∧ (∀x∈A. ⟨x, M⟩ ∈ r)
    using HasAmaximum˙def by auto
  fix M1 M2 assume
    A2: M1 ∈ A ∧ (∀x∈A. ⟨x, M1⟩ ∈ r) M2 ∈ A ∧ (∀x∈A. ⟨x, M2⟩ ∈ r)
    then have ⟨M1,M2⟩ ∈ r ⟨M2,M1⟩ ∈ r by auto
    with A1 show M1=M2 by (rule Fol1˙L4)
qed
```

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

```
lemma Order˙ZF˙4˙L2: assumes A1: antisym(r) and A2: HasAminimum(r,A)
  shows ∃!m. m∈A ∧ (∀x∈A. ⟨ m,x⟩ ∈ r)
proof
  from A2 show ∃m. m ∈ A ∧ (∀x∈A. ⟨m, x⟩ ∈ r)
    using HasAminimum˙def by auto
  fix m1 m2 assume
    A2: m1 ∈ A ∧ (∀x∈A. ⟨m1, x⟩ ∈ r) m2 ∈ A ∧ (∀x∈A. ⟨m2, x⟩ ∈ r)
    then have ⟨m1,m2⟩ ∈ r ⟨m2,m1⟩ ∈ r by auto
    with A1 show m1=m2 by (rule Fol1˙L4)
qed
```

Maximum of a set has desired properties.

```
lemma Order˙ZF˙4˙L3: assumes A1: antisym(r) and A2: HasAmaximum(r,A)
  shows Maximum(r,A) ∈ A ∀x∈A. ⟨x,Maximum(r,A)⟩ ∈ r
proof -
  let Max = THE M. M∈A ∧ (∀x∈A. ⟨ x,M⟩ ∈ r)
  from A1 A2 have ∃!M. M∈A ∧ (∀x∈A. ⟨ x,M⟩ ∈ r)
    by (rule Order˙ZF˙4˙L1)
  then have Max ∈ A ∧ (∀x∈A. ⟨ x,Max⟩ ∈ r)
    by (rule theI)
  then show Maximum(r,A) ∈ A ∀x∈A. ⟨x,Maximum(r,A)⟩ ∈ r
    using Maximum˙def by auto
qed
```

Minimum of a set has desired properties.

```
lemma Order˙ZF˙4˙L4: assumes A1: antisym(r) and A2: HasAminimum(r,A)
  shows Minimum(r,A) ∈ A ∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r
proof -
  let Min = THE m. m∈A ∧ (∀x∈A. ⟨ m,x⟩ ∈ r)
  from A1 A2 have ∃!m. m∈A ∧ (∀x∈A. ⟨ m,x⟩ ∈ r)
    by (rule Order˙ZF˙4˙L2)
  then have Min ∈ A ∧ (∀x∈A. ⟨ Min,x⟩ ∈ r)
    by (rule theI)
  then show Minimum(r,A) ∈ A ∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r
    using Minimum˙def by auto
qed
```

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

```
lemma Order˙ZF˙4˙L5:
  assumes A1: r –is total on″ (A∪B) and A2: trans(r)
  and A3: HasAmaximum(r,A) HasAmaximum(r,B)
  shows HasAmaximum(r,A∪B)
proof -
  from A3 obtain M K where
    D1: M∈A ∧ (∀x∈A. ⟨ x,M⟩ ∈ r) K∈B ∧ (∀x∈B. ⟨ x,K⟩ ∈ r)
    using HasAmaximum˙def by auto
  let L = GreaterOf(r,M,K)
  from D1 have T1: M ∈ A∪B K ∈ A∪B
    ∀x∈A. ⟨ x,M⟩ ∈ r ∀x∈B. ⟨ x,K⟩ ∈ r
    by auto
  with A1 A2 have ∀x∈A∪B.⟨ x,L⟩ ∈ r by (rule Order˙ZF˙3˙L2B)
  moreover from T1 have L ∈ A∪B using GreaterOf˙def IsTotal˙def
    by simp
  ultimately show HasAmaximum(r,A∪B) using HasAmaximum˙def by auto
qed
```

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

```
lemma Order˙ZF˙4˙L6:
  assumes A1: r –is total on″ (A∪B) and A2: trans(r)
  and A3: HasAminimum(r,A) HasAminimum(r,B)
  shows HasAminimum(r,A∪B)
proof -
  from A3 obtain m k where
    D1: m∈A ∧ (∀x∈A. ⟨ m,x⟩ ∈ r) k∈B ∧ (∀x∈B. ⟨ k,x⟩ ∈ r)
    using HasAminimum˙def by auto
  let l = SmallerOf(r,m,k)
  from D1 have T1: m ∈ A∪B k ∈ A∪B
    ∀x∈A. ⟨ m,x⟩ ∈ r ∀x∈B. ⟨ k,x⟩ ∈ r
    by auto
  with A1 A2 have ∀x∈A∪B.⟨ l,x⟩ ∈ r by (rule Order˙ZF˙3˙L5B)
  moreover from T1 have l ∈ A∪B using SmallerOf˙def IsTotal˙def
    by simp
  ultimately show HasAminimum(r,A∪B) using HasAminimum˙def by auto
qed
```

Set that has a maximum is bounded above.

```
lemma Order˙ZF˙4˙L7:
  assumes HasAmaximum(r,A)
  shows IsBoundedAbove(A,r)
  using assms HasAmaximum˙def IsBoundedAbove˙def by auto
```

Set that has a minimum is bounded below.

```
lemma Order˙ZF˙4˙L8A:
```

```
  assumes HasAminimum(r,A)
  shows IsBoundedBelow(A,r)
  using assms HasAminimum˙def IsBoundedBelow˙def by auto
```

For reflexive relations singletons have a minimum and maximum.

```
lemma Order˙ZF˙4˙L8: assumes refl(X,r) and a∈X
  shows HasAmaximum(r,–a″) HasAminimum(r,–a″)
  using assms refl˙def HasAmaximum˙def HasAminimum˙def by auto
```

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

```
lemma Order˙ZF˙4˙L9:
  assumes A1: r –is total on″ X and A2: trans(r)
  and A3: A⊆X and A4: a∈X and A5: HasAmaximum(r,A)
  shows HasAmaximum(r,A∪–a″)
proof -
  from A3 A4 have A∪–a″ ⊆ X by auto
  with A1 have r –is total on″ (A∪–a″)
    using Order˙ZF˙1˙L4 by blast
  moreover from A1 A2 A4 A5 have
    trans(r) HasAmaximum(r,A) by auto
  moreover from A1 A4 have HasAmaximum(r,–a″)
    using total˙is˙refl Order˙ZF˙4˙L8 by blast
  ultimately show HasAmaximum(r,A∪–a″) by (rule Order˙ZF˙4˙L5)
qed
```

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

```
lemma Order˙ZF˙4˙L10:
  assumes A1: r –is total on″ X and A2: trans(r)
  and A3: A⊆X and A4: a∈X and A5: HasAminimum(r,A)
  shows HasAminimum(r,A∪–a″)
proof -
  from A3 A4 have A∪–a″ ⊆ X by auto
  with A1 have r –is total on″ (A∪–a″)
    using Order˙ZF˙1˙L4 by blast
  moreover from A1 A2 A4 A5 have
    trans(r) HasAminimum(r,A) by auto
  moreover from A1 A4 have HasAminimum(r,–a″)
    using total˙is˙refl Order˙ZF˙4˙L8 by blast
  ultimately show HasAminimum(r,A∪–a″) by (rule Order˙ZF˙4˙L6)
qed
```

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

```
lemma Order˙ZF˙4˙L11:
  assumes A1: r –is total on″ X and
```

```
  A2: trans(r) and
  A3: r ⊆ X×X and
  A4: ∀A. IsBounded(A,r) ∧ A≠0 ⟶ HasAminimum(r,A) and
  A5: B≠0 and A6: IsBoundedBelow(B,r)
  shows HasAminimum(r,B)
proof -
  from A5 obtain b where T: b∈B by auto
  let L = −x∈B. ⟨x,b⟩ ∈ r″
  from A3 A6 T have T1: b∈X using Order˙ZF˙3˙L1B by blast
  with A1 T have T2: b ∈ L
    using total˙is˙refl refl˙def by simp
  then have L ≠ 0 by auto
  moreover have IsBounded(L,r)
  proof -
    have L ⊆ B by auto
    with A6 have IsBoundedBelow(L,r)
      using Order˙ZF˙3˙L12 by simp
    moreover have IsBoundedAbove(L,r)
      by (rule Order˙ZF˙3˙L15)
    ultimately have IsBoundedAbove(L,r) ∧ IsBoundedBelow(L,r)
      by blast
    then show IsBounded(L,r) using IsBounded˙def
      by simp
  qed
  ultimately have IsBounded(L,r) ∧ L ≠ 0 by blast
  with A4 have HasAminimum(r,L) by simp
  then obtain m where I: m∈L and II: ∀x∈L. ⟨ m,x⟩ ∈ r
    using HasAminimum˙def by auto
  then have III: ⟨m,b⟩ ∈ r by simp
  from I have m∈B by simp
  moreover have ∀x∈B. ⟨m,x⟩ ∈ r
  proof
    fix x assume A7: x∈B
    from A3 A6 have B⊆X using Order˙ZF˙3˙L1B by blast
    with A1 A7 T1 have x ∈  L ∪ −x∈B. ⟨b,x⟩ ∈ r″
      using Order˙ZF˙1˙L5 by simp
    then have x∈L ∨ ⟨b,x⟩ ∈ r by auto
    moreover
    { assume x∈L
      with II have ⟨m,x⟩ ∈ r by simp }
    moreover
    { assume ⟨b,x⟩ ∈ r
      with A2 III have trans(r) and ⟨m,b⟩ ∈ r ∧ ⟨b,x⟩ ∈ r
 by auto
      then have  ⟨m,x⟩ ∈ r by (rule Fol1˙L3) }
    ultimately show ⟨m,x⟩ ∈ r by auto
  qed
  ultimately show HasAminimum(r,B) using HasAminimum˙def
    by auto
```

qed

A dual to Order˙ZF˙4˙L11: If the order relation has a property that every nonempty bounded set attains a maximum (for example integers are like that), then every nonempty set bounded above attains a maximum.

```
lemma Order˙ZF˙4˙L11A:
  assumes A1: r –is total on″ X and
  A2: trans(r) and
  A3: r ⊆ X×X and
  A4: ∀A. IsBounded(A,r) ∧ A≠0 ⟶ HasAmaximum(r,A) and
  A5: B≠0 and A6: IsBoundedAbove(B,r)
  shows HasAmaximum(r,B)
proof -
  from A5 obtain b where T: b∈B by auto
  let U = –x∈B. ⟨b,x⟩ ∈ r″
  from A3 A6 T have T1: b∈X using Order˙ZF˙3˙L1A by blast
  with A1 T have T2: b ∈ U
    using total˙is˙refl refl˙def by simp
  then have U ≠ 0 by auto
  moreover have IsBounded(U,r)
  proof -
    have U ⊆ B by auto
    with A6 have IsBoundedAbove(U,r)
      using Order˙ZF˙3˙L13 by blast
    moreover have IsBoundedBelow(U,r)
      using IsBoundedBelow˙def by auto
    ultimately have IsBoundedAbove(U,r) ∧ IsBoundedBelow(U,r)
      by blast
    then show IsBounded(U,r) using IsBounded˙def
      by simp
  qed
  ultimately have IsBounded(U,r) ∧ U ≠ 0 by blast
  with A4 have HasAmaximum(r,U) by simp
  then obtain m where I: m∈U and II: ∀x∈U. ⟨x,m⟩ ∈ r
    using HasAmaximum˙def by auto
  then have III: ⟨b,m⟩ ∈ r by simp
  from I have m∈B by simp
  moreover have ∀x∈B. ⟨x,m⟩ ∈ r
  proof
    fix x assume A7: x∈B
    from A3 A6 have B⊆X using Order˙ZF˙3˙L1A by blast
    with A1 A7 T1 have x ∈ –x∈B. ⟨x,b⟩ ∈ r″ ∪ U
      using Order˙ZF˙1˙L5 by simp
    then have x∈U ∨ ⟨x,b⟩ ∈ r by auto
    moreover
    { assume x∈U
      with II have ⟨x,m⟩ ∈ r by simp }
    moreover
    { assume ⟨x,b⟩ ∈ r
```

```
    with A2 III have trans(r) and ⟨x,b⟩ ∈ r ∧ ⟨b,m⟩ ∈ r
 by auto
    then have  ⟨x,m⟩ ∈ r by (rule Fol1˙L3) }
   ultimately show ⟨x,m⟩ ∈ r by auto
  qed
  ultimately show HasAmaximum(r,B) using HasAmaximum˙def
   by auto
qed
```

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

```
lemma Order˙ZF˙4˙L12:
  assumes antisym(r) and HasAminimum(r,A) and ∀a∈A. ⟨L,a⟩ ∈ r
  shows ⟨L,Minimum(r,A)⟩ ∈ r
  using assms Order˙ZF˙4˙L4 by simp
```

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

```
lemma Order˙ZF˙4˙L13:
  assumes antisym(r) and HasAmaximum(r,A) and ∀a∈A. ⟨a,M⟩ ∈ r
  shows ⟨Maximum(r,A),M⟩ ∈ r
  using assms Order˙ZF˙4˙L3 by simp
```

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

```
lemma Order˙ZF˙4˙L14:
  assumes A1: antisym(r) and A2: M ∈ A and
  A3: ∀a∈A. ⟨a,M⟩ ∈ r
  shows Maximum(r,A) = M
proof -
  from A2 A3 have I: HasAmaximum(r,A) using HasAmaximum˙def
    by auto
  with A1 have ∃!M. M∈A ∧ (∀x∈A. ⟨x,M⟩ ∈ r)
    using Order˙ZF˙4˙L1 by simp
  moreover from A2 A3 have M∈A ∧ (∀x∈A. ⟨x,M⟩ ∈ r) by simp
  moreover from A1 I have
    Maximum(r,A) ∈ A ∧ (∀x∈A. ⟨x,Maximum(r,A)⟩ ∈ r)
    using Order˙ZF˙4˙L3 by simp
  ultimately show Maximum(r,A) = M by auto
qed
```

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

```
lemma Order˙ZF˙4˙L15:
  assumes A1: antisym(r) and A2: m ∈ A and
  A3: ∀a∈A. ⟨m,a⟩ ∈ r
  shows Minimum(r,A) = m
proof -
```

```
  from A2 A3 have I: HasAminimum(r,A) using HasAminimum˙def
    by auto
  with A1 have ∃!m. m∈A ∧ (∀x∈A. ⟨m,x⟩ ∈ r)
    using Order˙ZF˙4˙L2 by simp
  moreover from A2 A3 have m∈A ∧ (∀x∈A. ⟨m,x⟩ ∈ r) by simp
  moreover from A1 I have
    Minimum(r,A) ∈ A ∧ (∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r)
    using Order˙ZF˙4˙L4 by simp
  ultimately show Minimum(r,A) = m by auto
qed
```

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

```
lemma Order˙ZF˙4˙L16:
  assumes A1: antisym(r) and A2: r –is total on″ X and
  A3: A⊆X and
  A4: ¬HasAmaximum(r,A) and
  A5: x∈A
  shows ∃y∈A. ⟨x,y⟩ ∈ r ∧ y≠x
proof -
  { assume A6: ∀y∈A. ⟨x,y⟩ ∉ r ∨ y=x
    have ∀y∈A. ⟨y,x⟩ ∈ r
    proof
      fix y assume A7: y∈A
      with A6 have ⟨x,y⟩ ∉ r ∨ y=x by simp
      with A2 A3 A5 A7 show ⟨y,x⟩ ∈ r
 using IsTotal˙def Order˙ZF˙1˙L1 by auto
    qed
    with A5 have ∃x∈A.∀y∈A. ⟨y,x⟩ ∈ r
      by auto
    with A4 have False using HasAmaximum˙def by simp
  } then show ∃y∈A. ⟨x,y⟩ ∈ r ∧ y≠x by auto
qed
```

7.2 Supremum and Infimum

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

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

```
lemma Order˙ZF˙5˙L1: assumes u ∈ (⋂a∈A. r–a″) and a∈A
  shows ⟨a,u⟩ ∈ r
  using assms by auto
```

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

```
lemma Order˙ZF˙5˙L2: assumes l ∈ (⋂a∈A. r-–a″) and a∈A
  shows ⟨l,a⟩ ∈ r
```

```
  using assms by auto
```

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

```
lemma Order˙ZF˙5˙L3: assumes A1: antisym(r) and A2: A≠0 and
  A3: HasAminimum(r,⋂a∈A. r–a″) and
  A4: ∀a∈A. ⟨a,u⟩ ∈ r
  shows ⟨Supremum(r,A),u⟩ ∈ r
proof -
  let U = ⋂a∈A. r–a″
  from A4 have ∀a∈A. u ∈ r–a″ using image˙singleton˙iff
    by simp
  with A2 have u∈U by auto
  with A1 A3 show ⟨Supremum(r,A),u⟩ ∈ r
    using Order˙ZF˙4˙L4 Supremum˙def by simp
qed
```

Supremum is less or equal than any upper bound.

```
lemma sup˙leq˙up˙bnd: assumes antisym(r) HasAsupremum(r,A) ∀a∈A. ⟨a,u⟩ ∈ r
  shows ⟨Supremum(r,A),u⟩ ∈ r
proof -
  let U = ⋂a∈A. r–a″
  from assms(3) have  ∀a∈A. u ∈ r–a″ using image˙singleton˙iff by simp
  with assms(2) have u∈U using set˙sup˙not˙empty by auto
  with assms(1,2) show ⟨Supremum(r,A),u⟩ ∈ r
    unfolding HasAsupremum˙def Supremum˙def using Order˙ZF˙4˙L4 by simp
qed
```

Infimum is greater or equal than any lower bound. This lemma is obsolete and will be removed. Use inf˙geq˙lo˙bnd instead.

```
lemma Order˙ZF˙5˙L4: assumes A1: antisym(r) and A2: A≠0 and
  A3: HasAmaximum(r,⋂a∈A. r-–a″) and
  A4: ∀a∈A. ⟨l,a⟩ ∈ r
  shows ⟨l,Infimum(r,A)⟩ ∈ r
proof -
  let L = ⋂a∈A. r-–a″
  from A4 have ∀a∈A. l ∈ r-–a″ using vimage˙singleton˙iff
    by simp
  with A2 have l∈L by auto
  with A1 A3 show ⟨l,Infimum(r,A)⟩ ∈ r
    using Order˙ZF˙4˙L3 Infimum˙def by simp
qed
```

Infimum is greater or equal than any upper bound.

```
lemma inf˙geq˙lo˙bnd: assumes antisym(r) HasAnInfimum(r,A) ∀a∈A. ⟨u,a⟩ ∈ r
```

```
  shows ⟨u,Infimum(r,A)⟩ ∈ r
proof -
  let U = ⋂a∈A. r-`-a``
  from assms(3) have ∀a∈A. u ∈ r-`-a`` using vimage`singleton`iff by simp
  with assms(2) have u∈U using set`inf`not`empty by auto
  with assms(1,2) show ⟨u,Infimum(r,A)⟩ ∈ r
    unfolding HasAnInfimum`def Infimum`def using Order`ZF`4`L3 by simp
qed
```

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

```
lemma Order`ZF`5`L5: assumes A1: antisym(r) and A2: A≠0 and
  A3: ∀x∈A. ⟨x,z⟩ ∈ r and
  A4: ∀y. (∀x∈A. ⟨x,y⟩ ∈ r) ⟶ ⟨z,y⟩ ∈ r
  shows
  HasAminimum(r,⋂a∈A. r–a``)
  z = Supremum(r,A)
proof -
  let B = ⋂a∈A. r–a``
  from A2 A3 A4 have I: z ∈ B   ∀y∈B. ⟨z,y⟩ ∈ r
    by auto
  then show HasAminimum(r,⋂a∈A. r–a``)
    using HasAminimum`def by auto
  from A1 I show z = Supremum(r,A)
    using Order`ZF`4`L15 Supremum`def by simp
qed
```

The dual theorem to Order`ZF`5`L5: if z is an lower bound for A and is greater or equal than any other lower bound, then z is the infimum of A.

```
lemma inf`glb:
  assumes antisym(r) A≠0 ∀x∈A. ⟨z,x⟩ ∈ r ∀y. (∀x∈A. ⟨y,x⟩ ∈ r) ⟶ ⟨y,z⟩ ∈ r
  shows
  HasAmaximum(r,⋂a∈A. r-`-a``)
  z = Infimum(r,A)
proof -
  let B = ⋂a∈A. r-`-a``
  from assms(2,3,4) have I: z ∈ B   ∀y∈B. ⟨y,z⟩ ∈ r
    by auto
  then show HasAmaximum(r,⋂a∈A. r-`-a``)
    unfolding HasAmaximum`def by auto
  from assms(1) I show z = Infimum(r,A)
    using Order`ZF`4`L14 Infimum`def by simp
qed
```

Supremum and infimum of a singleton is the element.

```
lemma sup`inf`singl: assumes antisym(r) refl(X,r) z∈X
  shows
    HasAsupremum(r,–z``) Supremum(r,–z``) = z and
```

```
  HasAnInfimum(r,–z″) Infimum(r,–z″) = z
proof -
 from assms show Supremum(r,–z″) = z and Infimum(r,–z″) = z
  using inf˙glb Order˙ZF˙5˙L5 unfolding refl˙def by auto
 from assms show  HasAsupremum(r,–z″)
  using Order˙ZF˙5˙L5 unfolding HasAsupremum˙def refl˙def by blast
 from assms show HasAnInfimum(r,–z″)
  using inf˙glb unfolding HasAnInfimum˙def refl˙def by blast
qed
```

If a set has a maximum, then the maximum is the supremum. This lemma is obsolete, use max˙is˙sup instead.

```
lemma Order˙ZF˙5˙L6:
 assumes A1:  antisym(r) and A2: A≠0 and
 A3: HasAmaximum(r,A)
 shows
 HasAminimum(r,⋂a∈A. r–a″)
 Maximum(r,A) = Supremum(r,A)
proof -
 let M = Maximum(r,A)
 from A1 A3 have I: M ∈ A and II: ∀x∈A. ⟨x,M⟩ ∈ r
  using Order˙ZF˙4˙L3 by auto
 from I have III: ∀y. (∀x∈A. ⟨x,y⟩ ∈ r) ⟶ ⟨M,y⟩ ∈ r
  by simp
 with A1 A2 II show HasAminimum(r,⋂a∈A. r–a″)
  by (rule Order˙ZF˙5˙L5)
 from A1 A2 II III show M = Supremum(r,A)
  by (rule Order˙ZF˙5˙L5)
qed
```

Another version of Order˙ZF˙5˙L6 that: if a sat has a maximum then it has a supremum and the maximum is the supremum.

```
lemma max˙is˙sup: assumes antisym(r) A≠0 HasAmaximum(r,A)
 shows HasAsupremum(r,A) and Maximum(r,A) = Supremum(r,A)
proof -
 let M = Maximum(r,A)
 from assms(1,3) have M ∈ A and I: ∀x∈A. ⟨x,M⟩ ∈ r using Order˙ZF˙4˙L3
  by auto
 with assms(1,2) have HasAminimum(r,⋂a∈A. r–a″) using Order˙ZF˙5˙L5(1)
  by blast
 then show HasAsupremum(r,A) unfolding HasAsupremum˙def by simp
 from assms(1,2) ‹M ∈ A› I show M = Supremum(r,A) using Order˙ZF˙5˙L5(2)
  by blast
qed
```

Minimum is the infimum if it exists.

```
lemma min˙is˙inf: assumes antisym(r) A≠0 HasAminimum(r,A)
 shows HasAnInfimum(r,A) and Minimum(r,A) = Infimum(r,A)
```

```
proof -
  let M = Minimum(r,A)
  from assms(1,3) have M∈A and I: ∀x∈A. ⟨M,x⟩ ∈ r using  Order˙ZF˙4˙L4
    by auto
  with assms(1,2) have HasAmaximum(r,⋂a∈A. r-–a″) using inf˙glb(1) by blast
  then show HasAnInfimum(r,A) unfolding HasAnInfimum˙def by simp
  from assms(1,2) ‹M ∈ A› I show M = Infimum(r,A) using inf˙glb(2) by blast
qed
```

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

```
lemma min˙max˙two˙el: assumes refl(X,r) r –is total on″ X x∈X y∈X
  shows HasAminimum(r,–x,y″) and HasAmaximum(r,–x,y″)
   using assms unfolding refl˙def IsTotal˙def HasAminimum˙def HasAmaximum˙def
by auto
```

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

```
lemma inf˙sup˙two˙el:assumes antisym(r) refl(X,r) r –is total on″ X x∈X y∈X
  shows
    HasAnInfimum(r,–x,y″)
    Minimum(r,–x,y″) = Infimum(r,–x,y″)
    HasAsupremum(r,–x,y″)
    Maximum(r,–x,y″) = Supremum(r,–x,y″)
  using assms min˙max˙two˙el max˙is˙sup min˙is˙inf by auto
```

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

```
lemma sup˙in˙space:
  assumes r ⊆ X×X antisym(r) HasAminimum(r,⋂a∈A. r–a″)
  shows Supremum(r,A) ∈ X and ∀x∈A. ⟨x,Supremum(r,A)⟩ ∈ r
proof -
   from assms(3) have A≠0 using set˙sup˙not˙empty unfolding HasAsupremum˙def
by simp
  then obtain a where a∈A by auto
  with assms(1,2,3) show Supremum(r,A) ∈ X unfolding Supremum˙def
    using Order˙ZF˙4˙L4 Order˙ZF˙5˙L1 by blast
  from assms(2,3) show ∀x∈A. ⟨x,Supremum(r,A)⟩ ∈ r unfolding Supremum˙def
    using Order˙ZF˙4˙L4 by blast
qed
```

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

```
lemma inf˙in˙space:
  assumes r ⊆ X×X antisym(r) HasAmaximum(r,⋂a∈A. r-–a″)
  shows Infimum(r,A) ∈ X and ∀x∈A. ⟨Infimum(r,A),x⟩ ∈ r
proof -
  from assms(3) have A≠0 using set˙inf˙not˙empty unfolding HasAnInfimum˙def by
simp
  then obtain a where a∈A by auto
```

```
  with assms(1,2,3) show Infimum(r,A) ∈ X unfolding Infimum˙def
    using Order˙ZF˙4˙L3 Order˙ZF˙5˙L1 by blast
  from assms(2,3) show ∀x∈A. ⟨Infimum(r,A),x⟩ ∈ r unfolding Infimum˙def
    using Order˙ZF˙4˙L3 by blast
qed
```

Properties of supremum of a set for complete relations.

```
lemma Order˙ZF˙5˙L7:
  assumes A1: r ⊆ X×X and A2: antisym(r) and
  A3: r –is complete″ and
  A4: A≠0 and A5: ∃x∈X. ∀y∈A. ⟨y,x⟩ ∈ r
  shows Supremum(r,A) ∈ X and ∀x∈A. ⟨x,Supremum(r,A)⟩ ∈ r
proof -
  from A3 A4 A5 have HasAminimum(r,⋂a∈A. r–a″)
    unfolding IsBoundedAbove˙def IsComplete˙def by blast
  with A1 A2 show Supremum(r,A) ∈ X and ∀x∈A. ⟨x,Supremum(r,A)⟩ ∈ r
    using sup˙in˙space by auto
qed
```

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

```
lemma inf˙inf:
  assumes
    r ⊆ X×X antisym(r) trans(r)
    ∀T∈𝒯. HasAnInfimum(r,T)
    HasAnInfimum(r,–Infimum(r,T).T∈𝒯″)
  shows
    HasAnInfimum(r,⋃𝒯) and Infimum(r,–Infimum(r,T).T∈𝒯″) = Infimum(r,⋃𝒯)
proof -
  let i = Infimum(r,–Infimum(r,T).T∈𝒯″)
  note assms(2)
  moreover from assms(4,5) have ⋃𝒯 ≠ 0 using set˙inf˙not˙empty by blast
  moreover
  have ∀T∈𝒯.∀t∈T. ⟨i,t⟩ ∈ r
  proof -
    { fix T t assume T∈𝒯 t∈T
      with assms(1,2,4) have ⟨Infimum(r,T),t⟩ ∈ r
        unfolding HasAnInfimum˙def using inf˙in˙space(2) by blast
      moreover from assms(1,2,5) ‹T∈𝒯› have ⟨i,Infimum(r,T)⟩ ∈ r
        unfolding HasAnInfimum˙def using inf˙in˙space(2) by blast
      moreover note assms(3)
      ultimately have ⟨i,t⟩ ∈ r unfolding trans˙def by blast
    } thus thesis by simp
  qed
  hence I: ∀t∈⋃𝒯. ⟨i,t⟩ ∈ r by auto
  moreover have J: ∀y. (∀x∈⋃𝒯. ⟨y,x⟩ ∈ r) ⟶ ⟨y,i⟩ ∈ r
  proof -
    { fix y x assume A: ∀x∈⋃𝒯. ⟨y,x⟩ ∈ r
      with assms(2,4) have ∀a∈–Infimum(r,T).T∈𝒯″. ⟨y,a⟩ ∈ r using inf˙geq˙lo˙bnd
        by simp
```

```
    with assms(2,5) have ⟨y,i⟩ ∈ r by (rule inf˙geq˙lo˙bnd)
  } thus thesis by simp
qed
ultimately have HasAmaximum(r,⋂a∈⋃𝒯. r-–a″) by (rule inf˙glb)
then show HasAnInfimum(r,⋃𝒯) unfolding HasAnInfimum˙def by simp
from assms(2) ‹⋃𝒯 ≠ 0› I J show i = Infimum(r,⋃𝒯) by (rule inf˙glb)
qed
```

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

```
lemma sup˙sup:
  assumes
    r ⊆ X×X antisym(r) trans(r)
    ∀T∈𝒯. HasAsupremum(r,T)
    HasAsupremum(r,–Supremum(r,T).T∈𝒯″)
  shows
      HasAsupremum(r,⋃𝒯) and Supremum(r,–Supremum(r,T).T∈𝒯″) = Supremum(r,⋃𝒯)
proof -
  let s = Supremum(r,–Supremum(r,T).T∈𝒯″)
  note assms(2)
  moreover from assms(4,5) have ⋃𝒯 ≠ 0 using set˙sup˙not˙empty by blast
  moreover
  have ∀T∈𝒯.∀t∈T. ⟨t,s⟩ ∈ r
  proof -
    { fix T t assume T∈𝒯 t∈T
      with assms(1,2,4) have ⟨t,Supremum(r,T)⟩ ∈ r
        unfolding HasAsupremum˙def using sup˙in˙space(2) by blast
      moreover from assms(1,2,5) ‹T∈𝒯› have ⟨Supremum(r,T),s⟩ ∈ r
        unfolding HasAsupremum˙def using sup˙in˙space(2) by blast
      moreover note assms(3)
      ultimately have ⟨t,s⟩ ∈ r unfolding trans˙def by blast
    } thus thesis by simp
  qed
  hence I: ∀t∈⋃𝒯. ⟨t,s⟩ ∈ r by auto
  moreover have J: ∀y. (∀x∈⋃𝒯. ⟨x,y⟩ ∈ r) ⟶ ⟨s,y⟩ ∈ r
  proof -
    { fix y x assume A: ∀x∈⋃𝒯. ⟨x,y⟩ ∈ r
    with assms(2,4) have ∀a∈–Supremum(r,T).T∈𝒯″. ⟨a,y⟩ ∈ r using sup˙leq˙up˙bnd
        by simp
      with assms(2,5) have ⟨s,y⟩ ∈ r by (rule sup˙leq˙up˙bnd)
    } thus thesis by simp
  qed
  ultimately have HasAminimum(r,⋂a∈⋃𝒯. r–a″) by (rule Order˙ZF˙5˙L5)
  then show HasAsupremum(r,⋃𝒯) unfolding HasAsupremum˙def by simp
  from assms(2) ‹⋃𝒯 ≠ 0› I J show s = Supremum(r,⋃𝒯) by (rule Order˙ZF˙5˙L5)
qed
```

If the relation is a linear order then for any element y smaller than the

supremum of a set we can find one element of the set that is greater than y.

```
lemma Order˙ZF˙5˙L8:
  assumes A1: r ⊆ X×X  and A2: IsLinOrder(X,r) and
  A3: r –is complete″ and
  A4: A⊆X  A≠0 and A5: ∃x∈X. ∀y∈A. ⟨y,x⟩ ∈ r and
  A6: ⟨y,Supremum(r,A)⟩ ∈ r   y ≠ Supremum(r,A)
  shows ∃z∈A. ⟨y,z⟩ ∈ r ∧ y ≠ z
proof -
  from A2 have
    I: antisym(r) and
    II: trans(r) and
    III: r –is total on″ X
    using IsLinOrder˙def by auto
  from A1 A6 have T1: y∈X by auto
  { assume A7: ∀z ∈ A. ⟨y,z⟩ ∉ r ∨ y=z
    from A4 I have antisym(r) and A≠0 by auto
    moreover have ∀x∈A. ⟨x,y⟩ ∈ r
    proof
      fix x assume A8: x∈A
      with A4 have T2: x∈X by auto
      from A7 A8 have ⟨y,x⟩ ∉ r ∨ y=x by simp
      with III T1 T2 show ⟨x,y⟩ ∈ r
 using IsTotal˙def total˙is˙refl refl˙def by auto
    qed
    moreover have ∀u. (∀x∈A. ⟨x,u⟩ ∈ r) ⟶ ⟨y,u⟩ ∈ r
    proof-
      { fix u assume A9: ∀x∈A. ⟨x,u⟩ ∈ r
 from A4 A5 have IsBoundedAbove(A,r) and A≠0
   using IsBoundedAbove˙def by auto
 with  A3 A4 A6 I A9  have
   ⟨y,Supremum(r,A)⟩ ∈ r ∧ ⟨Supremum(r,A),u⟩ ∈ r
   using IsComplete˙def Order˙ZF˙5˙L3 by simp
 with II have ⟨y,u⟩ ∈ r by (rule Fol1˙L3)
      } then show ∀u. (∀x∈A. ⟨x,u⟩ ∈ r) ⟶ ⟨y,u⟩ ∈ r
 by simp
    qed
    ultimately have y = Supremum(r,A)
      by (rule Order˙ZF˙5˙L5)
    with A6 have False by simp
  } then show ∃z∈A. ⟨y,z⟩ ∈ r ∧ y ≠ z by auto
qed
```

7.3 Strict versions of order relations

One of the problems with translating formalized mathematics from Metamath to IsarMathLib is that Metamath uses strict orders (of the $<$ type) while in IsarMathLib we mostly use nonstrict orders (of the $\leq$ type). This doesn't really make any difference, but is annoying as we have to prove

many theorems twice. In this section we prove some theorems to make it easier to translate the statements about strict orders to statements about the corresponding non-strict order and vice versa.

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

```
definition
  StrictVersion(r) ≡ r - –⟨x,x⟩. x ∈ domain(r)″
```

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

```
lemma def˙of˙strict˙ver: shows
  ⟨x,y⟩ ∈ StrictVersion(r) ⟷ ⟨x,y⟩ ∈ r ∧ x≠y
  using StrictVersion˙def domain˙def by auto
```

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

```
lemma strict˙of˙antisym:
  assumes A1: antisym(r) and A2: ⟨a,b⟩ ∈ StrictVersion(r)
  shows ⟨b,a⟩ ∉ StrictVersion(r)
proof -
  { assume A3: ⟨b,a⟩ ∈ StrictVersion(r)
    with A2 have ⟨a,b⟩ ∈ r  and ⟨b,a⟩ ∈ r
      using def˙of˙strict˙ver by auto
    with A1 have a=b by (rule Fol1˙L4)
    with A2 have False using def˙of˙strict˙ver
      by simp
  } then show ⟨b,a⟩ ∉ StrictVersion(r) by auto
qed
```

The strict version of totality.

```
lemma strict˙of˙tot:
  assumes r –is total on″ X and a∈X  b∈X  a≠b
  shows ⟨a,b⟩ ∈ StrictVersion(r) ∨ ⟨b,a⟩ ∈ StrictVersion(r)
  using assms IsTotal˙def def˙of˙strict˙ver by auto
```

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

```
lemma strict˙ans˙tot˙trich:
  assumes A1: antisym(r) and A2: r –is total on″ X
  and A3: a∈X  b∈X
  and A4: s = StrictVersion(r)
  shows Exactly˙1˙of˙3˙holds(⟨a,b⟩ ∈ s, a=b,⟨b,a⟩ ∈ s)
proof -
  let p = ⟨a,b⟩ ∈ s
  let q = a=b
  let r = ⟨b,a⟩ ∈ s
  from A2 A3 A4 have p ∨ q ∨ r
    using strict˙of˙tot by auto
  moreover from A1 A4 have p ⟶ ¬q ∧ ¬r
```

```
    using def˙of˙strict˙ver strict˙of˙antisym by simp
  moreover from A4 have q ⟶ ¬p ∧ ¬r
    using def˙of˙strict˙ver by simp
  moreover from A1 A4 have r ⟶ ¬p ∧ ¬q
    using def˙of˙strict˙ver strict˙of˙antisym by auto
  ultimately show Exactly˙1˙of˙3˙holds(p, q, r)
    by (rule Fol1˙L5)
qed
```

A trichotomy law for linear order. This is a special case of strict˙ans˙tot˙trich.

```
corollary strict˙lin˙trich: assumes A1: IsLinOrder(X,r) and
  A2: a∈X  b∈X and
  A3: s = StrictVersion(r)
  shows Exactly˙1˙of˙3˙holds(⟨a,b⟩ ∈ s, a=b,⟨b,a⟩ ∈ s)
  using assms IsLinOrder˙def strict˙ans˙tot˙trich by auto
```

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

```
lemma geq˙impl˙not˙less:
  assumes A1: antisym(r) and A2: ⟨a,b⟩ ∈ r
  shows ⟨b,a⟩ ∉ StrictVersion(r)
proof -
  { assume A3: ⟨b,a⟩ ∈  StrictVersion(r)
    with A2 have ⟨a,b⟩ ∈ StrictVersion(r)
      using def˙of˙strict˙ver by auto
    with A1 A3 have False using strict˙of˙antisym
      by blast
  } then show ⟨b,a⟩ ∉ StrictVersion(r) by auto
qed
```

If an antisymmetric relation is transitive, then the strict version is also transitive, an explicit version strict˙of˙transB below.

```
lemma strict˙of˙transA:
  assumes A1: trans(r) and A2: antisym(r) and
  A3: s= StrictVersion(r) and  A4: ⟨a,b⟩ ∈ s  ⟨b,c⟩ ∈ s
  shows ⟨a,c⟩ ∈ s
proof -
  from A3 A4 have I: ⟨a,b⟩ ∈ r ∧ ⟨b,c⟩ ∈ r
    using def˙of˙strict˙ver by simp
  with A1 have ⟨a,c⟩ ∈ r by (rule Fol1˙L3)
  moreover
  { assume a=c
    with I have ⟨a,b⟩ ∈ r and ⟨b,a⟩ ∈ r by auto
    with A2 have a=b by (rule Fol1˙L4)
    with A3 A4 have False using def˙of˙strict˙ver by simp
  } then have a≠c by auto
  ultimately have  ⟨a,c⟩ ∈ StrictVersion(r)
    using def˙of˙strict˙ver by simp
```

```
  with A3 show thesis by simp
qed
```

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

```
lemma strict˙of˙transB:
  assumes A1: trans(r) and A2: antisym(r)
  shows trans(StrictVersion(r))
proof -
  let s = StrictVersion(r)
  from A1 A2 have
    ∀ x y z. ⟨x, y⟩ ∈ s ∧ ⟨y, z⟩ ∈ s ⟶ ⟨x, z⟩ ∈ s
    using strict˙of˙transA by blast
  then show trans(StrictVersion(r)) by (rule Fol1˙L2)
qed
```

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

```
lemma strict˙of˙compl:
  assumes A1: r ⊆ X×X and A2: IsLinOrder(X,r) and
  A3: r –is complete″ and
  A4: A⊆X  A≠0 and A5: s = StrictVersion(r) and
  A6: ∃u∈X. ∀y∈A. ⟨y,u⟩ ∈ s
  shows
  ∃x∈X. ( ∀y∈A. ⟨x,y⟩ ∉ s ) ∧ (∀y∈X. ⟨y,x⟩ ∈ s ⟶ (∃z∈A. ⟨y,z⟩ ∈ s))
proof -
  let x = Supremum(r,A)
  from A2 have I: antisym(r) using IsLinOrder˙def
    by simp
  moreover from A5 A6 have ∃u∈X. ∀y∈A. ⟨y,u⟩ ∈ r
    using def˙of˙strict˙ver by auto
  moreover note A1 A3 A4
  ultimately have II: x ∈ X   ∀y∈A. ⟨y,x⟩ ∈ r
    using Order˙ZF˙5˙L7 by auto
  then have III: ∃x∈X. ∀y∈A. ⟨y,x⟩ ∈ r by auto
  from A5 I II have x ∈ X   ∀y∈A. ⟨x,y⟩ ∉ s
    using geq˙impl˙not˙less by auto
  moreover from A1 A2 A3 A4 A5 III have
    ∀y∈X. ⟨y,x⟩ ∈ s ⟶ (∃z∈A. ⟨y,z⟩ ∈ s)
    using def˙of˙strict˙ver Order˙ZF˙5˙L8 by simp
  ultimately show
    ∃x∈X. ( ∀y∈A. ⟨x,y⟩ ∉ s ) ∧ (∀y∈X. ⟨y,x⟩ ∈ s ⟶ (∃z∈A. ⟨y,z⟩ ∈ s))
    by auto
qed
```

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

```
lemma strict˙ver˙rel: assumes A1: r ⊆ A×A
  shows StrictVersion(r) ⊆ A×A
```

using assms StrictVersion˙def by auto

end

8 Functions - introduction

theory func1 imports ZF.func Fol1 ZF1

begin

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

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

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

We define the notion of function that preserves a collection here. Given two collection of sets a function preserves the collections if the inverse image of sets in one collection belongs to the second one. This notion does not have a name in romantic math. It is used to define continuous functions in Topology˙ZF˙2 theory. We define it here so that we can use it for other purposes, like defining measurable functions. Recall that f-(A) means the inverse image of the set A.

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

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

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

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

lemma func1˙1˙L1: assumes f:A→C shows domain(f) = A
using assms domain˙of˙fun by simp

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

```
lemma fun˙is˙fun: assumes f:X→Y shows function(f)
  using assms fun˙is˙function by simp
```

A lemma explains what fstdom is for.

```
lemma fstdomdef: assumes A1: f: X×Y → Z and A2: Y≠0
  shows fstdom(f) = X
proof -
  from A1 have domain(f) = X×Y using func1˙1˙L1
    by simp
  with A2 show fstdom(f) = X unfolding fstdom˙def by auto
qed
```

A version of the Pi˙type lemma from the standard Isabelle/ZF library.

```
lemma func1˙1˙L1A: assumes A1: f:X→Y and A2: ∀x∈X. f(x) ∈ Z
  shows f:X→Z
proof -
  { fix x assume x∈X
    with A2 have f(x) ∈ Z by simp }
  with A1 show f:X→Z by (rule Pi˙type)
qed
```

A variant of func1˙1˙L1A.

```
lemma func1˙1˙L1B: assumes A1: f:X→Y and A2: Y⊆Z
  shows f:X→Z
proof -
  from A1 A2 have ∀x∈X. f(x) ∈ Z
    using apply˙funtype by auto
  with A1 show  f:X→Z using func1˙1˙L1A by blast
qed
```

There is a value for each argument.

```
lemma func1˙1˙L2: assumes A1: f:X→Y  x∈X
  shows ∃y∈Y. ⟨x,y⟩ ∈ f
proof-
  from A1 have f(x) ∈ Y using apply˙type by simp
  moreover from A1 have ⟨ x,f(x)⟩∈ f using apply˙Pair by simp
  ultimately show thesis by auto
qed
```

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

```
lemma vimage˙converse: shows r-(A) = converse(r)(A)
  using vimage˙iff image˙iff converse˙iff by auto
```

The image is the inverse image of converse.

```
lemma image˙converse: shows converse(r)-(A) = r(A)
  using vimage˙iff image˙iff converse˙iff by auto
```

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

```
lemma vimage˙comp: shows (r O s)-(A) = s-(r-(A))
  using vimage˙converse converse˙comp image˙comp image˙converse by simp
```

A version of vimage˙comp for three functions.

```
lemma vimage˙comp3: shows (r O s O t)-(A) = t-(s-(r-(A)))
  using vimage˙comp by simp
```

Inverse image of any set is contained in the domain.

```
lemma func1˙1˙L3: assumes A1: f:X→Y shows f-(D) ⊆ X
proof-
   have ∀x. x∈f-(D) ⟶ x ∈ domain(f)
      using  vimage˙iff domain˙iff by auto
    with A1 have ∀x. (x ∈ f-(D)) ⟶ (x∈X) using func1˙1˙L1 by simp
    then show thesis by auto
qed
```

The inverse image of the range is the domain.

```
lemma func1˙1˙L4: assumes f:X→Y shows f-(Y) = X
  using assms func1˙1˙L3 func1˙1˙L2 vimage˙iff by blast
```

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

```
lemma func1˙1˙L5:
  assumes A1: ⟨ x,y⟩ ∈ f and A2: f:X→Y
  shows x∈X ∧ y∈Y
proof
  from A1 A2 show x∈X using apply˙iff by simp
  with A2 have f(x)∈ Y using apply˙type by simp
  with A1 A2 show y∈Y using apply˙iff by simp
qed
```

Function is a subset of cartesian product.

```
lemma fun˙subset˙prod: assumes A1: f:X→Y shows f ⊆ X×Y
proof
  fix p assume p ∈ f
  with A1 have ∃x∈X. p = ⟨x, f(x)⟩
    using Pi˙memberD by simp
  then obtain x where I: p = ⟨x, f(x)⟩
    by auto
  with A1 ‹p ∈ f› have x∈X ∧ f(x) ∈ Y
    using func1˙1˙L5 by blast
  with I show p ∈ X×Y by auto
qed
```

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

```
lemma func1˙1˙L5A:
  assumes A1: f:X→Y  x∈X  y = f(x)
  shows ⟨x,y⟩ ∈ f  y ∈ range(f)
proof -
```

```
  from A1 show ⟨x,y⟩ ∈ f using apply˙Pair by simp
  then show y ∈ range(f) using rangeI by simp
qed
```

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

```
theorem fun˙is˙set˙of˙pairs: assumes A1: f:X→Y
  shows f = −⟨x, f(x)⟩. x ∈ X″
proof
  from A1 show −⟨x, f(x)⟩. x ∈ X″ ⊆ f using func1˙1˙L5A
    by auto
next
  { fix p assume p ∈ f
    with A1 have p ∈ X×Y using fun˙subset˙prod
      by auto
    with A1 ‹p ∈ f› have p ∈ −⟨x, f(x)⟩. x ∈ X″
      using apply˙equality by auto
  } thus f ⊆ −⟨x, f(x)⟩. x ∈ X″ by auto
qed
```

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

```
lemma func1˙1˙L5B:
  assumes  A1: f:X→Y shows range(f) ⊆ Y
proof
  fix y assume y ∈ range(f)
  then obtain x where ⟨ x,y⟩ ∈ f
    using range˙def converse˙def domain˙def by auto
  with A1 show y∈Y using func1˙1˙L5 by blast
qed
```

The image of any set is contained in the range.

```
lemma func1˙1˙L6: assumes A1: f:X→Y
  shows f(B) ⊆ range(f) and f(B) ⊆ Y
proof -
  show f(B) ⊆ range(f) using image˙iff rangeI by auto
  with A1 show f(B) ⊆ Y using func1˙1˙L5B by blast
qed
```

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

```
lemma func1˙1˙L6A: assumes A1: f:X→Y shows f-(A)⊆X
proof
  fix x
  assume A2: x∈f-(A) then obtain y where ⟨ x,y⟩ ∈ f
    using vimage˙iff by auto
  with A1 show  x∈X using func1˙1˙L5 by fast
qed
```

Image of a greater set is greater.

```
lemma func1˙1˙L8: assumes A1: A⊆B  shows f(A)⊆ f(B)
```

```
  using assms image˙Un by auto
```

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

```
lemma func1˙1˙L9: assumes A1: f:X→Y and A2: A⊆X
  shows A ⊆ f-(f(A))
proof -
  from A1 A2 have ∀x∈A. ⟨ x,f(x)⟩ ∈ f  using apply˙Pair by auto
  then show thesis using image˙iff by auto
qed
```

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

```
lemma inv˙im˙dom: assumes A1: f:X→Y shows f-(f(X)) = X
proof
  from A1 show f-(f(X)) ⊆ X using func1˙1˙L3 by simp
  from A1 show X ⊆ f-(f(X)) using func1˙1˙L9 by simp
qed
```

A technical lemma needed to make the func1˙1˙L11 proof more clear.

```
lemma func1˙1˙L10:
  assumes A1: f ⊆ X×Y and A2: ∃!y. (y∈Y ∧ ⟨x,y⟩ ∈ f)
  shows ∃!y. ⟨x,y⟩ ∈ f
proof
  from A2 show ∃y. ⟨x, y⟩ ∈ f by auto
  fix y n assume ⟨x,y⟩ ∈ f and ⟨x,n⟩ ∈ f
  with A1 A2 show y=n by auto
qed
```

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

```
lemma func1˙1˙L11:
  assumes f ⊆ X×Y and ∀x∈X. ∃!y. y∈Y ∧ ⟨x,y⟩ ∈ f
  shows f: X→Y using assms func1˙1˙L10 Pi˙iff˙old by simp
```

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

```
lemma func1˙1˙L11A: assumes A1: ∀x∈X. b(x) ∈ Y
  shows –⟨ x,y⟩ ∈ X×Y. b(x) = y″ : X→Y
proof -
  let f = –⟨ x,y⟩ ∈ X×Y. b(x) = y″
  have f ⊆ X×Y by auto
  moreover have ∀x∈X. ∃!y. y∈Y ∧ ⟨ x,y⟩ ∈ f
  proof
    fix x assume A2: x∈X
```

```
    show ∃!y. y∈Y ∧ ⟨x, y⟩ ∈ −⟨x,y⟩ ∈ X×Y . b(x) = y″
    proof
      from A2 A1 show
        ∃y. y∈Y ∧ ⟨x, y⟩ ∈ −⟨x,y⟩ ∈ X×Y . b(x) = y″
 by simp
    next
      fix y y1
      assume y∈Y ∧ ⟨x, y⟩ ∈ −⟨x,y⟩ ∈ X×Y . b(x) = y″
 and y1∈Y ∧ ⟨x, y1⟩ ∈ −⟨x,y⟩ ∈ X×Y . b(x) = y″
      then show y = y1 by simp
    qed
  qed
  ultimately show −⟨ x,y⟩ ∈ X×Y. b(x) = y″ : X→Y
    using func1˙1˙L11 by simp
qed
```

The next lemma will replace func1˙1˙L11A one day.

```
lemma ZF˙fun˙from˙total: assumes A1: ∀x∈X. b(x) ∈ Y
  shows −⟨x,b(x)⟩. x∈X″ : X→Y
proof -
  let f = −⟨x,b(x)⟩. x∈X″
  { fix x assume A2: x∈X
    have ∃!y. y∈Y ∧ ⟨x, y⟩ ∈ f
    proof
 from A1 A2 show ∃y. y∈Y ∧ ⟨x, y⟩ ∈ f
 by simp
    next fix y y1 assume y∈Y ∧ ⟨x, y⟩ ∈ f
 and y1∈Y ∧ ⟨x, y1⟩ ∈ f
      then show y = y1 by simp
    qed
  } then have ∀x∈X. ∃!y. y∈Y ∧ ⟨ x,y⟩ ∈ f
    by simp
  moreover from A1 have f ⊆ X×Y by auto
  ultimately show thesis using func1˙1˙L11
    by simp
qed
```

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

```
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
```

Identical meaning as ZF˙fun˙from˙tot˙val, but phrased a bit differently.

```
lemma ZF˙fun˙from˙tot˙val0:
  assumes f:X→Y and f = −⟨x,b(x)⟩. x∈X″
  shows ∀x∈X. f(x) = b(x)
  using assms ZF˙fun˙from˙tot˙val by simp
```

Another way of expressing that lambda expression is a function.

```
lemma lam˙is˙fun˙range: assumes f=−⟨x,g(x)⟩. x∈X″
  shows f:X→range(f)
proof -
  have ∀x∈X. g(x) ∈ range(−⟨x,g(x)⟩. x∈X″) unfolding range˙def
    by auto
 then have −⟨x,g(x)⟩. x∈X″ : X→range(−⟨x,g(x)⟩. x∈X″) by (rule ZF˙fun˙from˙total)
  with assms show thesis by auto
qed
```

Yet another way of expressing value of a function.

```
lemma ZF˙fun˙from˙tot˙val1:
  assumes x∈X shows −⟨x,b(x)⟩. x∈X″(x)=b(x)
proof -
  let f = −⟨x,b(x)⟩. x∈X″
  have f:X→range(f) using lam˙is˙fun˙range by simp
  with assms show thesis using ZF˙fun˙from˙tot˙val0 by simp
qed
```

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 assms show g : X∪A∪–a″ → Y∪B∪–c″
    using func1˙1˙L11C func1˙1˙L11D by simp
  from A1 A2 A3 have I:
    f:X→Y  ∀x∈A. b(x)∈B  X∩A = 0  h = f ∪ –⟨x,b(x)⟩. x∈A″
    by auto
  from assms have
    II: h : X∪A → Y∪B  a∉ X∪A  g = h ∪ –⟨a,c⟩″
    using func1˙1˙L11C by auto
  then have III: ∀x∈X∪A. g(x) = h(x) by (rule func1˙1˙L11D)
  moreover from I have  ∀x∈X. h(x) = f(x)
    by (rule func1˙1˙L11C)
  ultimately show ∀x∈X. g(x) = f(x) by simp
  from I have ∀x∈A. h(x) = b(x) by (rule func1˙1˙L11C)
  with III show ∀x∈A. g(x) = b(x) by simp
  from II show g(a) = c by (rule func1˙1˙L11D)
qed
```

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

```
lemma fun˙union˙overlap: assumes ∀x∈A∩B. h(x) ∈ Y  ∀x∈A-B. f(x) ∈ Y  ∀x∈B-A.
g(x) ∈ Y
  shows –⟨x,if x∈A-B then f(x) else if x∈B-A then g(x) else h(x)⟩. x ∈ A∪B″: A∪B
→ Y
proof -
  let F = –⟨x,if x∈A-B then f(x) else if x∈B-A then g(x) else h(x)⟩. x ∈ A∩B″
  from assms have ∀x∈A∪B. (if x∈A-B then f(x) else if x∈B-A then g(x) else h(x))
∈ Y
    by auto
  then show thesis by (rule ZF˙fun˙from˙total)
qed
```

Inverse image of intersection is the intersection of inverse images.

```
lemma invim˙inter˙inter˙invim: assumes f:X→Y
  shows f-(A∩B) = f-(A) ∩ f-(B)
  using assms fun˙is˙fun function˙vimage˙Int by simp
```

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

```
lemma func1˙1˙L12:
  assumes A1: B ⊆ Pow(Y) and A2: B≠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
```

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

```
lemma inv˙im˙inter˙im: assumes f:X→Y
  shows f-(A ∩ f(X)) = f-(A)
  using assms invim˙inter˙inter˙invim inv˙im˙dom func1˙1˙L6A
  by blast
```

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

```
lemma func1˙1˙L13: assumes A1:f-(A) ≠ 0 shows A≠0
  using assms by auto
```

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

```
lemma func1˙1˙L13A: assumes A1: f(A)≠0 shows A≠0
  using assms by auto
```

What is the inverse image of a singleton?

```
lemma func1˙1˙L14: assumes f∈X→Y
  shows f-(–y″) = –x∈X. f(x) = y″
  using assms func1˙1˙L6A vimage˙singleton˙iff apply˙iff by auto
```

A lemma that can be used instead fun˙extension˙iff to show that two functions are equal

```
lemma func˙eq: assumes f: X→Y  g: X→Z
  and  ∀x∈X. f(x) = g(x)
  shows f = g using assms fun˙extension˙iff by simp
```

Function defined on a singleton is a single pair.

```
lemma func˙singleton˙pair: assumes A1: f : –a″→X
  shows f = –⟨a, f(a)⟩″
proof -
  let g = –⟨a, f(a)⟩″
```

```
  note A1
  moreover have g : –a″ → –f(a)″ using singleton˙fun by simp
  moreover have ∀x ∈ –a″. f(x) = g(x) using singleton˙apply
    by simp
  ultimately show f = g by (rule func˙eq)
qed
```

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

```
lemma pair˙func˙singleton: assumes A1: y ∈ Y
  shows –⟨x,y⟩″ : –x″ → Y
proof -
  have –⟨x,y⟩″ : –x″ → –y″ using singleton˙fun by simp
  moreover from A1 have –y″ ⊆ Y by simp
  ultimately show –⟨x,y⟩″ : –x″ → Y
    by (rule func1˙1˙L1B)
qed
```

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

```
lemma pair˙val: shows –⟨x,y⟩″(x) = y
  using singleton˙fun apply˙equality by simp
```

A more familiar definition of inverse image.

```
lemma func1˙1˙L15: assumes A1: f:X→Y
  shows f-(A) = –x∈X. f(x) ∈ A″
proof -
  have f-(A) = (⋃y∈A . f-–y″)
    by (rule vimage˙eq˙UN)
  with A1 show thesis using func1˙1˙L14 by auto
qed
```

A more familiar definition of image.

```
lemma func˙imagedef: assumes A1: f:X→Y and A2: A⊆X
  shows f(A) = –f(x). x ∈ A″
proof
  from A1 show f(A) ⊆ –f(x). x ∈ A″
    using image˙iff apply˙iff by auto
  show –f(x). x ∈ A″ ⊆ f(A)
  proof
    fix y assume y ∈ –f(x). x ∈ A″
    then obtain x where x∈A and  y = f(x)
      by auto
    with A1 A2 have ⟨x,y⟩ ∈ f using apply˙iff by force
    with A1 A2 ‹x∈A› show y ∈ f(A) using image˙iff by auto
  qed
qed
```

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

```
lemma image˙id˙same: assumes A⊆X shows id(X)(A) = A
  using assms id˙type id˙conv by auto
```

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

```
lemma vimage˙id˙same: assumes A⊆X shows id(X)-(A) = A
  using assms id˙type id˙conv by auto
```

What is the image of a singleton?

```
lemma singleton˙image:
  assumes f∈X→Y and x∈X
  shows f–x″ = –f(x)″
  using assms func˙imagedef by auto
```

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

```
lemma func1˙1˙L15D: assumes f:X→Y  x∈A  A⊆X
  shows f(x) ∈ f(A)
  using assms func˙imagedef by auto
```

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

```
lemma range˙image˙domain:
  assumes A1: f:X→Y shows f(X) = range(f)
proof
  show f(X) ⊆ range(f) using image˙def by auto
  { fix y assume y ∈ range(f)
    then obtain x where ⟨y,x⟩ ∈ converse(f) by auto
    with A1 have x∈X using func1˙1˙L5 by blast
    with A1 have f(x) ∈ f(X) using func˙imagedef
      by auto
    with A1  ‹⟨y,x⟩ ∈ converse(f)› have y ∈ f(X)
      using apply˙equality by auto
  } then show range(f) ⊆ f(X) by auto
qed
```

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

```
lemma diff˙image˙diff: assumes A1: f: X→Y and A2: A⊆X
  shows f(X) - f(A) ⊆ f(X-A)
proof
  fix y assume y ∈ f(X) - f(A)
  hence y ∈ f(X) and I: y ∉ f(A) by auto
  with A1 obtain x where x∈X and II: y = f(x)
    using func˙imagedef by auto
  with A1 A2 I have x∉A
    using func1˙1˙L15D by auto
  with ‹x∈X› have x ∈ X-A X-A ⊆ X by auto
  with A1 II show y ∈ f(X-A)
```

```
  using func1˙1˙L15D by simp
qed
```

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

```
lemma image˙of˙Inter: assumes  A1: f:X→Y and
  A2: I≠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 union is the union of images.

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

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 assms func˙imagedef by simp
```

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

```
lemma func1˙1˙L15C: assumes  A1: f:X→Y and A2: g:Y→Z
  and A3: A⊆X
  shows
  g(f(A)) =  –g(f(x)). x∈A″
  g(f(A)) = (g O f)(A)
proof -
  from A1 A3 have –f(x). x∈A″ ⊆ Y
    using apply˙funtype by auto
  with A2 have g–f(x). x∈A″ = –g(f(x)). x∈A″
    using func˙imagedef by auto
  with A1 A3 show I: g(f(A)) =  –g(f(x)). x∈A″
    using func˙imagedef by simp
  from A1 A3 have ∀x∈A. (g O f)(x) = g(f(x))
    using comp˙fun˙apply by auto
  with I have g(f(A)) = –(g O f)(x). x∈A″
    by simp
  moreover from A1 A2 A3 have (g O f)(A) = –(g O f)(x). x∈A″
    using comp˙fun func˙imagedef by blast
  ultimately show g(f(A)) = (g O f)(A)
    by simp
qed
```

What is the image of a set defined by a meta-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
```

A composition of functions is a function. This is a slight generalization of standard Isabelle's comp˙fun

```
lemma comp˙fun˙subset:
  assumes A1: g:A→B  and A2: f:C→D and A3: B ⊆ C
  shows f O g : A → D
proof -
  from A1 A3 have g:A→C by (rule func1˙1˙L1B)
  with A2 show f O g : A → D using comp˙fun by simp
qed
```

This lemma supersedes the lemma comp˙eq˙id˙iff in Isabelle/ZF. Contributed by Victor Porton.

```
lemma comp˙eq˙id˙iff1: assumes A1: g: B→A and A2: f: A→C
  shows (∀y∈B. f(g(y)) = y) ⟷ f O g = id(B)
proof -
  from assms have f O g: B→C and id(B): B→B
    using comp˙fun id˙type by auto
  then have (∀y∈B. (f O g)y = id(B)(y)) ⟷ f O g = id(B)
    by (rule fun˙extension˙iff)
  moreover from A1 have
    ∀y∈B. (f O g)y = f(gy) and ∀y∈B. id(B)(y) = y
    by auto
  ultimately show (∀y∈B. f(gy) = y) ⟷ f O g = id(B) by simp
qed
```

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

```
lemma fun˙Union˙apply: assumes A1: ⋃F : X→Y and
  A2: f∈F and A3: f:A→B and A4: x∈A
  shows (⋃F)(x) = f(x)
proof -
  from A3 A4 have ⟨x, f(x)⟩ ∈ f using apply˙Pair
    by simp
  with A2 have ⟨x, f(x)⟩ ∈ ⋃F by auto
  with A1 show (⋃F)(x) = f(x) using apply˙equality
    by simp
qed
```

8.2 Functions restricted to a set

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

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

```
lemma func1˙2˙L1: assumes A1: f:X→Y and A2: B⊆X
  shows restrict(f,B)-(A) = f-(A) ∩ B
proof -
  let g = restrict(f,B)
  from A1 A2 have g:B→Y
    using restrict˙type2 by simp
  with A2 A1 show g-(A) = f-(A) ∩ B
    using func1˙1˙L15 restrict˙if by auto
qed
```

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

```
lemma func1˙2˙L2:
  assumes A1: f:X→Y and A2: g ∈ A→Z
  and A3: A⊆X and A4: f ∩ A×Z = g
  shows ∀x∈A. g(x) = f(x)
proof
  fix x assume x∈A
  with A2 have ⟨ x,g(x)⟩ ∈ g using apply˙Pair by simp
  with A4 A1 show g(x) = f(x)  using apply˙iff by auto
qed
```

Here is the actual criterion.

```
lemma func1˙2˙L3:
  assumes A1: f:X→Y and A2: g:A→Z
  and A3: A⊆X and A4: f ∩ A×Z = g
  shows g = restrict(f,A)
proof
  from A4 show g ⊆ restrict(f, A) using restrict˙iff by auto
  show restrict(f, A) ⊆ g
  proof
    fix z assume A5:z ∈ restrict(f,A)
    then obtain x y where D1:z∈f ∧ x∈A  ∧ z = ⟨x,y⟩
      using restrict˙iff by auto
    with A1 have y = f(x) using apply˙iff by auto
    with A1 A2 A3 A4 D1 have y = g(x) using func1˙2˙L2 by simp
    with A2 D1 show z∈g using apply˙Pair by simp
  qed
qed
```

Which function space a restricted function belongs to?

```
lemma func1˙2˙L4:
  assumes A1: f:X→Y and A2: A⊆X and A3: ∀x∈A. f(x) ∈ Z
  shows restrict(f,A) : A→Z
proof -
  let g = restrict(f,A)
```

```
  from A1 A2 have g : A→Y
    using restrict˙type2 by simp
  moreover {
    fix x assume x∈A
    with A1 A3 have g(x) ∈ Z using restrict by simp}
  ultimately show thesis by (rule Pi˙type)
qed
```

A simpler case of func1˙2˙L4, where the range of the original and restricted function are the same.

```
corollary restrict˙fun: assumes A1: f:X→Y and A2: A⊆X
  shows restrict(f,A) : A → Y
proof -
  from assms have ∀x∈A. f(x) ∈ Y using apply˙funtype
    by auto
  with assms show thesis using func1˙2˙L4 by simp
qed
```

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

```
lemma comp˙restrict:
  assumes A1: f : A→B and A2: g : X → C and A3: B⊆X
  shows g O f = restrict(g,B) O f
proof -
  from assms have g O f : A → C using comp˙fun˙subset
    by simp
  moreover from assms have restrict(g,B) O f : A → C
    using restrict˙fun comp˙fun by simp
  moreover from A1 have
    ∀x∈A. (g O f)(x) = (restrict(g,B) O f)(x)
    using comp˙fun˙apply apply˙funtype restrict
    by simp
  ultimately show g O f = restrict(g,B) O f
    by (rule func˙eq)
qed
```

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

```
lemma right˙comp˙id˙any: shows r O id(C) = restrict(r,C)
  unfolding restrict˙def by auto
```

8.3 Constant functions

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

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

```
definition
  ConstantFunction(X,c) ≡ X×−c″
```

Constant function 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
```

8.4 Injections, surjections, bijections etc.

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

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

```
lemma inj˙image˙dif:
  assumes A1: f ∈ inj(A,B) and A2: C ⊆ A
  shows f(A-C) = f(A) - f(C)
proof
  show f(A - C) ⊆ f(A) - f(C)
  proof
    fix y assume A3: y ∈ f(A - C)
    from A1 have f:A→B using inj˙def by simp
    moreover have A-C ⊆ A by auto
    ultimately have f(A-C) = –f(x). x ∈ A-C″
      using func˙imagedef by simp
    with A3 obtain x where I: f(x) = y and x ∈ A-C
      by auto
    hence x∈A by auto
    with ‹f:A→B› I have y ∈ f(A)
      using func˙imagedef by auto
    moreover have y ∉  f(C)
    proof -
      { assume y ∈ f(C)
 with A2 ‹f:A→B› obtain x0
   where II: f(x0) = y and x0 ∈ C
```

```
    using func˙imagedef by auto
  with A1 A2 I ‹x∈A› have
    f ∈ inj(A,B) f(x) = f(x0)  x∈A x0 ∈ A
    by auto
  then have x = x0 by (rule inj˙apply˙equality)
  with ‹x ∈ A-C› ‹x0 ∈ C› have False by simp
      } thus thesis by auto
    qed
    ultimately show y ∈ f(A) - f(C) by simp
  qed
  from A1 A2 show f(A) - f(C) ⊆ f(A-C)
    using inj˙def diff˙image˙diff by auto
qed
```

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

```
lemma inj˙image˙inter: assumes A1: f ∈ inj(X,Y) and A2: A⊆X B⊆X
  shows f(A∩B) = f(A) ∩ f(B)
proof
  show f(A∩B) ⊆ f(A) ∩ f(B) using image˙Int˙subset by simp
  { from A1 have f:X→Y using inj˙def by simp
    fix y assume y ∈ f(A) ∩ f(B)
    then have y ∈ f(A) and  y ∈ f(B) by auto
    with A2 ‹f:X→Y› obtain xA xB where
    xA ∈ A xB ∈ B and I: y = f(xA)  y = f(xB)
      using func˙imagedef by auto
    with A2 have xA ∈ X xB ∈ X and  f(xA) =  f(xB) by auto
    with A1 have xA = xB using inj˙def by auto
    with ‹xA ∈ A› ‹xB ∈ B› have f(xA) ∈ –f(x). x ∈ A∩B″ by auto
    moreover from A2 ‹f:X→Y› have f(A∩B) = –f(x). x ∈ A∩B″
      using func˙imagedef by blast
    ultimately have f(xA) ∈ f(A∩B) by simp
    with I have y ∈ f(A∩B) by simp
  } thus f(A) ∩ f(B) ⊆ f(A ∩ B) by auto
qed
```

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

```
lemma surj˙range˙image˙domain: assumes A1: f ∈ surj(A,B)
  shows f(A) = B
proof -
  from A1 have f(A) = range(f)
    using surj˙def range˙image˙domain by auto
  with A1 show f(A) = B  using surj˙range
    by simp
qed
```

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

```
lemma inj˙vimage˙image: assumes f ∈ inj(X,Y) and A⊆X
  shows f-(f(A)) = A
proof -
```

```
  have f-(f(A)) = (converse(f) O f)(A)
    using vimage˙converse image˙comp by simp
  with assms show thesis using left˙comp˙inverse image˙id˙same
    by simp
qed
```

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

```
lemma surj˙image˙vimage: assumes A1: f ∈ surj(X,Y) and A2: A⊆Y
  shows f(f-(A)) = A
proof -
  have f(f-(A)) = (f O converse(f))(A)
    using vimage˙converse image˙comp by simp
  with assms show thesis using right˙comp˙inverse image˙id˙same
    by simp
qed
```

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

```
lemma surj˙subsets: assumes A1: f ∈ surj(X,Y) and A2: B ⊆ Pow(Y)
  shows – f(U). U ∈ –f-(V). V∈B″ ″ = B
proof
  { fix W assume W ∈ – f(U). U ∈ –f-(V). V∈B″ ″
    then obtain U where I: U ∈ –f-(V). V∈B″ and II: W = f(U) by auto
    then obtain V where V∈B and U = f-(V) by auto
    with II have W = f(f-(V)) by simp
    moreover from assms ‹V∈B› have f ∈ surj(X,Y) and V⊆Y by auto
    ultimately have W=V using surj˙image˙vimage by simp
    with ‹V∈B› have W ∈ B by simp
  } thus – f(U). U ∈ –f-(V). V∈B″ ″ ⊆ B by auto
  { fix W assume W∈B
    let U = f-(W)
    from ‹W∈B› have U ∈ –f-(V). V∈B″ by auto
    moreover from A1 A2 ‹W∈B› have W = f(U) using surj˙image˙vimage by auto
    ultimately have W ∈ – f(U). U ∈ –f-(V). V∈B″ ″ by auto
  } thus B ⊆ – f(U). U ∈ –f-(V). V∈B″ ″ by auto
qed
```

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

```
lemma bij˙restrict˙rem:
  assumes A1: f ∈ bij(A,B) and A2: a∈A
  shows restrict(f, A-–a″) ∈ bij(A-–a″, B-–f(a)″)
proof -
  let C = A-–a″
  from A1 have f ∈ inj(A,B)  C ⊆ A
    using bij˙def by auto
  then have restrict(f,C) ∈ bij(C, f(C))
    using restrict˙bij by simp
  moreover have f(C) =  B-–f(a)″
  proof -
```

```
    from A2 ‹f ∈ inj(A,B)› have f(C) = f(A) - f–a″
      using inj˙image˙dif by simp
    moreover from A1 have f(A) = B
      using bij˙def surj˙range˙image˙domain by auto
    moreover from A1 A2 have f–a″ = –f(a)″
      using bij˙is˙fun singleton˙image by blast
    ultimately show f(C) =  B-–f(a)″ by simp
  qed
  ultimately show thesis by simp
qed
```

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

```
lemma domain˙of˙bij:
  assumes A1: f ∈ bij(X,Y) shows domain(f) = X
proof -
  from A1 have f:X→Y using bij˙is˙fun by simp
  then show domain(f) = X using func1˙1˙L1 by simp
qed
```

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

```
lemma inj˙inv˙back˙in˙set:
  assumes A1: f ∈ inj(A,B) and A2: C⊆A and A3: y ∈ f(C)
  shows
  converse(f)(y) ∈ C
  f(converse(f)(y)) = y
proof -
  from A1 have I: f:A→B using inj˙is˙fun by simp
  with A2 A3 obtain x where II: x∈C   y = f(x)
    using func˙imagedef by auto
  with A1 A2 show converse(f)(y) ∈ C using left˙inverse
    by auto
  from A1 A2 I II show f(converse(f)(y)) = y
    using func1˙1˙L5A right˙inverse by auto
qed
```

For 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
```

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

```
lemma inj˙inj˙range: assumes f ∈ inj(A,B)
  shows f ∈ inj(A,range(f))
  using assms inj˙def range˙of˙fun by auto
```

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

```
lemma inj˙bij˙range: assumes f ∈ inj(A,B)
  shows f ∈ bij(A,range(f))
proof -
  from assms have f ∈ surj(A,range(f)) using inj˙def fun˙is˙surj
    by auto
  with assms show thesis using inj˙inj˙range bij˙def by simp
qed
```

A lemma about extending a surjection by one point.

```
lemma surj˙extend˙point:
  assumes A1: f ∈ surj(X,Y) and A2: a∉X and
  A3: g = f ∪ –⟨a,b⟩″
  shows g ∈ surj(X∪–a″,Y∪–b″)
proof -
  from A1 A2 A3 have g : X∪–a″ → Y∪–b″
    using surj˙def func1˙1˙L11D by simp
  moreover have ∀y ∈ Y∪–b″. ∃x ∈ X∪–a″. y = g(x)
  proof
    fix y assume y ∈  Y ∪ –b″
    then have y ∈ Y ∨ y = b by auto
    moreover
    { assume y ∈ Y
      with A1 obtain x where x∈X and y = f(x)
 using surj˙def by auto
      with A1 A2 A3 have x ∈  X∪–a″ and y = g(x)
 using surj˙def func1˙1˙L11D by auto
      then have ∃x ∈ X∪–a″. y = g(x) by auto }
    moreover
    { assume y = b
      with A1 A2 A3 have y = g(a)
 using surj˙def func1˙1˙L11D by auto
      then have ∃x ∈ X∪–a″. y = g(x) by auto }
    ultimately show ∃x ∈ X∪–a″. y = g(x)
      by auto
  qed
  ultimately show g ∈ surj(X∪–a″,Y∪–b″)
    using surj˙def by auto
qed
```

A lemma about extending an injection by one point. Essentially the same as standard Isabelle's inj˙extend.

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

A lemma about extending a bijection by one point.

```
lemma bij˙extend˙point: assumes f ∈ bij(X,Y) a∉X b∉Y
  shows (f ∪ –⟨a,b⟩″) ∈ bij(X∪–a″,Y∪–b″)
  using assms surj˙extend˙point inj˙extend˙point bij˙def
  by simp
```

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

```
lemma comp˙inv˙id˙eq:
  assumes A1: converse(b) O a = id(A) and
  A2: a ⊆ A×B b ∈ surj(A,B)
  shows a = b
proof -
  from A1 have (b O converse(b)) O a = b O id(A)
    using comp˙assoc by simp
  with A2 have id(B) O a = b O id(A)
    using right˙comp˙inverse by simp
  moreover
  from A2 have a ⊆ A×B and b ⊆ A×B
    using surj˙def fun˙subset˙prod
    by auto
  then have id(B) O a = a and b O id(A) = b
    using left˙comp˙id right˙comp˙id by auto
  ultimately show a = b by simp
qed
```

A special case of comp˙inv˙id˙eq - the $a^{-1}b = 1$ implies $a = b$ law for bijections.

```
lemma comp˙inv˙id˙eq˙bij:
  assumes A1: a ∈ bij(A,B) b ∈ bij(A,B) and
  A2: converse(b) O a = id(A)
  shows a = b
proof -
  from A1 have  a ⊆ A×B and b ∈ surj(A,B)
    using bij˙def surj˙def fun˙subset˙prod
    by auto
  with A2 show a = b by (rule comp˙inv˙id˙eq)
qed
```

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

```
lemma bij˙converse˙converse: assumes a ∈ bij(A,B)
  shows converse(converse(a)) = a
proof -
  from assms have a ⊆ A×B using bij˙def surj˙def fun˙subset˙prod by simp
  then show thesis using converse˙converse by simp
qed
```

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

```
lemma comp˙id˙conv: assumes A1: a ∈ bij(A,B) b ∈ bij(B,A) and
  A2: b O a = id(A)
  shows a = converse(b) and b = converse(a)
proof -
  from A1 have a ∈ bij(A,B) and converse(b) ∈ bij(A,B) using bij˙converse˙bij
    by auto
```

```
  moreover from assms have converse(converse(b)) O a = id(A)
    using bij˙converse˙converse by simp
  ultimately show a = converse(b) by (rule comp˙inv˙id˙eq˙bij)
  with assms show b = converse(a) using bij˙converse˙converse by simp
qed
```

A version of comp˙id˙conv with weaker assumptions.

```
lemma comp˙conv˙id: assumes A1: a ∈ bij(A,B) and A2: b:B→A and
  A3: ∀x∈A. b(a(x)) = x
  shows b ∈ bij(B,A) and  a = converse(b) and b = converse(a)
proof -
  have b ∈ surj(B,A)
  proof -
    have ∀x∈A. ∃y∈B. b(y) = x
    proof -
      { fix x assume x∈A
        let y = a(x)
        from A1 A3 ‹x∈A› have y∈B and b(y) = x
          using bij˙def inj˙def apply˙funtype by auto
        hence ∃y∈B. b(y) = x by auto
      } thus thesis by simp
    qed
    with A2 show b ∈ surj(B,A) using surj˙def by simp
  qed
  moreover have b ∈ inj(B,A)
  proof -
    have ∀w∈B.∀y∈B. b(w) = b(y) ⟶ w=y
    proof -
      { fix w y assume w∈B  y∈B and I: b(w) = b(y)
        from A1 have a ∈ surj(A,B) unfolding bij˙def by simp
        with ‹w∈B› obtain x_w where x_w ∈ A and II: a(x_w) = w
          using surj˙def by auto
        with I have b(a(x_w)) = b(y) by simp
        moreover from ‹a ∈ surj(A,B)› ‹y∈B› obtain x_y where
          x_y ∈ A and III: a(x_y) = y
          using surj˙def by auto
       moreover from A3 ‹x_w ∈ A›  ‹x_y ∈ A› have b(a(x_w)) = x_w and  b(a(x_y)) = x_y
          by auto
        ultimately have x_w = x_y by simp
        with II III have w=y by simp
      } thus thesis by auto
    qed
    with A2 show b ∈ inj(B,A) using inj˙def by auto
  qed
  ultimately show b ∈ bij(B,A) using bij˙def by simp
  from assms have b O a = id(A) using bij˙def inj˙def comp˙eq˙id˙iff1 by auto
  with A1 ‹b ∈ bij(B,A)› show a = converse(b) and b = converse(a)
    using comp˙id˙conv by auto
qed
```

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

```
lemma surj˙singleton˙image: assumes A1: f ∈ surj(X,Y)
  shows (⋃x∈X. –f(x)″) = Y
proof
  from A1 show (⋃x∈X. –f(x)″) ⊆ Y
    using surj˙def apply˙funtype by auto
next
  { fix y assume y ∈ Y
    with A1 have y ∈ (⋃x∈X. –f(x)″)
      using surj˙def by auto
  } then show  Y ⊆ (⋃x∈X. –f(x)″) by auto
qed
```

8.5 Functions of two variables

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

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

```
lemma cart˙prod˙fun: assumes f1:X1→Y1  f2:X2→Y2 and
  g = –⟨p,⟨f1(fst(p)),f2(snd(p))⟩⟩. p ∈ X1×X2″
  shows g: X1×X2 → Y1×Y2 using assms apply˙funtype  ZF˙fun˙from˙total by simp
```

A reformulation of cart˙prod˙fun above in a sligtly different notation.

```
lemma prod˙fun:
  assumes f:X1→X2  g:X3→X4
  shows –⟨⟨x,y⟩,⟨fx,gy⟩⟩. ⟨x,y⟩∈X1×X3″:X1×X3→X2×X4
proof -
  have –⟨⟨x,y⟩,⟨fx,gy⟩⟩. ⟨x,y⟩∈X1×X3″ = –⟨p,⟨f(fst(p)),g(snd(p))⟩⟩. p ∈ X1×X3″
    by auto
  with assms show thesis using cart˙prod˙fun by simp
qed
```

Product of two surjections is a surjection.

```
theorem prod˙functions˙surj:
  assumes f∈surj(A,B) g∈surj(C,D)
  shows –⟨⟨a1,a2⟩,⟨fa1,ga2⟩⟩.⟨a1,a2⟩∈A×C″ ∈ surj(A×C,B×D)
proof -
  let h = –⟨⟨x, y⟩, f(x), g(y)⟩ . ⟨x,y⟩ ∈ A × C″
  from assms have fun: f:A→Bg:C→D unfolding surj˙def by auto
  then have pfun: h : A × C → B × D using prod˙fun by auto
  {
```

```
    fix b assume b∈B×D
    then obtain b1 b2 where b=⟨b1,b2⟩ b1∈B b2∈D by auto
    with assms obtain a1 a2 where f(a1)=b1 g(a2)=b2 a1∈A a2∈C
      unfolding surj˙def by blast
    hence ⟨⟨a1,a2⟩,⟨b1,b2⟩⟩ ∈ h by auto
    with pfun have h⟨a1,a2⟩=⟨b1,b2⟩ using apply˙equality by auto
    with ‹b=⟨b1,b2⟩› ‹a1∈A› ‹a2∈C› have ∃a∈A×C. h(a)=b
      by auto
  } hence ∀b∈B×D. ∃a∈A×C. h(a) = b by auto
  with pfun show thesis unfolding surj˙def by auto
qed
```

For a function of two variables created from functions of one variable as in cart˙prod˙fun above, the inverse image of a cartesian product of sets is the cartesian product of inverse images.

```
lemma cart˙prod˙fun˙vimage: assumes f₁:X₁→Y₁  f₂:X₂→Y₂ and
  g = −⟨p,⟨f₁(fst(p)),f₂(snd(p))⟩⟩. p ∈ X₁×X₂″
  shows g-(A₁×A₂) = f₁-(A₁) × f₂-(A₂)
proof -
  from assms have g: X₁×X₂ → Y₁×Y₂ using cart˙prod˙fun
    by simp
  then have g-(A₁×A₂) = −p ∈ X₁×X₂. g(p) ∈ A₁×A₂″ using func1˙1˙L15
    by simp
  with assms ‹g: X₁×X₂ → Y₁×Y₂› show g-(A₁×A₂) = f₁-(A₁) × f₂-(A₂)
    using ZF˙fun˙from˙tot˙val func1˙1˙L15 by auto
qed
```

For a function of two variables defined on $X \times Y$, if we fix an $x \in X$ we obtain a function on Y. Note that if domain(f) is $X \times Y$, range(domain(f)) extracts Y from $X \times Y$.

```
definition
  Fix1stVar(f,x) ≡ −⟨y,f⟨x,y⟩⟩. y ∈ range(domain(f))″
```

For every $y \in Y$ we can fix the second variable in a binary function $f : X \times Y \to Z$ to get a function on X.

```
definition
  Fix2ndVar(f,y) ≡ −⟨x,f⟨x,y⟩⟩. x ∈ domain(domain(f))″
```

We defined Fix1stVar and Fix2ndVar so that the domain of the function is not listed in the arguments, but is recovered from the function. The next lemma is a technical fact that makes it easier to use this definition.

```
lemma fix˙var˙fun˙domain: assumes A1: f : X×Y → Z
  shows
  x∈X ⟶ Fix1stVar(f,x) = −⟨y,f⟨x,y⟩⟩. y ∈ Y″
  y∈Y ⟶ Fix2ndVar(f,y) = −⟨x,f⟨x,y⟩⟩. x ∈ X″
proof -
  from A1 have I: domain(f) = X×Y using func1˙1˙L1 by simp
```

```
  { assume x∈X
    with I have range(domain(f)) = Y by auto
    then have Fix1stVar(f,x) = −⟨y,f⟨x,y⟩⟩. y ∈ Y″
      using Fix1stVar˙def by simp
  } then show x∈X ⟶ Fix1stVar(f,x) = −⟨y,f⟨x,y⟩⟩. y ∈ Y″
    by simp
  { assume y∈Y
    with I have domain(domain(f)) = X by auto
    then have Fix2ndVar(f,y) = −⟨x,f⟨x,y⟩⟩. x ∈ X″
      using Fix2ndVar˙def by simp
  } then show y∈Y ⟶ Fix2ndVar(f,y) = −⟨x,f⟨x,y⟩⟩. x ∈ X″
    by simp
qed
```

If we fix the first variable, we get a function of the second variable.

```
lemma fix˙1st˙var˙fun: assumes A1: f : X×Y → Z and A2: x∈X
  shows Fix1stVar(f,x) : Y → Z
proof -
  from A1 A2 have ∀y∈Y. f⟨x,y⟩ ∈ Z
    using apply˙funtype by simp
  then have −⟨y,f⟨x,y⟩⟩. y ∈ Y″ :  Y → Z using ZF˙fun˙from˙total by simp
  with A1 A2 show Fix1stVar(f,x) : Y → Z using fix˙var˙fun˙domain by simp
qed
```

If we fix the second variable, we get a function of the first variable.

```
lemma fix˙2nd˙var˙fun: assumes A1: f : X×Y → Z and A2: y∈Y
  shows Fix2ndVar(f,y) : X → Z
proof -
  from A1 A2 have ∀x∈X. f⟨x,y⟩ ∈ Z
    using apply˙funtype by simp
  then have −⟨x,f⟨x,y⟩⟩. x ∈ X″ :  X → Z
    using ZF˙fun˙from˙total by simp
  with A1 A2 show Fix2ndVar(f,y) : X → Z
    using fix˙var˙fun˙domain by simp
qed
```

What is the value of Fix1stVar(f,x) at $y \in Y$ and the value of Fix2ndVar(f,y) at $x \in X$"?

```
lemma fix˙var˙val:
  assumes A1: f : X×Y → Z and A2: x∈X  y∈Y
  shows
  Fix1stVar(f,x)(y) = f⟨x,y⟩
  Fix2ndVar(f,y)(x) = f⟨x,y⟩
proof -
  let f1 = −⟨y,f⟨x,y⟩⟩. y ∈ Y″
  let f2 = −⟨x,f⟨x,y⟩⟩. x ∈ X″
  from A1 A2 have I:
    Fix1stVar(f,x) = f1
    Fix2ndVar(f,y) = f2
```

```
    using fix˙var˙fun˙domain by auto
  moreover from A1 A2 have
    Fix1stVar(f,x) : Y → Z
    Fix2ndVar(f,y) : X → Z
    using fix˙1st˙var˙fun fix˙2nd˙var˙fun by auto
  ultimately have f1 : Y → Z and  f2 : X → Z
    by auto
  with A2 have f1(y) = f⟨x,y⟩ and f2(x) = f⟨x,y⟩
    using ZF˙fun˙from˙tot˙val by auto
  with I show
    Fix1stVar(f,x)(y) = f⟨x,y⟩
    Fix2ndVar(f,y)(x) = f⟨x,y⟩
    by auto
qed
```

Fixing the second variable commutes with restrictig the domain.

```
lemma fix˙2nd˙var˙restr˙comm:
  assumes A1: f : X×Y → Z and A2: y∈Y and A3: X1 ⊆ X
  shows Fix2ndVar(restrict(f,X1×Y),y) = restrict(Fix2ndVar(f,y),X1)
proof -
  let g = Fix2ndVar(restrict(f,X1×Y),y)
  let h = restrict(Fix2ndVar(f,y),X1)
  from A3 have I: X1×Y ⊆ X×Y by auto
  with A1 have II: restrict(f,X1×Y) : X1×Y → Z
    using restrict˙type2 by simp
  with A2 have g : X1 → Z
    using fix˙2nd˙var˙fun by simp
  moreover
  from A1 A2 have III: Fix2ndVar(f,y) : X → Z
    using fix˙2nd˙var˙fun by simp
  with A3 have h : X1 → Z
    using restrict˙type2 by simp
  moreover
  { fix z assume A4: z ∈ X1
    with A2 I II have g(z) = f⟨z,y⟩
      using restrict fix˙var˙val by simp
    also from A1 A2 A3 A4 have f⟨z,y⟩ = h(z)
      using restrict fix˙var˙val by auto
    finally have g(z) = h(z) by simp
  } then have ∀z ∈ X1. g(z) = h(z) by simp
  ultimately show g = h by (rule func˙eq)
qed
```

The next lemma expresses the inverse image of a set by function with fixed first variable in terms of the original function.

```
lemma fix˙1st˙var˙vimage:
  assumes A1: f : X×Y → Z and A2: x∈X
  shows Fix1stVar(f,x)-(A) = −y∈Y. ⟨x,y⟩ ∈ f-(A)″
proof -
```

```
  from assms have Fix1stVar(f,x)-(A) = –y∈Y. Fix1stVar(f,x)(y) ∈ A″
    using fix˙1st˙var˙fun func1˙1˙L15 by blast
  with assms show thesis using fix˙var˙val func1˙1˙L15 by auto
qed
```

The next lemma expresses the inverse image of a set by function with fixed second variable in terms of the original function.

```
lemma fix˙2nd˙var˙vimage:
  assumes A1: f : X×Y → Z and A2: y∈Y
  shows Fix2ndVar(f,y)-(A) = –x∈X. ⟨x,y⟩ ∈ f-(A)″
proof -
  from assms have I: Fix2ndVar(f,y)-(A) = –x∈X. Fix2ndVar(f,y)(x) ∈ A″
    using fix˙2nd˙var˙fun func1˙1˙L15 by blast
  with assms show thesis using fix˙var˙val func1˙1˙L15 by auto
qed

end
```

9 Semilattices and Lattices

```
theory Lattice˙ZF imports Order˙ZF˙1a func1

begin
```

Lattices can be introduced in algebraic way as commutative idempotent ($x \cdot x = x$) semigroups or as partial orders with some additional properties. These two approaches are equivalent. In this theory we will use the order-theoretic approach.

9.1 Semilattices

We start with a relation r which is a partial order on a set L. Such situation is defined in Order˙ZF as the predicate IsPartOrder(L,r).

A partially ordered (L, r) set is a join-semilattice if each two-element subset of L has a supremum (i.e. the least upper bound).

```
definition
  IsJoinSemilattice(L,r) ≡
    r⊆L×L ∧ IsPartOrder(L,r) ∧ (∀x∈L. ∀y∈L. HasAsupremum(r,–x,y″))
```

A partially ordered (L, r) set is a meet-semilattice if each two-element subset of L has an infimum (i.e. the greatest lower bound).

```
definition
  IsMeetSemilattice(L,r) ≡
    r⊆L×L ∧ IsPartOrder(L,r) ∧ (∀x∈L. ∀y∈L. HasAnInfimum(r,–x,y″))
```

A partially ordered (L, r) set is a lattice if it is both join and meet-semilattice, i.e. if every two element set has a supremum (least upper bound) and infimum (greatest lower bound).

```
definition
  IsAlattice (infixl –is a lattice on″ 90) where
  r –is a lattice on″ L ≡ IsJoinSemilattice(L,r) ∧ IsMeetSemilattice(L,r)
```

Join is a binary operation whose value on a pair $\langle x, y\rangle$ is defined as the supremum of the set $\{x, y\}$.

```
definition
  Join(L,r) ≡ –⟨p,Supremum(r,–fst(p),snd(p)″)⟩ . p ∈ L×L″
```

Meet is a binary operation whose value on a pair $\langle x, y\rangle$ is defined as the infimum of the set $\{x, y\}$.

```
definition
  Meet(L,r) ≡ –⟨p,Infimum(r,–fst(p),snd(p)″)⟩ . p ∈ L×L″
```

In a join-semilattice join is indeed a binary operation.

```
lemma join˙is˙binop: assumes IsJoinSemilattice(L,r)
  shows Join(L,r) :  L×L → L
proof -
  from assms have ∀ p ∈ L×L. Supremum(r,–fst(p),snd(p)″) ∈ L
   unfolding IsJoinSemilattice˙def IsPartOrder˙def HasAsupremum˙def using sup˙in˙space

    by auto
  then show thesis unfolding Join˙def using ZF˙fun˙from˙total by simp
qed
```

The value of Join(L,r) on a pair $\langle x, y\rangle$ is the supremum of the set $\{x, y\}$, hence its is greater or equal than both.

```
lemma join˙val:
  assumes IsJoinSemilattice(L,r) x∈L y∈L
  defines j ≡ Join(L,r)⟨x,y⟩
  shows j∈L j = Supremum(r,–x,y″) ⟨x,j⟩ ∈ r ⟨y,j⟩ ∈ r
proof -
  from assms(1) have Join(L,r) :  L×L → L using join˙is˙binop by simp
 with assms(2,3,4) show j = Supremum(r,–x,y″) unfolding Join˙def using ZF˙fun˙from˙tot˙val

    by auto
  from assms(2,3,4) ‹Join(L,r) : L×L → L›  show j∈L using apply˙funtype by simp
  from assms(1,2,3) have r ⊆ L×L antisym(r) HasAminimum(r,⋂z∈–x,y″. r–z″)
    unfolding IsJoinSemilattice˙def IsPartOrder˙def HasAsupremum˙def by auto
  with ‹j = Supremum(r,–x,y″)› show ⟨x,j⟩ ∈ r and ⟨y,j⟩ ∈ r
    using sup˙in˙space(2) by auto
qed
```

In a meet-semilattice meet is indeed a binary operation.

```
lemma meet˙is˙binop: assumes IsMeetSemilattice(L,r)
  shows Meet(L,r) :  L×L → L
proof -
  from assms have ∀ p ∈ L×L. Infimum(r,–fst(p),snd(p)˝) ∈ L
   unfolding IsMeetSemilattice˙def IsPartOrder˙def HasAnInfimum˙def using inf˙in˙space

    by auto
  then show thesis unfolding Meet˙def using ZF˙fun˙from˙total by simp
qed
```

The value of Meet(L,r) on a pair $\langle x, y\rangle$ is the infimum of the set $\{x, y\}$, hence is less or equal than both.

```
lemma meet˙val:
  assumes IsMeetSemilattice(L,r) x∈L y∈L
  defines m ≡ Meet(L,r)⟨x,y⟩
  shows m∈L m = Infimum(r,–x,y˝) ⟨m,x⟩ ∈ r ⟨m,y⟩ ∈ r
proof -
  from assms(1) have Meet(L,r) : L×L → L using meet˙is˙binop by simp
 with assms(2,3,4) show  m = Infimum(r,–x,y˝) unfolding Meet˙def using ZF˙fun˙from˙tot˙val

    by auto
 from assms(2,3,4) ‹Meet(L,r) : L×L → L›  show m∈L using apply˙funtype by simp
  from assms(1,2,3) have r ⊆ L×L antisym(r) HasAmaximum(r,⋂z∈–x,y˝. r-–z˝)
    unfolding IsMeetSemilattice˙def IsPartOrder˙def HasAnInfimum˙def by auto
  with ‹m = Infimum(r,–x,y˝)› show ⟨m,x⟩ ∈ r and ⟨m,y⟩ ∈ r
    using inf˙in˙space(2) by auto
qed
```

The next locale defines a a notation for join-semilattice. We will use the ⊔ symbol rather than more common ∨ to avoid confusion with logical "or".

```
locale join˙semilatt =
  fixes L
  fixes r
  assumes joinLatt: IsJoinSemilattice(L,r)
  fixes join (infixl ⊔ 71)
  defines join˙def [simp]: x ⊔ y ≡ Join(L,r)⟨x,y⟩
  fixes sup (sup ˙ )
  defines sup˙def [simp]: sup A  ≡ Supremum(r,A)
```

Join of the elements of the lattice is in the lattice.

```
lemma (in join˙semilatt) join˙props: assumes x∈L y∈L
  shows x⊔y ∈ L and x⊔y = sup –x,y˝
proof -
  from joinLatt assms have Join(L,r)⟨x,y⟩ ∈ L using join˙is˙binop apply˙funtype
    by blast
  thus x⊔y ∈ L by simp
  from joinLatt assms have Join(L,r)⟨x,y⟩ = Supremum(r,–x,y˝) using join˙val(2)
    by simp
```

```
  thus x⊔y = sup –x,y″ by simp
qed
```

Join is associative.

```
lemma (in join˙semilatt) join˙assoc: assumes x∈L y∈L z∈L
  shows x⊔(y⊔z) = x⊔y⊔z
proof -
  from joinLatt assms(2,3) have x⊔(y⊔z) = x⊔(sup –y,z″) using join˙val(2) by simp
  also from assms joinLatt have ... = sup –sup –x″, sup –y,z″″
    unfolding IsJoinSemilattice˙def IsPartOrder˙def using join˙props sup˙inf˙singl(2)

    by auto
  also have ... = sup –x,y,z″
  proof -
    let 𝒯 = —x″,–y,z″″
    from joinLatt have r ⊆ L×L antisym(r) trans(r)
      unfolding IsJoinSemilattice˙def IsPartOrder˙def by auto
    moreover from joinLatt assms have ∀T∈𝒯. HasAsupremum(r,T)
      unfolding IsJoinSemilattice˙def IsPartOrder˙def using sup˙inf˙singl(1) by blast
    moreover from joinLatt assms have HasAsupremum(r,–Supremum(r,T).T∈𝒯″)
      unfolding IsJoinSemilattice˙def IsPartOrder˙def  HasAsupremum˙def
      using sup˙in˙space(1) sup˙inf˙singl(2) by auto
     ultimately have Supremum(r,–Supremum(r,T).T∈𝒯″) = Supremum(r,⋃𝒯) by
(rule sup˙sup)
   moreover have –Supremum(r,T).T∈𝒯″ = –sup –x″, sup –y,z″″ and ⋃𝒯 = –x,y,z″
      by auto
    ultimately show (sup –sup –x″, sup –y,z″″) =  sup –x,y,z″ by simp
  qed
  also have ... = sup –sup –x,y″, sup –z″″
  proof -
    let 𝒯 = —x,y″,–z″″
    from joinLatt have r ⊆ L×L antisym(r) trans(r)
      unfolding IsJoinSemilattice˙def IsPartOrder˙def by auto
    moreover from joinLatt assms have ∀T∈𝒯. HasAsupremum(r,T)
      unfolding IsJoinSemilattice˙def IsPartOrder˙def using sup˙inf˙singl(1) by blast
    moreover from joinLatt assms have HasAsupremum(r,–Supremum(r,T).T∈𝒯″)
      unfolding IsJoinSemilattice˙def IsPartOrder˙def  HasAsupremum˙def
      using sup˙in˙space(1) sup˙inf˙singl(2) by auto
     ultimately have Supremum(r,–Supremum(r,T).T∈𝒯″) = Supremum(r,⋃𝒯) by
(rule sup˙sup)
   moreover have –Supremum(r,T).T∈𝒯″ = –sup –x,y″, sup –z″″ and ⋃𝒯 = –x,y,z″
      by auto
    ultimately show (sup –x,y,z″) = sup –sup –x,y″, sup –z″″ by auto
  qed
  also from assms joinLatt have ... = sup –sup –x,y″, z″
    unfolding IsJoinSemilattice˙def IsPartOrder˙def using join˙props sup˙inf˙singl(2)

    by auto
  also from assms joinLatt have ... = (sup –x,y″) ⊔ z
```

```
    unfolding IsJoinSemilattice˙def IsPartOrder˙def using join˙props by auto
  also from joinLatt assms(1,2) have ... = x⊔y⊔z using join˙val(2) by simp
  finally show x⊔(y⊔z) = x⊔y⊔z by simp
qed
```

Join is idempotent.

```
lemma (in join˙semilatt) join˙idempotent: assumes x∈L shows x⊔x = x
 using joinLatt assms join˙val(2) IsJoinSemilattice˙def IsPartOrder˙def sup˙inf˙singl(2)
  by auto
```

The meet˙semilatt locale is the dual of the join-semilattice locale defined above. We will use the ⊓ symbol to denote join, giving it ab bit higher precedence.

```
locale meet˙semilatt =
  fixes L
  fixes r
  assumes meetLatt: IsMeetSemilattice(L,r)
  fixes join (infixl ⊓ 72)
  defines join˙def [simp]: x ⊓ y ≡ Meet(L,r)⟨x,y⟩
  fixes sup (inf ˙ )
  defines sup˙def [simp]: inf A  ≡ Infimum(r,A)
```

Meet of the elements of the lattice is in the lattice.

```
lemma (in meet˙semilatt) meet˙props: assumes x∈L y∈L
  shows x⊓y ∈ L and x⊓y = inf –x,y″
proof -
  from meetLatt assms have Meet(L,r)⟨x,y⟩ ∈ L using meet˙is˙binop apply˙funtype
    by blast
  thus x⊓y ∈ L by simp
  from meetLatt assms have Meet(L,r)⟨x,y⟩ = Infimum(r,–x,y″) using meet˙val(2)
by blast
  thus x⊓y = inf –x,y″ by simp
qed
```

Meet is associative.

```
lemma (in meet˙semilatt) meet˙assoc: assumes x∈L y∈L z∈L
  shows x⊓(y⊓z) = x⊓y⊓z
proof -
  from meetLatt assms(2,3) have x⊓(y⊓z) = x⊓(inf –y,z″) using meet˙val by simp
  also from assms meetLatt have ... = inf –inf –x″, inf –y,z″″
   unfolding IsMeetSemilattice˙def IsPartOrder˙def using meet˙props sup˙inf˙singl(4)

    by auto
  also have ... = inf –x,y,z″
  proof -
    let 𝒯 = —x″,–y,z″″
    from meetLatt have r ⊆ L×L antisym(r) trans(r)
      unfolding IsMeetSemilattice˙def IsPartOrder˙def by auto
```

```
    moreover from meetLatt assms have ∀T∈𝒯. HasAnInfimum(r,T)
      unfolding IsMeetSemilattice˙def IsPartOrder˙def using sup˙inf˙singl(3) by blast
    moreover from meetLatt assms have HasAnInfimum(r,–Infimum(r,T).T∈𝒯″)
      unfolding IsMeetSemilattice˙def IsPartOrder˙def  HasAnInfimum˙def
      using inf˙in˙space(1) sup˙inf˙singl(4) by auto
     ultimately have Infimum(r,–Infimum(r,T).T∈𝒯″) = Infimum(r,⋃𝒯) by (rule
inf˙inf)
    moreover have –Infimum(r,T).T∈𝒯″ = –inf –x″, inf –y,z″″ and ⋃𝒯 = –x,y,z″
      by auto
    ultimately show (inf –inf –x″, inf –y,z″″) =  inf –x,y,z″ by simp
  qed
  also have ... = inf –inf –x,y″, inf –z″″
  proof -
    let 𝒯 = —x,y″,–z″″
    from meetLatt have r ⊆ L×L antisym(r) trans(r)
      unfolding IsMeetSemilattice˙def IsPartOrder˙def by auto
    moreover from meetLatt assms have ∀T∈𝒯. HasAnInfimum(r,T)
      unfolding IsMeetSemilattice˙def IsPartOrder˙def using sup˙inf˙singl(3) by blast
    moreover from meetLatt assms have HasAnInfimum(r,–Infimum(r,T).T∈𝒯″)
      unfolding IsMeetSemilattice˙def IsPartOrder˙def HasAnInfimum˙def
      using inf˙in˙space(1) sup˙inf˙singl(4) by auto
     ultimately have Infimum(r,–Infimum(r,T).T∈𝒯″) = Infimum(r,⋃𝒯) by (rule
inf˙inf)
    moreover have –Infimum(r,T).T∈𝒯″ = –inf –x,y″, inf –z″″ and ⋃𝒯 = –x,y,z″
      by auto
    ultimately show (inf –x,y,z″) = inf –inf –x,y″, inf –z″″ by auto
  qed
  also from assms meetLatt have ... = inf –inf –x,y″, z″
   unfolding IsMeetSemilattice˙def IsPartOrder˙def using meet˙props sup˙inf˙singl(4)

    by auto
  also from assms meetLatt have ... = (inf –x,y″) ⊓ z
    unfolding IsMeetSemilattice˙def IsPartOrder˙def using meet˙props by auto
  also from meetLatt assms(1,2) have ... = x⊓y⊓z using meet˙val by simp
  finally show x⊓(y⊓z) = x⊓y⊓z by simp
qed
```

Meet is idempotent.

```
lemma (in meet˙semilatt) meet˙idempotent: assumes x∈L shows x⊓x = x
 using meetLatt assms meet˙val IsMeetSemilattice˙def IsPartOrder˙def sup˙inf˙singl(4)
  by auto

end
```

10 Order on natural numbers

```
theory NatOrder˙ZF imports Nat˙ZF˙IML Order˙ZF

begin
```

This theory proves that $\leq$ is a linear order on $\mathbb{N}$. $\leq$ is defined in Isabelle's Nat theory, and linear order is defined in Order˙ZF theory. Contributed by Seo Sanghyeon.

10.1 Order on natural numbers

This is the only section in this theory.

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

```
lemma NatOrder˙ZF˙1˙L1:
  assumes a∈nat and b∈nat
  shows a ≤ b ∨ b ≤ a
proof -
  from assms have I: Ord(a) ∧ Ord(b)
    using nat˙into˙Ord by auto
  then have a ∈ b ∨ a = b ∨ b ∈ a
    using Ord˙linear by simp
  with I have a ¡ b ∨ a = b ∨ b ¡ a
    using ltI by auto
  with I show a ≤ b ∨ b ≤ a
    using le˙iff by auto
qed
```

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

```
lemma NatOrder˙ZF˙1˙L2:
  shows
  antisym(Le)
  trans(Le)
  Le –is total on″ nat
  IsLinOrder(nat,Le)
proof -
  show antisym(Le)
    using antisym˙def Le˙def le˙anti˙sym by auto
  moreover show trans(Le)
    using trans˙def Le˙def le˙trans by blast
  moreover show Le –is total on″ nat
    using IsTotal˙def Le˙def NatOrder˙ZF˙1˙L1 by simp
  ultimately show IsLinOrder(nat,Le)
    using IsLinOrder˙def by simp
qed
```

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

```
lemma natord˙lin˙on˙each˙nat:
  assumes A1: n ∈ nat shows IsLinOrder(n,Le)
```

```
proof -
  from A1 have n ⊆ nat using nat˙subset˙nat
    by simp
  then show thesis using NatOrder˙ZF˙1˙L2 ord˙linear˙subset
    by blast
qed

end
```

11 Binary operations

theory func˙ZF imports func1

begin

In this theory we consider properties of functions that are binary operations, that is they map $X \times X$ into X.

11.1 Lifting operations to a function space

It happens quite often that we have a binary operation on some set and we need a similar operation that is defined for functions on that set. For example once we know how to add real numbers we also know how to add real-valued functions: for $f, g : X \to \mathbf{R}$ we define $(f + g)(x) = f(x) + g(x)$. Note that formally the + means something different on the left hand side of this equality than on the right hand side. This section aims at formalizing this process. We will call it "lifting to a function space", if you have a suggestion for a better name, please let me know.

Since we are writing in generic set notation, the definition below is a bit complicated. Here it what it says: Given a set X and another set f (that represents a binary function on X) we are defining f lifted to function space over X as the binary function (a set of pairs) on the space $F = X \to \text{range}(f)$ such that the value of this function on pair $\langle a, b\rangle$ of functions on X is another function c on X with values defined by $c(x) = f\langle a(x), b(x)\rangle$.

```
definition
Lift2FcnSpce (infix –lifted to function space over˝ 65) where
 f –lifted to function space over˝ X ≡
  –⟨ p,–⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X˝⟩.
  p ∈ (X→range(f))×(X→range(f))˝
```

The result of the lift belongs to the function space.

```
lemma func˙ZF˙1˙L1:
  assumes A1: f : Y×Y→Y
  and A2: p ∈(X→range(f))×(X→range(f))
  shows
```

```
  −⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X″ : X→range(f)
  proof -
    have ∀x∈X. f⟨fst(p)(x),snd(p)(x)⟩ ∈ range(f)
    proof
      fix x assume x∈X
      let p = ⟨fst(p)(x),snd(p)(x)⟩
      from A2 ‹x∈X› have
 fst(p)(x) ∈ range(f)  snd(p)(x) ∈ range(f)
 using apply˙type by auto
      with A1 have p ∈ Y×Y
 using func1˙1˙L5B by blast
      with A1 have ⟨p, f(p)⟩ ∈ f
 using apply˙Pair by simp
      with A1 show
 f(p) ∈ range(f)
 using rangeI by simp
    qed
    then show thesis using ZF˙fun˙from˙total by simp
qed
```

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

```
lemma func˙ZF˙1˙L2:
  assumes A1: f : Y×Y→Y
  and A2: p ∈ (X→range(f))×(X→range(f)) and A3: x∈X
  and A4: P = −⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X″
  shows P(x) = f⟨fst(p)(x),snd(p)(x)⟩
proof -
  from A1 A2 have
    −⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X″ : X → range(f)
    using func˙ZF˙1˙L1 by simp
  with A4 have P :  X → range(f) by simp
  with  A3 A4 show P(x) = f⟨fst(p)(x),snd(p)(x)⟩
    using ZF˙fun˙from˙tot˙val by simp
qed
```

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

```
theorem func˙ZF˙1˙L3:
  assumes f : Y×Y→Y
  and F = f −lifted to function space over″ X
  shows F : (X→range(f))×(X→range(f))→(X→range(f))
  using assms Lift2FcnSpce˙def func˙ZF˙1˙L1 ZF˙fun˙from˙total
  by simp
```

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

```
theorem func˙ZF˙1˙L4:
  assumes A1: f : Y×Y→Y
  and A2: F = f −lifted to function space over″ X
  and A3: s:X→range(f) r:X→range(f)
```

```
  and A4: x∈X
  shows (F⟨s,r⟩)(x) = f⟨s(x),r(x)⟩
proof -
  let p = ⟨s,r⟩
  let P = –⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X″
  from A1 A3 A4 have
    f : Y×Y→Y  p ∈ (X→range(f))×(X→range(f))
    x∈X  P = –⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X″
    by auto
  then have P(x) = f⟨fst(p)(x),snd(p)(x)⟩
    by (rule func˙ZF˙1˙L2)
  hence P(x) = f⟨s(x),r(x)⟩ by auto
  moreover have P = F⟨s,r⟩
  proof -
    from A1 A2 have F : (X→range(f))×(X→range(f))→(X→range(f))
      using func˙ZF˙1˙L3 by simp
    moreover from A3 have p ∈ (X→range(f))×(X→range(f))
      by auto
    moreover from A2 have
      F = –⟨p,–⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X″⟩.
      p ∈ (X→range(f))×(X→range(f))″
      using Lift2FcnSpce˙def by simp
    ultimately show thesis using ZF˙fun˙from˙tot˙val
      by simp
  qed
  ultimately show (F⟨s,r⟩)(x) = f⟨s(x),r(x)⟩ by auto
qed
```

11.2 Associative and commutative operations

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

Typically we say that a binary operation "$\cdot$" on a set G is "associative" if $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ for all $x, y, z \in G$. Our actual definition below does not use the multiplicative notation so that we can apply it equally to the additive notation + or whatever infix symbol we may want to use. Instead, we use the generic set theory notation and write $P\langle x, y\rangle$ to denote the value of the operation P on a pair $\langle x, y\rangle \in G \times G$.

```
definition
  IsAssociative (infix –is associative on″ 65) where
  P –is associative on″ G ≡ P : G×G→G ∧
  (∀ x ∈ G. ∀ y ∈ G. ∀ z ∈ G.
  ( P(⟨P(⟨x,y⟩),z⟩) = P( ⟨x,P(⟨y,z⟩)⟩ )))
```

A binary function $f : X \times X \to Y$ is commutative if $f\langle x, y\rangle = f\langle y, x\rangle$. Note that in the definition of associativity above we talk about binary "operation" and here we say use the term binary "function". This is not set in stone,

but usually the word "operation" is used when the range is a factor of the domain, while the word "function" allows the range to be a completely unrelated set.

```
definition
  IsCommutative (infix –is commutative on″ 65) where
  f –is commutative on″ G ≡ ∀x∈G. ∀y∈G. f⟨x,y⟩ = f⟨y,x⟩
```

The lift of a commutative function is commutative.

```
lemma func˙ZF˙2˙L1:
  assumes A1: f : G×G→G
  and A2: F = f –lifted to function space over″ X
  and A3: s : X→range(f) r : X→range(f)
  and A4: f –is commutative on″ G
  shows F⟨s,r⟩ = F⟨r,s⟩
proof -
  from A1 A2 have
    F : (X→range(f))×(X→range(f))→(X→range(f))
    using func˙ZF˙1˙L3 by simp
  with A3 have
    F⟨s,r⟩ : X→range(f) and F⟨r,s⟩ : X→range(f)
    using apply˙type by auto
  moreover have
    ∀x∈X. (F⟨s,r⟩)(x) = (F⟨r,s⟩)(x)
  proof
    fix x assume x∈X
    from A1 have range(f)⊆G
      using func1˙1˙L5B by simp
    with A3 ‹x∈X› have s(x) ∈ G and r(x) ∈ G
      using apply˙type by auto
    with A1 A2 A3 A4 ‹x∈X› show
      (F⟨s,r⟩)(x) = (F⟨r,s⟩)(x)
      using func˙ZF˙1˙L4 IsCommutative˙def by simp
  qed
  ultimately show thesis using fun˙extension˙iff
    by simp
qed
```

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

```
lemma func˙ZF˙2˙L2:
  assumes f : G×G→G
  and f –is commutative on″ G
  and F = f –lifted to function space over″ X
  shows F –is commutative on″ (X→range(f))
  using assms IsCommutative˙def func˙ZF˙2˙L1 by simp
```

The lift of an associative function is associative.

```
lemma func˙ZF˙2˙L3:
  assumes A2: F = f –lifted to function space over″ X
```

```
  and A3: s : X→range(f) r : X→range(f) q : X→range(f)
  and A4: f –is associative on″ G
  shows F⟨F⟨s,r⟩,q⟩ = F⟨s,F⟨r,q⟩⟩
proof -
  from A4 A2 have
    F : (X→range(f))×(X→range(f))→(X→range(f))
    using IsAssociative˙def func˙ZF˙1˙L3 by auto
  with A3 have I:
    F⟨s,r⟩ : X→range(f)
    F⟨r,q⟩ : X→range(f)
    F⟨F⟨s,r⟩,q⟩ : X→range(f)
    F⟨s,F⟨r,q⟩⟩: X→range(f)
    using apply˙type by auto
  moreover have
    ∀x∈X. (F⟨F⟨s,r⟩,q⟩)(x) = (F⟨s,F⟨r,q⟩⟩)(x)
  proof
    fix x assume x∈X
    from A4 have f:G×G→G
      using IsAssociative˙def by simp
    then have range(f)⊆G
      using func1˙1˙L5B by simp
    with A3 ‹x∈X› have
      s(x) ∈ G r(x) ∈ G q(x) ∈ G
      using apply˙type by auto
    with A2 I A3 A4 ‹x∈X› ‹f:G×G→G› show
      (F⟨F⟨s,r⟩,q⟩)(x) = (F⟨s,F⟨r,q⟩⟩)(x)
      using func˙ZF˙1˙L4 IsAssociative˙def by simp
  qed
  ultimately show thesis using fun˙extension˙iff
    by simp
qed
```

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

```
lemma func˙ZF˙2˙L4:
  assumes A1: f –is associative on″ G
  and A2: F = f –lifted to function space over″ X
  shows F –is associative on″ (X→range(f))
proof -
  from A1 A2 have
    F : (X→range(f))×(X→range(f))→(X→range(f))
    using IsAssociative˙def func˙ZF˙1˙L3 by auto
  moreover from A1 A2 have
    ∀s ∈ X→range(f). ∀ r ∈ X→range(f). ∀q ∈ X→range(f).
    F⟨F⟨s,r⟩,q⟩ = F⟨s,F⟨r,q⟩⟩
    using func˙ZF˙2˙L3 by simp
  ultimately show thesis using IsAssociative˙def
    by simp
qed
```

11.3 Restricting operations

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

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

```
lemma func˙ZF˙4˙L1:
  assumes A1: f:X×X→Y and A2: A⊆X
  and A3: f –is commutative on″ X
  shows restrict(f,A×A) –is commutative on″ A
proof -
  { fix x y assume x∈A and y∈A
    with A2 have x∈X and y∈X by auto
    with A3 ‹x∈A› ‹y∈A› have
      restrict(f,A×A)⟨x,y⟩ = restrict(f,A×A)⟨y,x⟩
      using IsCommutative˙def restrict˙if by simp }
  then show thesis using IsCommutative˙def by simp
qed
```

Next we define what it means that a set is closed with respect to an operation.

```
definition
  IsOpClosed (infix –is closed under″ 65) where
  A –is closed under″ f ≡ ∀x∈A. ∀y∈A. f⟨x,y⟩ ∈ A
```

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

```
lemma func˙ZF˙4˙L2:assumes A1: f –is associative on″ X
  and A2: A⊆X and A3: A –is closed under″ f
  and A4: x∈A y∈A z∈A
  and A5: g = restrict(f,A×A)
  shows g⟨g⟨x,y⟩,z⟩ = g⟨x,g⟨y,z⟩⟩
proof -
  from A4 A2 have I: x∈X y∈X z∈X
    by auto
  from A3 A4 A5 have
    g⟨g⟨x,y⟩,z⟩ = f⟨f⟨x,y⟩,z⟩
    g⟨x,g⟨y,z⟩⟩ = f⟨x,f⟨y,z⟩⟩
    using IsOpClosed˙def restrict˙if by auto
  moreover from A1 I have
    f⟨f⟨x,y⟩,z⟩ = f⟨x,f⟨y,z⟩⟩
    using IsAssociative˙def by simp
  ultimately show thesis by simp
qed
```

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

```
lemma func˙ZF˙4˙L3: assumes A1: f –is associative on″ X
```

```
  and A2: A⊆X and A3: A –is closed under″ f
  shows restrict(f,A×A) –is associative on″ A
proof -
  let g = restrict(f,A×A)
  from A1 have f:X×X→X
    using IsAssociative˙def by simp
  moreover from A2 have A×A ⊆ X×X by auto
  moreover from A3 have ∀p ∈ A×A. g(p) ∈ A
    using IsOpClosed˙def restrict˙if by auto
  ultimately have g : A×A→A
    using func1˙2˙L4 by simp
  moreover from  A1 A2 A3 have
    ∀ x ∈ A. ∀ y ∈ A. ∀ z ∈ A.
    g⟨g⟨x,y⟩,z⟩ = g⟨ x,g⟨y,z⟩⟩
    using func˙ZF˙4˙L2 by simp
  ultimately show thesis
    using IsAssociative˙def by simp
qed
```

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

```
lemma func˙ZF˙4˙L4: assumes A –is closed under″ f
  and A⊆B and x∈A  y∈A and g = restrict(f,B×B)
  shows g⟨x,y⟩ ∈ A
  using assms IsOpClosed˙def restrict by auto
```

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

```
lemma func˙ZF˙4˙L5:
  assumes A1: A –is closed under″ f
  and A2: A⊆B
  shows A –is closed under″ restrict(f,B×B)
proof -
  let g = restrict(f,B×B)
  from A1 A2 have ∀x∈A. ∀y∈A. g⟨x,y⟩ ∈ A
    using func˙ZF˙4˙L4 by simp
  then show thesis using IsOpClosed˙def by simp
qed
```

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

```
lemma func˙ZF˙4˙L6:
  assumes A –is closed under″ f
  and B –is closed under″ f
  and x ∈ A∩B y∈ A∩B
  shows f⟨x,y⟩ ∈ A∩B using assms IsOpClosed˙def by auto
```

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

```
lemma func˙ZF˙4˙L7:
  assumes A –is closed under″ f
  B –is closed under″ f
  shows A∩B –is closed under″ f
  using assms IsOpClosed˙def by simp
```

11.4 Compositions

For any set X we can consider a binary operation on the set of functions $f : X \to X$ defined by $C(f, g) = f \circ g$. Composition of functions (or relations) is defined in the standard Isabelle distribution as a higher order function and denoted with the letter O. In this section we consider the corresponding two-argument ZF-function (binary operation), that is a subset of $((X \to X) \times (X \to X)) \times (X \to X)$.

We define the notion of composition on the set X as the binary operation on the function space $X \to X$ that takes two functions and creates the their composition.

```
definition
  Composition(X) ≡
  –⟨p,fst(p) O snd(p)⟩. p ∈ (X→X)×(X→X)″
```

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

```
lemma func˙ZF˙5˙L1: shows Composition(X) : (X→X)×(X→X)→(X→X)
  using comp˙fun Composition˙def ZF˙fun˙from˙total by simp
```

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

```
lemma func˙ZF˙5˙L2: assumes f:X→X and g:X→X
  shows Composition(X)⟨f,g⟩ = f O g
proof -
  from assms have
    Composition(X) : (X→X)×(X→X)→(X→X)
    ⟨f,g⟩ ∈ (X→X)×(X→X)
    Composition(X) = –⟨p,fst(p) O snd(p)⟩. p ∈ (X→X)×(X→X)″
    using  func˙ZF˙5˙L1 Composition˙def by auto
  then show Composition(X)⟨f,g⟩ = f O g
    using  ZF˙fun˙from˙tot˙val by auto
qed
```

What is the value of a composition on an argument?

```
lemma func˙ZF˙5˙L3: assumes f:X→X and g:X→X and x∈X
  shows (Composition(X)⟨f,g⟩)(x) = f(g(x))
  using assms func˙ZF˙5˙L2 comp˙fun˙apply by simp
```

The essential condition to show that composition is associative.

```
lemma func˙ZF˙5˙L4: assumes A1: f:X→X g:X→X h:X→X
  and A2: C = Composition(X)
  shows C⟨C⟨f,g⟩,h⟩ = C⟨ f,C⟨g,h⟩⟩
proof -
  from A2 have C : ((X→X)×(X→X))→(X→X)
    using func˙ZF˙5˙L1 by simp
  with A1 have I:
    C⟨f,g⟩ : X→X
    C⟨g,h⟩ : X→X
    C⟨C⟨f,g⟩,h⟩ : X→X
    C⟨ f,C⟨g,h⟩⟩ : X→X
    using apply˙funtype by auto
  moreover have
    ∀ x ∈ X. C⟨C⟨f,g⟩,h⟩(x) = C⟨f,C⟨g,h⟩⟩(x)
  proof
    fix x assume x∈X
    with A1 A2 I have
      C⟨C⟨f,g⟩,h⟩  (x) = f(g(h(x)))
      C⟨ f,C⟨g,h⟩⟩(x) = f(g(h(x)))
      using func˙ZF˙5˙L3 apply˙funtype by auto
    then show C⟨C⟨f,g⟩,h⟩(x) = C⟨ f,C⟨g,h⟩⟩(x)
      by simp
    qed
  ultimately show thesis using fun˙extension˙iff by simp
qed
```

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

```
lemma func˙ZF˙5˙L5: shows Composition(X) –is associative on˝ (X→X)
proof -
  let C = Composition(X)
  have ∀f∈X→X. ∀g∈X→X. ∀h∈X→X.
    C⟨C⟨f,g⟩,h⟩ = C⟨f,C⟨g,h⟩⟩
    using func˙ZF˙5˙L4 by simp
  then show thesis using func˙ZF˙5˙L1 IsAssociative˙def
    by simp
qed
```

11.5 Identity function

In this section we show some additional facts about the identity function defined in the standard Isabelle's Perm theory. Note there is also image˙id˙same lemma in func1 theory.

A function that maps every point to itself is the identity on its domain.

```
lemma indentity˙fun: assumes A1: f:X→Y and A2:∀x∈X. f(x)=x
  shows f = id(X)
```

```
proof -
  from assms have f:X→Y and id(X):X→X and ∀x∈X. f(x) = id(X)(x)
    using id˙type id˙conv by auto
  then show thesis by (rule func˙eq)
qed
```

Composing a function with identity does not change the function.

```
lemma func˙ZF˙6˙L1A: assumes A1: f : X→X
  shows Composition(X)⟨f,id(X)⟩ = f
  Composition(X)⟨id(X),f⟩ = f
proof -
  have Composition(X) : (X→X)×(X→X)→(X→X)
    using func˙ZF˙5˙L1 by simp
  with A1 have Composition(X)⟨id(X),f⟩ : X→X
    Composition(X)⟨f,id(X)⟩ : X→X
    using id˙type apply˙funtype by auto
  moreover note A1
  moreover from A1 have
    ∀x∈X. (Composition(X)⟨id(X),f⟩)(x) = f(x)
    ∀x∈X. (Composition(X)⟨f,id(X)⟩)(x) = f(x)
    using id˙type func˙ZF˙5˙L3 apply˙funtype id˙conv
    by auto
  ultimately show Composition(X)⟨id(X),f⟩ = f
    Composition(X)⟨f,id(X)⟩ = f
    using fun˙extension˙iff by auto
qed
```

An intuitively clear, but surprisingly nontrivial fact: identity is the only function from a singleton to itself.

```
lemma singleton˙fun˙id: shows (–x″ → –x″) = –id(–x″)″
proof
  show –id(–x″)″ ⊆ (–x″ → –x″)
    using id˙def by simp
  { let g = id(–x″)
    fix f assume f : –x″ → –x″
    then have f : –x″ → –x″ and g : –x″ → –x″
      using id˙def by auto
    moreover from ‹f : –x″ → –x″› have ∀x ∈ –x″. f(x) = g(x)
      using apply˙funtype id˙def by auto
    ultimately have f = g by (rule func˙eq)
  } then show  (–x″ → –x″) ⊆ –id(–x″)″ by auto
qed
```

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

```
lemma single˙bij˙id: shows bij(–x″,–x″) = –id(–x″)″
proof
  show –id(–x″)″ ⊆ bij(–x″,–x″) using id˙bij
    by simp
  { fix f assume f ∈ bij(–x″,–x″)
```

```
    then have f : –x″ → –x″ using bij˙is˙fun
      by simp
    then have f ∈ –id(–x″)″ using singleton˙fun˙id
      by simp
  } then show bij(–x″,–x″) ⊆ –id(–x″)″ by auto
qed
```

A kind of induction for the identity: if a function f is the identity on a set with a fixpoint of f removed, then it is the indentity on the whole set.

```
lemma id˙fixpoint˙rem: assumes A1: f:X→X and
  A2: p∈X and A3: f(p) = p and
  A4: restrict(f, X-–p″) = id(X-–p″)
  shows f = id(X)
proof -
  from A1 have f: X→X and id(X) : X→X
    using id˙def by auto
  moreover
  { fix x assume x∈X
    { assume x ∈ X-–p″
      then have f(x) = restrict(f, X-–p″)(x)
 using restrict by simp
      with A4 ‹x ∈ X-–p″› have f(x) = x
 using id˙def by simp }
    with A2 A3 ‹x∈X› have f(x) = x by auto
  } then have ∀x∈X. f(x) = id(X)(x)
    using id˙def by simp
  ultimately show f = id(X) by (rule func˙eq)
qed
```

11.6 Lifting to subsets

Suppose we have a binary operation $f : X \times X \to X$ written additively as $f\langle x, y\rangle = x + y$. Such operation naturally defines another binary operation on the subsets of X that satisfies $A + B = \{x + y : x \in A, y \in B\}$. This new operation which we will call "f lifted to subsets" inherits many properties of f, such as associativity, commutativity and existence of the neutral element. This notion is useful for considering interval arithmetics.

The next definition describes the notion of a binary operation lifted to subsets. It is written in a way that might be a bit unexpected, but really it is the same as the intuitive definition, but shorter. In the definition we take a pair $p \in Pow(X) \times Pow(X)$, say $p = \langle A, B\rangle$, where $A, B \subseteq X$. Then we assign this pair of sets the set $\{f\langle x, y\rangle : x \in A, y \in B\} = \{f(x') : x' \in A \times B\}$ The set on the right hand side is the same as the image of $A \times B$ under f. In the definition we don't use A and B symbols, but write fst(p) and snd(p), resp. Recall that in Isabelle/ZF fst(p) and snd(p) denote the first and second components of an ordered pair p. See the lemma lift˙subsets˙explained for a

more intuitive notation.

```
definition
  Lift2Subsets (infix –lifted to subsets of″ 65) where
  f –lifted to subsets of″ X ≡
  –⟨p, f(fst(p)×snd(p))⟩. p ∈ Pow(X)×Pow(X)″
```

The lift to subsets defines a binary operation on the subsets.

```
lemma lift˙subsets˙binop: assumes A1: f : X × X → Y
  shows (f –lifted to subsets of″ X) : Pow(X) × Pow(X) → Pow(Y)
proof -
  let F = –⟨p, f(fst(p)×snd(p))⟩. p ∈ Pow(X)×Pow(X)″
  from A1 have ∀p ∈ Pow(X) × Pow(X). f(fst(p)×snd(p)) ∈ Pow(Y)
    using func1˙1˙L6 by simp
  then have F : Pow(X) × Pow(X) → Pow(Y)
    by (rule ZF˙fun˙from˙total)
  then show thesis unfolding Lift2Subsets˙def by simp
qed
```

The definition of the lift to subsets rewritten in a more intuitive notation. We would like to write the last assertion as F⟨A,B⟩ = –f⟨x,y⟩. x ∈ A, y ∈ B″, but Isabelle/ZF does not allow such syntax.

```
lemma lift˙subsets˙explained: assumes A1: f : X×X → Y
  and A2: A ⊆ X  B ⊆ X and A3: F = f –lifted to subsets of″ X
  shows
  F⟨A,B⟩ ⊆ Y and
  F⟨A,B⟩ = f(A×B)
  F⟨A,B⟩ = –f(p). p ∈ A×B″
  F⟨A,B⟩ = –f⟨x,y⟩ . ⟨x,y⟩ ∈ A×B″
proof -
  let p = ⟨A,B⟩
  from assms have
    I: F : Pow(X) × Pow(X) → Pow(Y) and  p ∈ Pow(X) × Pow(X)
    using lift˙subsets˙binop by auto
  moreover from A3 have F = –⟨p, f(fst(p)×snd(p))⟩. p ∈ Pow(X)×Pow(X)″
    unfolding  Lift2Subsets˙def by simp
  ultimately show F⟨A,B⟩ =  f(A×B)
    using  ZF˙fun˙from˙tot˙val by auto
  also
  from A1 A2 have A×B ⊆ X×X by auto
  with A1 have f(A×B) = –f(p). p ∈ A×B″
    by (rule func˙imagedef)
  finally show  F⟨A,B⟩ = –f(p) . p ∈ A×B″ by simp
  also
  have ∀x∈A. ∀y ∈ B. f⟨x,y⟩ = f⟨x,y⟩ by simp
  then have –f(p). p ∈ A×B″ = –f⟨x,y⟩.  ⟨x,y⟩ ∈ A×B″
    by (rule ZF1˙1˙L4A)
  finally show F⟨A,B⟩ = –f⟨x,y⟩ . ⟨x,y⟩ ∈ A×B″
    by simp
```

```
  from A2 I show F⟨A,B⟩ ⊆ Y using apply˙funtype by blast
qed
```

A sufficient condition for a point to belong to a result of lifting to subsets.

```
lemma lift˙subset˙suff:  assumes A1: f : X × X → Y and
  A2: A ⊆ X  B ⊆ X and A3: x∈A y∈B and
  A4: F = f –lifted to subsets of″ X
  shows f⟨x,y⟩ ∈ F⟨A,B⟩
proof -
  from A3 have f⟨x,y⟩ ∈ –f(p) . p ∈ A×B″ by auto
  moreover from A1 A2 A4 have –f(p). p ∈ A×B″ = F⟨A,B⟩
    using lift˙subsets˙explained by simp
  ultimately show f⟨x,y⟩ ∈ F⟨A,B⟩ by simp
qed
```

A kind of converse of lift˙subset˙apply, providing a necessary condition for a point to be in the result of lifting to subsets.

```
lemma lift˙subset˙nec: assumes A1: f : X × X → Y and
  A2: A ⊆ X  B ⊆ X and
  A3: F = f –lifted to subsets of″ X and
  A4: z ∈ F⟨A,B⟩
  shows ∃x y. x∈A ∧ y∈B ∧ z = f⟨x,y⟩
proof -
  from A1 A2 A3 have F⟨A,B⟩ = –f(p). p ∈ A×B″
    using lift˙subsets˙explained by simp
  with A4 show thesis by auto
qed
```

Lifting to subsets inherits commutativity.

```
lemma lift˙subset˙comm: assumes A1: f : X × X → Y and
  A2: f –is commutative on″ X and
  A3: F = f –lifted to subsets of″ X
  shows F –is commutative on″ Pow(X)
proof -
  have ∀A ∈ Pow(X). ∀B ∈ Pow(X). F⟨A,B⟩ = F⟨B,A⟩
  proof -
    { fix A assume A ∈ Pow(X)
      fix B assume B ∈ Pow(X)
      have  F⟨A,B⟩ = F⟨B,A⟩
      proof -
 have ∀z ∈  F⟨A,B⟩. z ∈  F⟨B,A⟩
 proof
   fix z assume I: z ∈ F⟨A,B⟩
   with A1 A3 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› have
     ∃x y. x∈A ∧ y∈B ∧ z = f⟨x,y⟩
     using lift˙subset˙nec by simp
   then obtain x y where x∈A and y∈B and z = f⟨x,y⟩
     by auto
   with A2 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› have z = f⟨y,x⟩
```

```
    using IsCommutative˙def by auto
  with A1 A3 I ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› ‹x∈A› ‹y∈B›
  show z ∈ F⟨B,A⟩ using lift˙subset˙suff by simp
qed
moreover have ∀z ∈  F⟨B,A⟩. z ∈  F⟨A,B⟩
proof
  fix z assume I: z ∈ F⟨B,A⟩
  with A1 A3 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› have
    ∃x y. x∈B ∧ y∈A ∧ z = f⟨x,y⟩
    using lift˙subset˙nec by simp
  then obtain x y where x∈B and y∈A and z = f⟨x,y⟩
    by auto
  with A2 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› have z = f⟨y,x⟩
    using IsCommutative˙def by auto
  with A1 A3 I ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› ‹x∈B› ‹y∈A›
  show z ∈ F⟨A,B⟩ using lift˙subset˙suff by simp
qed
ultimately show F⟨A,B⟩ = F⟨B,A⟩ by auto
     qed
   } thus thesis by auto
 qed
 then show F –is commutative on″ Pow(X)
   unfolding IsCommutative˙def by auto
qed
```

Lifting to subsets inherits associativity. To show that $F\langle\langle A, B\rangle C\rangle = F\langle A, F\langle B, C\rangle\rangle$ we prove two inclusions and the proof of the second inclusion is very similar to the proof of the first one.

```
lemma lift˙subset˙assoc:  assumes
  A1: f –is associative on″ X and A2: F = f –lifted to subsets of″ X
  shows F –is associative on″ Pow(X)
proof -
  from A1 have f : X×X → X unfolding IsAssociative˙def by simp
  with A2 have F : Pow(X)×Pow(X) → Pow(X)
    using lift˙subsets˙binop by simp
  moreover have ∀A ∈ Pow(X).∀B ∈ Pow(X). ∀C ∈ Pow(X).
    F⟨F⟨A,B⟩,C⟩ = F⟨A,F⟨B,C⟩⟩
  proof -
    { fix A B C
      assume A ∈ Pow(X)  B ∈ Pow(X)  C ∈ Pow(X)
      have F⟨F⟨A,B⟩,C⟩ ⊆ F⟨A,F⟨B,C⟩⟩
      proof
 fix z assume I: z ∈ F⟨F⟨A,B⟩,C⟩
 from ‹f:X×X → X› A2 ‹A ∈ Pow(X)›  ‹B ∈ Pow(X)›
 have F⟨A,B⟩ ∈ Pow(X)
   using lift˙subsets˙binop apply˙funtype by blast
 with ‹f:X×X → X› A2 ‹C ∈ Pow(X)› I have
   ∃x y. x ∈ F⟨A,B⟩ ∧ y ∈ C ∧ z = f⟨x,y⟩
   using lift˙subset˙nec by simp
```

```
then obtain x y where
  II: x ∈ F⟨A,B⟩ and y ∈ C and III: z = f⟨x,y⟩
  by auto
from ‹f:X×X → X› A2 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› II have
  ∃ s t. s ∈ A ∧ t ∈ B ∧ x = f⟨s,t⟩
  using lift˙subset˙nec by auto
then obtain s t where s∈A and t∈B and x = f⟨s,t⟩
  by auto
with A1 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› ‹C ∈ Pow(X)› III
  ‹s∈A› ‹t∈B› ‹y∈C› have IV: z = f⟨s, f⟨t,y⟩⟩
  using IsAssociative˙def by blast
from ‹f:X×X → X› A2 ‹B ∈ Pow(X)› ‹C ∈ Pow(X)› ‹t∈B› ‹y∈C›
have f⟨t,y⟩ ∈ F⟨B,C⟩ using lift˙subset˙suff by simp
moreover from ‹f:X×X → X› A2 ‹B ∈ Pow(X)› ‹C ∈ Pow(X)›
have F⟨B,C⟩ ⊆ X using lift˙subsets˙binop apply˙funtype
  by blast
moreover note ‹f:X×X → X› A2 ‹A ∈ Pow(X)› ‹s∈A› IV
ultimately show z ∈ F⟨A,F⟨B,C⟩⟩
  using lift˙subset˙suff by simp
    qed
    moreover have F⟨A,F⟨B,C⟩⟩ ⊆ F⟨F⟨A,B⟩,C⟩
    proof
fix z assume I: z ∈ F⟨A,F⟨B,C⟩⟩
from ‹f:X×X → X› A2 ‹B ∈ Pow(X)› ‹C ∈ Pow(X)›
have F⟨B,C⟩ ∈ Pow(X)
  using lift˙subsets˙binop apply˙funtype by blast
with ‹f:X×X → X› A2 ‹A ∈ Pow(X)› I have
  ∃x y. x ∈ A ∧ y ∈ F⟨B,C⟩ ∧ z = f⟨x,y⟩
  using lift˙subset˙nec by simp
then obtain x y where
  x ∈ A and II: y ∈ F⟨B,C⟩ and III: z = f⟨x,y⟩
  by auto
from ‹f:X×X → X› A2 ‹B ∈ Pow(X)› ‹C ∈ Pow(X)› II have
  ∃ s t. s ∈ B ∧ t ∈ C ∧ y = f⟨s,t⟩
  using lift˙subset˙nec by auto
then obtain s t where s∈B and t∈C and y = f⟨s,t⟩
  by auto
with III have z = f⟨x,f⟨s,t⟩⟩ by simp
moreover from A1 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› ‹C ∈ Pow(X)›
  ‹x∈A› ‹s∈B› ‹t∈C› have f⟨f⟨x,s⟩,t⟩ = f⟨x,f⟨s,t⟩⟩
  using IsAssociative˙def by blast
ultimately have IV: z = f⟨f⟨x,s⟩,t⟩ by simp
from ‹f:X×X → X› A2 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› ‹x∈A› ‹s∈B›
have f⟨x,s⟩ ∈ F⟨A,B⟩ using lift˙subset˙suff by simp
moreover from ‹f:X×X → X› A2 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)›
have F⟨A,B⟩ ⊆ X using lift˙subsets˙binop apply˙funtype
  by blast
moreover note ‹f:X×X → X› A2 ‹C ∈ Pow(X)› ‹t∈C› IV
ultimately show z ∈ F⟨F⟨A,B⟩,C⟩
```

```
    using lift˙subset˙suff by simp
      qed
      ultimately have F⟨F⟨A,B⟩,C⟩ = F⟨A,F⟨B,C⟩⟩ by auto
    } thus thesis by auto
  qed
  ultimately show thesis unfolding IsAssociative˙def
    by auto
qed
```

11.7 Distributive operations

In this section we deal with pairs of operations such that one is distributive with respect to the other, that is $a\cdot(b+c) = a\cdot b+a\cdot c$ and $(b+c)\cdot a = b\cdot a+c\cdot a$. We show that this property is preserved under restriction to a set closed with respect to both operations. In EquivClass1 theory we show that this property is preserved by projections to the quotient space if both operations are congruent with respect to the equivalence relation.

We define distributivity as a statement about three sets. The first set is the set on which the operations act. The second set is the additive operation (a ZF function) and the third is the multiplicative operation.

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

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

```
lemma func˙ZF˙7˙L1:
  assumes A1: IsDistributive(X,A,M)
  and A2: Y⊆X
  and A3: Y –is closed under″ A  Y –is closed under″ M
  and A4: A_r = restrict(A,Y×Y) M_r = restrict(M,Y×Y)
  and A5: a∈Y  b∈Y  c∈Y
  shows 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⟩ ⟩
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
    A_r⟨b,c⟩ ∈ Y  M_r⟨a,b⟩ ∈ Y  M_r⟨a,c⟩ ∈ Y
    M_r⟨b,a⟩ ∈ Y  M_r⟨c,a⟩ ∈ Y
    using restrict by auto
  with A1 A2 A4 A5 show thesis
    using restrict IsDistributive˙def by auto
qed
```

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

to both operations.

lemma func˙ZF˙7˙L2:
  assumes IsDistributive(X,A,M)
  and Y⊆X
  and Y –is closed under″ A
  Y –is closed under″ M
  and A_r = restrict(A,Y×Y) M_r = restrict(M,Y×Y)
  shows IsDistributive(Y,A_r,M_r)
proof -
  from assms 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

end

12 More on functions

theory func˙ZF˙1 imports ZF.Order Order˙ZF˙1a func˙ZF

begin

In this theory we consider some properties of functions related to order relations

12.1 Functions and order

This section deals with functions between ordered sets.

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

lemma func˙ZF˙8˙L1:
  assumes f:X→Y and A⊆X and ∀x∈A. ⟨L,f(x)⟩ ∈ r
  shows IsBoundedBelow(f(A),r)
proof -
  from assms have ∀y ∈ f(A). ⟨L,y⟩ ∈ r
    using func˙imagedef by simp
  then show IsBoundedBelow(f(A),r)
    by (rule Order˙ZF˙3˙L9)
qed

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

lemma func˙ZF˙8˙L2:

```
  assumes f:X→Y and A⊆X and ∀x∈A. ⟨f(x),U⟩ ∈ r
  shows IsBoundedAbove(f(A),r)
proof -
  from assms have ∀y ∈ f(A). ⟨y,U⟩ ∈ r
    using func˙imagedef by simp
  then show IsBoundedAbove(f(A),r)
    by (rule Order˙ZF˙3˙L10)
qed
```

Identity is an order isomorphism.

```
lemma id˙ord˙iso: shows id(X) ∈ ord˙iso(X,r,X,r)
  using id˙bij id˙def ord˙iso˙def by simp
```

Identity is the only order automorphism of a singleton.

```
lemma id˙ord˙auto˙singleton:
  shows ord˙iso(–x″,r,–x″,r) = –id(–x″)″
  using id˙ord˙iso ord˙iso˙def single˙bij˙id
  by auto
```

The image of a maximum by an order isomorphism is a maximum. Note that from the fact the r is antisymmetric and f is an order isomorphism between (A, r) and (B, R) we can not conclude that R is antisymmetric (we can only show that $R \cap (B \times B)$ is).

```
lemma max˙image˙ord˙iso:
  assumes A1: antisym(r) and A2: antisym(R) and
  A3: f ∈ ord˙iso(A,r,B,R) and
  A4: HasAmaximum(r,A)
  shows HasAmaximum(R,B) and Maximum(R,B) = f(Maximum(r,A))
proof -
  let M = Maximum(r,A)
  from A1 A4 have M ∈ A using Order˙ZF˙4˙L3 by simp
  from A3 have f:A→B using ord˙iso˙def bij˙is˙fun
    by simp
  with ‹M ∈ A› have I: f(M) ∈ B
    using apply˙funtype by simp
  { fix y assume y ∈ B
    let x = converse(f)(y)
    from A3 have converse(f) ∈ ord˙iso(B,R,A,r)
      using ord˙iso˙sym by simp
    then have converse(f): B → A
      using ord˙iso˙def bij˙is˙fun by simp
    with ‹y ∈ B› have x ∈ A
      by simp
    with A1 A3 A4 ‹x ∈ A› ‹M ∈ A› have ⟨f(x), f(M)⟩ ∈ R
      using Order˙ZF˙4˙L3 ord˙iso˙apply by simp
    with A3 ‹y ∈ B› have ⟨y, f(M)⟩ ∈ R
      using right˙inverse˙bij ord˙iso˙def by auto
  } then have II: ∀y ∈ B. ⟨y, f(M)⟩ ∈ R by simp
```

```
  with A2 I show Maximum(R,B) = f(M)
    by (rule Order˙ZF˙4˙L14)
  from I II show HasAmaximum(R,B)
    using HasAmaximum˙def by auto
qed
```

Maximum is a fixpoint of order automorphism.

```
lemma max˙auto˙fixpoint:
  assumes antisym(r) and f ∈ ord˙iso(A,r,A,r)
  and HasAmaximum(r,A)
  shows Maximum(r,A) = f(Maximum(r,A))
  using assms max˙image˙ord˙iso by blast
```

If two sets are order isomorphic and we remove x and $f(x)$, respectively, from the sets, then they are still order isomorphic.

```
lemma ord˙iso˙rem˙point:
  assumes A1: f ∈ ord˙iso(A,r,B,R) and A2: a ∈ A
  shows restrict(f,A-–a″) ∈ ord˙iso(A-–a″,r,B-–f(a)″,R)
proof -
  let f0 = restrict(f,A-–a″)
  have A-–a″ ⊆ A by auto
  with A1 have f0 ∈ ord˙iso(A-–a″,r,f(A-–a″),R)
    using ord˙iso˙restrict˙image by simp
  moreover
  from A1 have f ∈ inj(A,B)
    using ord˙iso˙def bij˙def by simp
  with A2  have f(A-–a″) = f(A) - f–a″
    using inj˙image˙dif by simp
  moreover from A1 have f(A) = B
    using ord˙iso˙def bij˙def surj˙range˙image˙domain
    by auto
  moreover
  from A1 have f: A→B
    using ord˙iso˙def bij˙is˙fun by simp
  with A2 have f–a″ = –f(a)″
    using singleton˙image by simp
  ultimately show thesis by simp
qed
```

If two sets are order isomorphic and we remove maxima from the sets, then they are still order isomorphic.

```
corollary ord˙iso˙rem˙max:
  assumes A1: antisym(r) and f ∈ ord˙iso(A,r,B,R) and
  A4: HasAmaximum(r,A) and  A5: M = Maximum(r,A)
  shows restrict(f,A-–M″) ∈ ord˙iso(A-–M″, r, B-–f(M)″,R)
  using assms Order˙ZF˙4˙L3 ord˙iso˙rem˙point by simp
```

Lemma about extending order isomorphisms by adding one point to the domain.

```
lemma ord˙iso˙extend:  assumes A1: f ∈ ord˙iso(A,r,B,R) and
  A2: M_A ∉ A M_B ∉ B and
  A3: ∀a∈A. ⟨a, M_A⟩ ∈ r  ∀b∈B. ⟨b, M_B⟩ ∈ R and
  A4: antisym(r)  antisym(R) and
  A5: ⟨M_A,M_A⟩ ∈ r ⟷ ⟨M_B,M_B⟩ ∈ R
  shows f ∪ –⟨ M_A,M_B⟩″ ∈ ord˙iso(A∪–M_A″ ,r,B∪–M_B″ ,R)
proof -
  let g = f ∪ –⟨ M_A,M_B⟩″
  from A1 A2 have
    g : A∪–M_A″ → B∪–M_B″ and
    I: ∀x∈A. g(x) = f(x) and II: g(M_A) = M_B
    using ord˙iso˙def bij˙def inj˙def func1˙1˙L11D
    by auto
  from A1 A2 have g ∈ bij(A∪–M_A″,B∪–M_B″)
    using ord˙iso˙def bij˙extend˙point by simp
  moreover have ∀x ∈ A∪–M_A″. ∀ y ∈ A∪–M_A″.
    ⟨x,y⟩ ∈ r ⟷ ⟨g(x), g(y)⟩ ∈ R
  proof -
    { fix x y
      assume x ∈ A∪–M_A″ and y ∈ A∪–M_A″
      then have x∈A ∧ y ∈ A ∨ x∈A ∧ y = M_A ∨
 x = M_A ∧ y ∈ A ∨ x = M_A ∧ y = M_A
 by auto
      moreover
      { assume x∈A ∧ y ∈ A
 with A1 I have ⟨x,y⟩ ∈ r ⟷ ⟨g(x), g(y)⟩ ∈ R
   using ord˙iso˙def by simp }
      moreover
      { assume x∈A ∧ y = M_A
 with A1 A3 I II have ⟨x,y⟩ ∈ r ⟷ ⟨g(x), g(y)⟩ ∈ R
   using ord˙iso˙def bij˙def inj˙def apply˙funtype
   by auto }
      moreover
      { assume x = M_A ∧ y ∈ A
 with A2 A3 A4 have ⟨x,y⟩ ∉ r
   using antisym˙def by auto
 moreover
 { assume A6: ⟨g(x), g(y)⟩ ∈ R
   from A1 I II ‹x = M_A ∧ y ∈ A› have
     III: g(y) ∈ B  g(x) = M_B
     using ord˙iso˙def bij˙def inj˙def apply˙funtype
     by auto
   with A3 have ⟨g(y), g(x)⟩ ∈ R by simp
   with A4 A6 have g(y) = g(x) using antisym˙def
     by auto
   with A2 III have False by simp
 } hence ⟨g(x), g(y)⟩ ∉ R by auto
 ultimately have ⟨x,y⟩ ∈ r ⟷ ⟨g(x), g(y)⟩ ∈ R
 by simp }
```

```
    moreover
    { assume x = M_A ∧ y = M_A
 with A5 II have ⟨x,y⟩ ∈ r ⟷ ⟨g(x), g(y)⟩ ∈ R
   by simp }
    ultimately have ⟨x,y⟩ ∈ r ⟷ ⟨g(x), g(y)⟩ ∈ R
 by auto
   } thus thesis by auto
  qed
  ultimately show thesis using ord˙iso˙def
    by simp
qed
```

A kind of converse to ord˙iso˙rem˙max: if two linearly ordered sets sets are order isomorphic after removing the maxima, then they are order isomorphic.

```
lemma rem˙max˙ord˙iso:
  assumes A1: IsLinOrder(X,r)  IsLinOrder(Y,R) and
  A2: HasAmaximum(r,X)  HasAmaximum(R,Y)
  ord˙iso(X - –Maximum(r,X)″,r,Y - –Maximum(R,Y)″,R) ≠ 0
  shows ord˙iso(X,r,Y,R) ≠ 0
proof -
  let M_A = Maximum(r,X)
  let A = X - –M_A″
  let M_B = Maximum(R,Y)
  let B = Y - –M_B″
  from A2 obtain f where f ∈ ord˙iso(A,r,B,R)
    by auto
  moreover have M_A ∉ A and M_B ∉ B
    by auto
  moreover from A1 A2 have
    ∀a∈A. ⟨a,M_A⟩ ∈ r and ∀b∈B. ⟨b,M_B⟩ ∈ R
    using IsLinOrder˙def Order˙ZF˙4˙L3 by auto
  moreover from A1 have antisym(r) and antisym(R)
    using IsLinOrder˙def by auto
  moreover from A1 A2 have ⟨M_A,M_A⟩ ∈ r ⟷ ⟨M_B,M_B⟩ ∈ R
    using IsLinOrder˙def Order˙ZF˙4˙L3 IsLinOrder˙def
      total˙is˙refl refl˙def by auto
  ultimately have
    f ∪ –⟨ M_A,M_B⟩″ ∈ ord˙iso(A∪–M_A″ ,r,B∪–M_B″ ,R)
    by (rule ord˙iso˙extend)
  moreover from A1 A2 have
    A∪–M_A″ = X and B∪–M_B″ = Y
  using IsLinOrder˙def Order˙ZF˙4˙L3 by auto
  ultimately show ord˙iso(X,r,Y,R) ≠ 0
    using ord˙iso˙extend by auto
qed
```

12.2 Functions in cartesian products

In this section we consider maps arising naturally in cartesian products.

There is a natural bijection etween $X = Y \times \{y\}$ (a "slice") and Y. We will call this the SliceProjection(Y×–y″). This is really the ZF equivalent of the meta-function fst(x).

```
definition
  SliceProjection(X) ≡ –⟨p,fst(p)⟩. p ∈ X ″
```

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

```
lemma slice˙proj˙bij: shows
  SliceProjection(X×–y″): X×–y″ → X
  domain(SliceProjection(X×–y″)) = X×–y″
  ∀p∈X×–y″. SliceProjection(X×–y″)(p) = fst(p)
  SliceProjection(X×–y″) ∈ bij(X×–y″,X)
proof -
  let P = SliceProjection(X×–y″)
  have  ∀p ∈ X×–y″. fst(p) ∈ X by simp
  moreover from this have
    –⟨p,fst(p)⟩. p ∈ X×–y″ ″ : X×–y″ → X
    by (rule ZF˙fun˙from˙total)
  ultimately show
    I: P: X×–y″ → X and II: ∀p∈X×–y″. P(p) = fst(p)
    using ZF˙fun˙from˙tot˙val SliceProjection˙def by auto
  hence
    ∀a ∈ X×–y″. ∀ b ∈ X×–y″. P(a) = P(b) ⟶ a=b
    by auto
  with I have P ∈ inj(X×–y″,X) using inj˙def
    by simp
  moreover from II have ∀x∈X. ∃p∈X×–y″. P(p) = x
    by simp
  with I have P ∈ surj(X×–y″,X) using surj˙def
    by simp
  ultimately show P ∈ bij(X×–y″,X)
    using bij˙def by simp
  from I show domain(SliceProjection(X×–y″)) = X×–y″
    using func1˙1˙L1 by simp
qed
```

Given 2 functions $f : A \to B$ and $g : C \to D$, we can consider a function $h : A \times C \to B \times D$ such that $h(x, y) = \langle f(x), g(y)\rangle$

```
definition
  ProdFunction where
  ProdFunction(f,g) ≡ –⟨z,⟨f(fst(z)),g(snd(z))⟩⟩. z∈domain(f)×domain(g)″
```

For given functions $f : A \to B$ and $g : C \to D$ the function ProdFunction(f,g) maps $A \times C$ to $B \times D$.

```
lemma prodFunction:
  assumes f:A→B g:C→D
  shows ProdFunction(f,g):(A×C)→(B×D)
proof-
```

```
  from assms have ∀z ∈ domain(f)×domain(g). ⟨f(fst(z)),g(snd(z))⟩ ∈ B×D
    using func1˙1˙L1 apply˙type by auto
 with assms show thesis unfolding ProdFunction˙def using func1˙1˙L1 ZF˙fun˙from˙total
    by simp
qed
```

For given functions $f : A \to B$ and $g : C \to D$ and points $x \in A$, $y \in C$ the value of the function ProdFunction(f,g) on $\langle x, y\rangle$ is $\langle f(x), g(y)\rangle$.

```
lemma prodFunctionApp:
  assumes f:A→B g:C→D x∈A y∈C
  shows ProdFunction(f,g)⟨x,y⟩ = ⟨f(x),g(y)⟩
proof -
  let z = ⟨x,y⟩
  from assms have z ∈ A×C and ProdFunction(f,g):(A×C)→(B×D)
    using prodFunction by auto
   moreover from assms(1,2) have ProdFunction(f,g) = –⟨z,⟨f(fst(z)),g(snd(z))⟩⟩.
z∈A×C″
    unfolding ProdFunction˙def using func1˙1˙L1 by blast
  ultimately show thesis using ZF˙fun˙from˙tot˙val by auto
qed
```

Somewhat technical lemma about inverse image of a set by a ProdFunction(f,f).

```
lemma prodFunVimage: assumes x∈X f:X→Y
  shows ⟨x,t⟩ ∈ ProdFunction(f,f)-(V) ⟷ t∈X ∧ ⟨fx,ft⟩ ∈ V
proof -
  from assms(2) have T:ProdFunction(f,f)-(V) = –z ∈ X×X. ProdFunction(f,f)(z)
∈ V″
    using prodFunction func1˙1˙L15 by blast
  with assms show thesis using prodFunctionApp by auto
qed
```

12.3 Induced relations and order isomorphisms

When we have two sets X, Y, function $f : X \to Y$ and a relation R on Y we can define a relation r on X by saying that $x\ r\ y$ if and only if $f(x)\ R\ f(y)$. This is especially interesting when f is a bijection as all reasonable properties of R are inherited by r. This section treats mostly the case when R is an order relation and f is a bijection. The standard Isabelle's Order theory defines the notion of a space of order isomorphisms between two sets relative to a relation. We expand that material proving that order isomrphisms preserve interesting properties of the relation.

We call the relation created by a relation on Y and a mapping $f : X \to Y$ the InducedRelation(f,R).

```
definition
  InducedRelation(f,R) ≡
```

```
  –p ∈ domain(f)×domain(f). ⟨f(fst(p)),f(snd(p))⟩ ∈ R″
```

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

```
lemma def˙of˙ind˙relA:
  assumes ⟨x,y⟩ ∈ InducedRelation(f,R)
  shows ⟨f(x),f(y)⟩ ∈ R
  using assms InducedRelation˙def by simp
```

A reformulation of the definition of the relation induced by a function, kind of converse of def˙of˙ind˙relA.

```
lemma def˙of˙ind˙relB: assumes f:A→B and
  x∈A  y∈A and ⟨f(x),f(y)⟩ ∈ R
  shows ⟨x,y⟩ ∈ InducedRelation(f,R)
  using assms func1˙1˙L1 InducedRelation˙def by simp
```

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

```
lemma ord˙iso˙apply˙conv:
  assumes f ∈ ord˙iso(A,r,B,R) and
  ⟨f(x),f(y)⟩ ∈ R and x∈A  y∈A
  shows ⟨x,y⟩ ∈ r
  using assms ord˙iso˙def by simp
```

The next lemma tells us where the induced relation is defined

```
lemma ind˙rel˙domain:
  assumes  R ⊆ B×B and f:A→B
  shows InducedRelation(f,R) ⊆ A×A
  using assms func1˙1˙L1 InducedRelation˙def
  by auto
```

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

```
lemma bij˙is˙ord˙iso: assumes A1: f ∈ bij(A,B)
  shows f ∈ ord˙iso(A,InducedRelation(f,R),B,R)
proof -
  let r = InducedRelation(f,R)
  { fix x y assume A2: x∈A  y∈A
    have ⟨x,y⟩ ∈ r ⟷ ⟨f(x),f(y)⟩ ∈ R
    proof
      assume ⟨x,y⟩ ∈ r then show ⟨f(x),f(y)⟩ ∈ R
 using def˙of˙ind˙relA by simp
    next assume ⟨f(x),f(y)⟩ ∈ R
      with A1 A2 show ⟨x,y⟩ ∈ r
 using bij˙is˙fun def˙of˙ind˙relB by blast
    qed }
  with A1 show f ∈ ord˙iso(A,InducedRelation(f,R),B,R)
    using ord˙isoI by simp
qed
```

An order isomoprhism preserves antisymmetry.

```
lemma ord˙iso˙pres˙antsym: assumes A1: f ∈ ord˙iso(A,r,B,R) and
  A2: r ⊆ A×A and A3: antisym(R)
  shows antisym(r)
proof -
  { fix x y
    assume A4: ⟨x,y⟩ ∈ r   ⟨y,x⟩ ∈ r
    from A1 have f ∈ inj(A,B)
      using ord˙iso˙is˙bij bij˙is˙inj by simp
    moreover
    from A1 A2 A4 have
      ⟨f(x), f(y)⟩ ∈ R and ⟨f(y), f(x)⟩ ∈ R
      using ord˙iso˙apply by auto
    with A3 have f(x) = f(y) by (rule Fol1˙L4)
    moreover from A2 A4 have x∈A  y∈A by auto
    ultimately have x=y by (rule inj˙apply˙equality)
  } then have ∀x y. ⟨x,y⟩ ∈ r ∧ ⟨y,x⟩ ∈ r ⟶ x=y by auto
  then show antisym(r) using imp˙conj antisym˙def
    by simp
qed
```

Order isomoprhisms preserve transitivity.

```
lemma ord˙iso˙pres˙trans: assumes A1: f ∈ ord˙iso(A,r,B,R) and
  A2: r ⊆ A×A and A3: trans(R)
  shows trans(r)
proof -
  { fix x y z
    assume A4: ⟨x, y⟩ ∈ r   ⟨y, z⟩ ∈ r
    note A1
    moreover
    from A1 A2 A4 have
      ⟨f(x), f(y)⟩ ∈ R ∧ ⟨f(y), f(z)⟩ ∈ R
      using ord˙iso˙apply by auto
    with A3 have ⟨f(x),f(z)⟩ ∈ R by (rule Fol1˙L3)
    moreover from A2 A4 have x∈A  z∈A by auto
    ultimately have ⟨x, z⟩ ∈ r using ord˙iso˙apply˙conv
      by simp
  } then have  ∀ x y z. ⟨x, y⟩ ∈ r ∧ ⟨y, z⟩ ∈ r ⟶ ⟨x, z⟩ ∈ r
    by blast
  then show trans(r) by (rule Fol1˙L2)
qed
```

Order isomorphisms preserve totality.

```
lemma ord˙iso˙pres˙tot: assumes A1: f ∈ ord˙iso(A,r,B,R) and
  A2: r ⊆ A×A and A3: R  –is total on″ B
  shows r  –is total on″ A
proof -
  { fix x y
    assume x∈A  y∈A  ⟨x,y⟩ ∉ r
```

```
    with A1 have ⟨f(x),f(y)⟩ ∉ R using ord˙iso˙apply˙conv
      by auto
    moreover
    from A1 have f:A→B using ord˙iso˙is˙bij bij˙is˙fun
      by simp
    with A3 ‹x∈A› ‹y∈A› have
      ⟨f(x),f(y)⟩ ∈ R ∨ ⟨f(y),f(x)⟩ ∈ R
      using apply˙funtype IsTotal˙def by simp
    ultimately have ⟨f(y),f(x)⟩ ∈ R by simp
    with A1 ‹x∈A› ‹y∈A› have ⟨y,x⟩ ∈ r
      using ord˙iso˙apply˙conv by simp
  } then have ∀x∈A. ∀y∈A. ⟨x,y⟩ ∈ r ∨ ⟨y,x⟩ ∈ r
    by blast
  then show r –is total on″ A using IsTotal˙def
    by simp
qed
```

Order isomorphisms preserve linearity.

```
lemma ord˙iso˙pres˙lin: assumes f ∈ ord˙iso(A,r,B,R) and
  r ⊆ A×A and IsLinOrder(B,R)
  shows IsLinOrder(A,r)
  using assms ord˙iso˙pres˙antsym ord˙iso˙pres˙trans ord˙iso˙pres˙tot
    IsLinOrder˙def by simp
```

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

```
lemma ind˙rel˙pres˙lin:
  assumes A1: f ∈ bij(A,B) and A2: IsLinOrder(B,R)
  shows IsLinOrder(A,InducedRelation(f,R))
proof -
  let r = InducedRelation(f,R)
  from A1 have f ∈ ord˙iso(A,r,B,R) and r ⊆ A×A
    using bij˙is˙ord˙iso domain˙of˙bij InducedRelation˙def
    by auto
  with A2 show IsLinOrder(A,r) using ord˙iso˙pres˙lin
    by simp
qed
```

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

```
lemma ord˙iso˙pres˙bound˙above:
  assumes A1: f ∈ ord˙iso(A,r,B,R) and A2: r ⊆ A×A and
  A3: IsBoundedAbove(C,r)  C≠0
  shows IsBoundedAbove(f(C),R)  f(C) ≠ 0
proof -
  from A3 obtain u where I: ∀x∈C. ⟨x,u⟩ ∈ r
    using IsBoundedAbove˙def by auto
  from A1 have f:A→B using ord˙iso˙is˙bij bij˙is˙fun
```

```
    by simp
  from A2 A3 have C⊆A using Order˙ZF˙3˙L1A by blast
  from A3 obtain x where x∈C by auto
  with A2 I have u∈A by auto
  { fix y assume y ∈ f(C)
    with ‹f:A→B› ‹C⊆A› obtain x where x∈C and y = f(x)
      using func˙imagedef by auto
    with A1 I ‹C⊆A› ‹u∈A› have ⟨y,f(u)⟩ ∈ R
      using ord˙iso˙apply by auto
  } then have ∀y ∈ f(C). ⟨y,f(u)⟩ ∈ R by simp
  then show IsBoundedAbove(f(C),R) by (rule Order˙ZF˙3˙L10)
  from A3 ‹f:A→B› ‹C⊆A› show f(C) ≠ 0 using func1˙1˙L15A
    by simp
qed
```

Order isomorphisms preserve the property of having a minimum.

```
lemma ord˙iso˙pres˙has˙min:
  assumes A1: f ∈ ord˙iso(A,r,B,R) and  A2: r ⊆ A×A and
  A3: C⊆A and A4: HasAminimum(R,f(C))
  shows HasAminimum(r,C)
proof -
  from A4 obtain m where
    I: m ∈ f(C) and II: ∀y ∈ f(C). ⟨m,y⟩ ∈ R
    using HasAminimum˙def by auto
  let k = converse(f)(m)
  from A1 have f:A→B using ord˙iso˙is˙bij bij˙is˙fun
    by simp
  from A1 have f ∈ inj(A,B) using ord˙iso˙is˙bij bij˙is˙inj
    by simp
  with A3 I have k ∈ C and III: f(k) = m
    using inj˙inv˙back˙in˙set by auto
  moreover
  { fix x assume A5: x∈C
    with A3 II ‹f:A→B› ‹k ∈ C› III have
      k ∈ A   x∈A  ⟨f(k),f(x)⟩ ∈ R
      using func˙imagedef by auto
    with A1 have ⟨k,x⟩ ∈ r using ord˙iso˙apply˙conv
      by simp
  } then have ∀x∈C. ⟨k,x⟩ ∈ r by simp
  ultimately show HasAminimum(r,C) using HasAminimum˙def by auto
qed
```

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

```
lemma ord˙iso˙pres˙rel˙image:
  assumes A1: f ∈ ord˙iso(A,r,B,R) and
  A2: r ⊆ A×A  R ⊆ B×B and
  A3: a∈A
  shows f(r–a″) = R–f(a)″
```

```
proof
  from A1 have f:A→B using ord˙iso˙is˙bij bij˙is˙fun
    by simp
  moreover from A2 A3 have I: r–a″ ⊆ A by auto
  ultimately have I: f(r–a″) = –f(x). x ∈ r–a″ ″
    using func˙imagedef by simp
  { fix y assume A4: y ∈ f(r–a″)
    with I obtain x where
      x ∈ r–a″ and II: y = f(x)
      by auto
    with A1 A2 have ⟨f(a),f(x)⟩ ∈ R using ord˙iso˙apply
      by auto
    with II have y ∈  R–f(a)″ by auto
  } then show  f(r–a″) ⊆ R–f(a)″ by auto
  { fix y assume A5: y ∈ R–f(a)″
    let x = converse(f)(y)
    from A2 A5 have
      ⟨f(a),y⟩ ∈ R  f(a) ∈ B  and IV: y∈B
      by auto
    with A1 have III: ⟨converse(f)(f(a)),x⟩ ∈ r
      using ord˙iso˙converse by simp
    moreover from A1 A3 have converse(f)(f(a)) = a
      using ord˙iso˙is˙bij left˙inverse˙bij by blast
    ultimately have f(x) ∈ –f(x). x ∈  r–a″ ″
      by auto
    moreover from A1 IV have f(x) = y
      using ord˙iso˙is˙bij right˙inverse˙bij by blast
    moreover from A1 I have f(r–a″) = –f(x). x ∈  r–a″ ″
      using ord˙iso˙is˙bij bij˙is˙fun func˙imagedef by blast
    ultimately have y ∈ f(r–a″) by simp
  } then show R–f(a)″ ⊆ f(r–a″) by auto
qed
```

Order isomorphisms preserve collections of upper bounds.

```
lemma ord˙iso˙pres˙up˙bounds:
  assumes A1: f ∈ ord˙iso(A,r,B,R) and
  A2: r ⊆ A×A  R ⊆ B×B and
  A3: C⊆A
  shows –f(r–a″). a∈C″ = –R–b″. b ∈ f(C)″
proof
  from A1 have f:A→B
      using ord˙iso˙is˙bij bij˙is˙fun by simp
  { fix Y assume Y ∈ –f(r–a″). a∈C″
    then obtain a where a∈C and I: Y = f(r–a″)
      by auto
    from A3 ‹a∈C› have a∈A by auto
    with A1 A2 have f(r–a″) = R–f(a)″
      using ord˙iso˙pres˙rel˙image by simp
    moreover from A3 ‹f:A→B› ‹a∈C› have f(a) ∈ f(C)
```

```
    using func˙imagedef by auto
  ultimately have f(r–a″) ∈ – R–b″. b ∈ f(C) ″
    by auto
  with I have Y ∈ – R–b″. b ∈ f(C) ″ by simp
} then show –f(r–a″). a∈C″ ⊆ –R–b″. b ∈ f(C)″
  by blast
{ fix Y assume Y ∈ –R–b″. b ∈ f(C)″
  then obtain b where b ∈ f(C) and II: Y = R–b″
    by auto
  with A3 ‹f:A→B› obtain a where a∈C and b = f(a)
    using func˙imagedef by auto
  with A3 II have a∈A and Y = R–f(a)″ by auto
  with A1 A2 have Y = f(r–a″)
    using ord˙iso˙pres˙rel˙image by simp
  with ‹a∈C› have Y ∈ –f(r–a″). a∈C″ by auto
} then show –R–b″. b ∈ f(C)″ ⊆ –f(r–a″). a∈C″
  by auto
qed
```

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

```
lemma ord˙iso˙pres˙min˙up˙bounds:
  assumes A1: f ∈ ord˙iso(A,r,B,R) and  A2: r ⊆ A×A  R ⊆ B×B and
  A3: C⊆A and A4: C≠0
  shows f(⋂a∈C. r–a″) = (⋂b∈f(C). R–b″)
proof -
  from A1 have f ∈ inj(A,B)
    using ord˙iso˙is˙bij bij˙is˙inj by simp
  moreover note A4
  moreover from A2 A3 have ∀a∈C. r–a″ ⊆ A by auto
  ultimately have
    f(⋂a∈C. r–a″) = ( ⋂a∈C. f(r–a″) )
    using inj˙image˙of˙Inter by simp
  also from A1 A2 A3 have
    ( ⋂a∈C. f(r–a″) ) = ( ⋂b∈f(C). R–b″ )
    using ord˙iso˙pres˙up˙bounds by simp
  finally show f(⋂a∈C. r–a″) = (⋂b∈f(C). R–b″)
    by simp
qed
```

Order isomorphisms preserve completeness.

```
lemma ord˙iso˙pres˙compl:
  assumes A1: f ∈ ord˙iso(A,r,B,R) and
  A2: r ⊆ A×A  R ⊆ B×B and A3: R –is complete″
  shows r –is complete″
proof -
  { fix C
    assume A4: IsBoundedAbove(C,r)  C≠0
    with A1 A2 A3 have
```

```
    HasAminimum(R,⋂b ∈ f(C). R–b″)
    using ord˙iso˙pres˙bound˙above IsComplete˙def
    by simp
  moreover
  from A2 ‹IsBoundedAbove(C,r)› have I: C ⊆ A using Order˙ZF˙3˙L1A
    by blast
  with A1 A2 ‹C≠0› have f(⋂a∈C. r–a″) = (⋂b∈f(C). R–b″)
    using ord˙iso˙pres˙min˙up˙bounds by simp
  ultimately have HasAminimum(R,f(⋂a∈C. r–a″))
    by simp
  moreover
  from A2 have ∀a∈C. r–a″ ⊆ A
    by auto
  with ‹C≠0› 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
```

13 Finite sets - introduction

```
theory Finite˙ZF imports ZF1 Nat˙ZF˙IML ZF.Cardinal

begin
```

Standard Isabelle Finite.thy contains a very useful notion of finite powerset: the set of finite subsets of a given set. The definition, however, is specific to Isabelle and based on the notion of "datatype", obviously not something

that belongs to ZF set theory. This theory file devolops the notion of finite powerset similarly as in Finite.thy, but based on standard library's Cardinal.thy. This theory file is intended to replace IsarMathLib's Finite1 and Finite˙ZF˙1 theories that are currently derived from the "datatype" approach.

13.1 Definition and basic properties of finite powerset

The goal of this section is to prove an induction theorem about finite powersets: if the empty set has some property and this property is preserved by adding a single element of a set, then this property is true for all finite subsets of this set.

We defined the finite powerset FinPow(X) as those elements of the powerset that are finite.

```
definition
  FinPow(X) ≡ –A ∈ Pow(X). Finite(A)"
```

The cardinality of an element of finite powerset is a natural number.

```
lemma card˙fin˙is˙nat: assumes A ∈ FinPow(X)
  shows —A— ∈ nat and A ≈ —A—
  using assms FinPow˙def Finite˙def cardinal˙cong nat˙into˙Card
    Card˙cardinal˙eq by auto
```

A reformulation of card˙fin˙is˙nat: for a finit set A there is a bijection between $|A|$ and A.

```
lemma fin˙bij˙card: assumes A1: A ∈ FinPow(X)
  shows ∃b. b ∈ bij(—A—, A)
proof -
  from A1 have —A— ≈ A using card˙fin˙is˙nat eqpoll˙sym
    by blast
  then show thesis using eqpoll˙def by auto
qed
```

If a set has the same number of elements as $n \in \mathbb{N}$, then its cardinality is n. Recall that in set theory a natural number n is a set that has n elements.

```
lemma card˙card: assumes A ≈ n and n ∈ nat
  shows —A— = n
  using assms cardinal˙cong nat˙into˙Card Card˙cardinal˙eq
  by auto
```

If we add a point to a finite set, the cardinality increases by one. To understand the second assertion $|A \cup \{a\}| = |A| \cup \{|A|\}$ recall that the cardinality $|A|$ of A is a natural number and for natural numbers we have $n+1 = n \cup \{n\}$.

```
lemma card˙fin˙add˙one: assumes A1: A ∈ FinPow(X) and A2: a ∈ X-A
  shows
  —A ∪ –a"— = succ( —A— )
```

```
  —A ∪ –a″— = —A— ∪ ——A—″
proof -
  from A1 A2 have cons(a,A) ≈ cons( —A—, —A— )
    using card˙fin˙is˙nat mem˙not˙refl cons˙eqpoll˙cong
    by auto
  moreover have  cons(a,A) = A ∪ –a″ by (rule consdef)
  moreover have cons( —A—, —A— ) = —A— ∪ ——A—″
    by (rule consdef)
  ultimately have A∪–a″ ≈ succ( —A— ) using succ˙explained
    by simp
  with A1 show
    —A ∪ –a″— = succ( —A— ) and —A ∪ –a″— = —A— ∪ ——A—″
    using card˙fin˙is˙nat card˙card by auto
qed
```

We can decompose the finite powerset into collection of sets of the same natural cardinalities.

```
lemma finpow˙decomp:
  shows FinPow(X) = (⋃n ∈ nat. –A ∈ Pow(X). A ≈ n″)
  using Finite˙def FinPow˙def by auto
```

Finite powerset is the union of sets of cardinality bounded by natural numbers.

```
lemma finpow˙union˙card˙nat:
  shows FinPow(X) = (⋃n ∈ nat. –A ∈ Pow(X). A ≲ n″)
proof -
  have FinPow(X) ⊆ (⋃n ∈ nat. –A ∈ Pow(X). A ≲ n″)
    using finpow˙decomp FinPow˙def eqpoll˙imp˙lepoll
    by auto
  moreover have
    (⋃n ∈ nat. –A ∈ Pow(X). A ≲ n″) ⊆ FinPow(X)
    using lepoll˙nat˙imp˙Finite FinPow˙def by auto
  ultimately show thesis by auto
qed
```

A different form of finpow˙union˙card˙nat (see above) - a subset that has not more elements than a given natural number is in the finite powerset.

```
lemma lepoll˙nat˙in˙finpow:
  assumes n ∈ nat   A ⊆ X  A ≲ n
  shows A ∈ FinPow(X)
  using assms finpow˙union˙card˙nat by auto
```

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

```
lemma nat˙finpow˙nat: assumes n ∈ nat shows n ∈ FinPow(nat)
  using assms nat˙into˙Finite nat˙subset˙nat FinPow˙def
  by simp
```

A finite subset is a finite subset of itself.

```
lemma fin˙finpow˙self: assumes A ∈ FinPow(X) shows A ∈ FinPow(A)
  using assms FinPow˙def by auto
```

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

```
lemma rem˙add˙eq: assumes a∈A shows (A-–a″) ∪ –a″ = A
  using assms by auto
```

Induction for finite powerset. This is smilar to the standard Isabelle's Fin˙induct.

```
theorem FinPow˙induct: assumes A1: P(0) and
  A2: ∀ A ∈ FinPow(X). P(A) ⟶ (∀ a∈X. P(A ∪ –a″)) and
  A3: B ∈ FinPow(X)
  shows P(B)
proof -
  { fix n assume n ∈ nat
    moreover from A1 have I: ∀ B∈Pow(X). B ≲ 0 ⟶ P(B)
      using lepoll˙0˙is˙0 by auto
    moreover have ∀  k ∈ nat.
      (∀ B ∈ Pow(X). (B ≲ k ⟶ P(B))) ⟶
      (∀ B ∈ Pow(X). (B ≲ succ(k) ⟶ P(B)))
    proof -
      { fix k assume A4: k ∈ nat
 assume A5: ∀  B ∈ Pow(X). (B ≲ k ⟶ P(B))
 fix B assume A6: B ∈ Pow(X)  B ≲ succ(k)
 have P(B)
 proof -
   have B = 0 ⟶ P(B)
   proof -
     { assume B = 0
       then have B ≲ 0 using lepoll˙0˙iff
 by simp
       with I A6 have P(B) by simp
     } thus B = 0 ⟶ P(B) by simp
   qed
   moreover have B≠0 ⟶ P(B)
   proof -
     { assume B ≠ 0
       then obtain a where II: a∈B by auto
       let A = B - –a″
       from A6 II have A ⊆ X and A ≲ k
 using Diff˙sing˙lepoll by auto
       with A4 A5 have A ∈ FinPow(X) and P(A)
 using lepoll˙nat˙in˙finpow finpow˙decomp
 by auto
       with A2 A6 II have  P(A ∪ –a″)
 by auto
       moreover from II have A ∪ –a″ = B
 by auto
       ultimately have P(B) by simp
```

```
    } thus B≠0 ⟶ P(B) by simp
  qed
  ultimately show P(B) by auto
qed
    } thus thesis by blast
  qed
  ultimately have ∀B ∈ Pow(X). (B ≲ n ⟶ P(B))
    by (rule ind˙on˙nat)
 } then have ∀n ∈ nat. ∀B ∈ Pow(X). (B ≲ n ⟶ P(B))
  by auto
 with A3 show P(B) using finpow˙union˙card˙nat
  by auto
qed
```

A subset of a finite subset is a finite subset.

```
lemma subset˙finpow: assumes A ∈ FinPow(X) and B ⊆ A
 shows B ∈ FinPow(X)
 using assms FinPow˙def subset˙Finite by auto
```

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

```
lemma diff˙finpow:
 assumes A ∈ FinPow(X) shows A-B ∈ FinPow(X)
 using assms subset˙finpow by blast
```

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

```
corollary fin˙rem˙point˙fin: assumes A ∈ FinPow(X)
 shows A - –a″ ∈ FinPow(X)
 using assms diff˙finpow by simp
```

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

```
lemma card˙non˙empty˙succ:
 assumes A1: A ∈ FinPow(X) and A2: A ≠ 0
 shows ∃n ∈ nat. —A— = succ(n)
proof -
 from A2 obtain a where a ∈ A by auto
 let B = A - –a″
 from A1 ‹a ∈ A› have
  B ∈ FinPow(X) and a ∈ X - B
  using FinPow˙def fin˙rem˙point˙fin by auto
 then have —B ∪ –a″— = succ( —B— )
  using card˙fin˙add˙one by auto
 moreover from ‹a ∈ A› ‹B ∈ FinPow(X)› have
  A = B ∪ –a″ and —B— ∈ nat
  using card˙fin˙is˙nat by auto
 ultimately show ∃n ∈ nat. —A— = succ(n) by auto
qed
```

Nonempty set has non-zero cardinality. This is probably true without the assumption that the set is finite, but I couldn't derive it from standard

Isabelle theorems.

```
lemma card˙non˙empty˙non˙zero:
  assumes A ∈ FinPow(X) and A ≠ 0
  shows —A— ≠ 0
proof -
  from assms obtain n where —A— = succ(n)
    using card˙non˙empty˙succ by auto
  then show —A— ≠ 0 using succ˙not˙0
    by simp
qed
```

Another variation on the induction theme: If we can show something holds for the empty set and if it holds for all finite sets with at most k elements then it holds for all finite sets with at most $k+1$ elements, the it holds for all finite sets.

```
theorem FinPow˙card˙ind: assumes A1: P(0) and
  A2: ∀k∈nat.
  (∀A ∈ FinPow(X). A ≲ k ⟶ P(A)) ⟶
  (∀A ∈ FinPow(X). A ≲ succ(k) ⟶ P(A))
  and A3: A ∈  FinPow(X) shows P(A)
proof -
  from A3 have —A— ∈ nat and A ∈  FinPow(X) and A ≲ —A—
    using card˙fin˙is˙nat eqpoll˙imp˙lepoll by auto
  moreover have ∀n ∈ nat. (∀A ∈ FinPow(X).
    A ≲ n ⟶ P(A))
  proof
    fix n assume n ∈ nat
    moreover from A1 have ∀A ∈ FinPow(X). A ≲ 0 ⟶ P(A)
      using lepoll˙0˙is˙0 by auto
    moreover note A2
    ultimately show
      ∀A ∈ FinPow(X). A ≲ n ⟶ P(A)
      by (rule ind˙on˙nat)
  qed
  ultimately show P(A) by simp
qed
```

Another type of induction (or, maybe recursion). In the induction step we try to find a point in the set that if we remove it, the fact that the property holds for the smaller set implies that the property holds for the whole set.

```
lemma FinPow˙ind˙rem˙one: assumes A1: P(0) and
  A2: ∀  A ∈ FinPow(X). A ≠ 0 ⟶ (∃a∈A. P(A-–a″) ⟶ P(A))
  and A3: B ∈  FinPow(X)
  shows P(B)
proof -
  note A1
  moreover have ∀k∈nat.
  (∀B ∈ FinPow(X). B ≲ k ⟶ P(B)) ⟶
```

```
  (∀C ∈ FinPow(X). C ≲ succ(k) ⟶ P(C))
  proof -
    { fix k assume k ∈ nat
      assume A4: ∀B ∈ FinPow(X). B ≲ k ⟶ P(B)
      have ∀C ∈ FinPow(X). C ≲ succ(k) ⟶ P(C)
      proof -
 { fix C assume C ∈ FinPow(X)
   assume C ≲ succ(k)
   note A1
   moreover
   { assume C ≠ 0
     with A2 ‹C ∈ FinPow(X)› obtain a where
       a∈C and P(C-–a″) ⟶ P(C)
       by auto
     with A4 ‹C ∈ FinPow(X)› ‹C ≲ succ(k)›
     have P(C) using Diff˙sing˙lepoll fin˙rem˙point˙fin
       by simp }
   ultimately have P(C) by auto
 } thus thesis by simp
      qed
    } thus thesis by blast
  qed
  moreover note A3
  ultimately show P(B) by (rule FinPow˙card˙ind)
qed
```

Yet another induction theorem. This is similar, but slightly more complicated than FinPow˙ind˙rem˙one. The difference is in the treatment of the empty set to allow to show properties that are not true for empty set.

```
lemma FinPow˙rem˙ind: assumes A1: ∀A ∈ FinPow(X).
  A = 0 ∨ (∃a∈A. A = –a″ ∨ P(A-–a″) ⟶ P(A))
  and A2: A ∈  FinPow(X) and A3: A≠0
  shows P(A)
proof -
  have 0 = 0 ∨ P(0) by simp
  moreover have
    ∀k∈nat.
    (∀B ∈ FinPow(X). B ≲ k ⟶ (B=0 ∨ P(B))) ⟶
    (∀A ∈ FinPow(X). A ≲ succ(k) ⟶ (A=0 ∨ P(A)))
  proof -
    { fix k assume k ∈ nat
      assume A4: ∀B ∈ FinPow(X). B ≲ k ⟶ (B=0 ∨ P(B))
      have ∀A ∈ FinPow(X). A ≲ succ(k) ⟶ (A=0 ∨ P(A))
      proof -
 { fix A assume A ∈ FinPow(X)
   assume A ≲ succ(k)  A≠0
   from A1 ‹A ∈ FinPow(X)› ‹A≠0› obtain a
     where a∈A and A = –a″ ∨ P(A-–a″) ⟶ P(A)
     by auto
```

```
  let B = A-–a″
  from A4 ‹A ∈ FinPow(X)› ‹A ≲ succ(k)› ‹a∈A›
  have B = 0 ∨ P(B)
    using Diff˙sing˙lepoll fin˙rem˙point˙fin
    by simp
  with ‹a∈A› ‹A = –a″ ∨ P(A-–a″) ⟶ P(A)›
  have P(A) by auto
 } thus  thesis by auto
     qed
   } thus thesis by blast
 qed
 moreover note A2
 ultimately have A=0 ∨ P(A) by (rule FinPow˙card˙ind)
 with A3 show P(A) by simp
qed
```

If a family of sets is closed with respect to taking intersections of two sets then it is closed with respect to taking intersections of any nonempty finite collection.

```
lemma inter˙two˙inter˙fin:
  assumes A1: ∀V∈T. ∀W∈T. V ∩ W ∈ T and
  A2: N ≠ 0 and A3: N ∈ FinPow(T)
  shows (⋂N ∈ T)
proof -
  have 0 = 0 ∨ (⋂0 ∈ T) by simp
  moreover have ∀M ∈ FinPow(T). (M = 0 ∨ ⋂M ∈ T) ⟶
    (∀W ∈ T. M∪–W″ = 0 ∨ ⋂(M ∪ –W″) ∈ T)
  proof -
    { fix M assume M ∈ FinPow(T)
      assume A4: M = 0 ∨ ⋂M ∈ T
      { assume M = 0
 hence ∀W ∈ T. M∪–W″ = 0 ∨ ⋂(M ∪ –W″) ∈ T
   by auto }
      moreover
      { assume M ≠ 0
 with A4 have ⋂M ∈ T by simp
 { fix W assume W ∈ T
   from ‹M ≠ 0› have ⋂(M ∪ –W″) = (⋂M) ∩ W
     by auto
   with A1 ‹⋂M ∈ T› ‹W ∈ T› have ⋂(M ∪ –W″) ∈ T
     by simp
 } hence ∀W ∈ T. M∪–W″ = 0 ∨ ⋂(M ∪ –W″) ∈ T
   by simp }
      ultimately have ∀W ∈ T. M∪–W″ = 0 ∨ ⋂(M ∪ –W″) ∈ T
 by blast
    } thus thesis by simp
  qed
  moreover note ‹N ∈ FinPow(T)›
  ultimately have N = 0 ∨ (⋂N ∈ T)
```

```
    by (rule FinPow˙induct)
  with A2 show (⋂N ∈ T) by simp
qed
```

If a family of sets contains the empty set and is closed with respect to taking unions of two sets then it is closed with respect to taking unions of any finite collection.

```
lemma union˙two˙union˙fin:
  assumes A1: 0 ∈ C and A2: ∀A∈C. ∀B∈C. A∪B ∈ C and
  A3: N ∈ FinPow(C)
  shows ⋃N ∈ C
proof -
  from ‹0 ∈ C› have ⋃0 ∈ C by simp
  moreover have ∀M ∈ FinPow(C). ⋃M ∈ C ⟶ (∀A∈C. ⋃(M ∪ −A″) ∈ C)
  proof -
    { fix M assume M ∈ FinPow(C)
      assume ⋃M ∈ C
      fix A assume A∈C
      have ⋃(M ∪ −A″) = (⋃M) ∪ A by auto
      with A2 ‹⋃M ∈ C› ‹A∈C› have ⋃(M ∪ −A″) ∈ C
 by simp
    } thus thesis by simp
  qed
  moreover note ‹N ∈ FinPow(C)›
  ultimately show ⋃N ∈ C by (rule FinPow˙induct)
qed
```

Empty set is in finite power set.

```
lemma empty˙in˙finpow: shows 0 ∈ FinPow(X)
  using FinPow˙def by simp
```

Singleton is in the finite powerset.

```
lemma singleton˙in˙finpow: assumes x ∈ X
  shows −x″ ∈ FinPow(X) using assms FinPow˙def by simp
```

Union of two finite subsets is a finite subset.

```
lemma union˙finpow: assumes A ∈ FinPow(X) and B ∈ FinPow(X)
  shows A ∪ B ∈ FinPow(X)
  using assms FinPow˙def by auto
```

Union of finite number of finite sets is finite.

```
lemma fin˙union˙finpow: assumes M ∈ FinPow(FinPow(X))
  shows ⋃M ∈ FinPow(X)
  using assms empty˙in˙finpow union˙finpow union˙two˙union˙fin
  by simp
```

If a set is finite after removing one element, then it is finite.

```
lemma rem˙point˙fin˙fin:
```

```
  assumes A1: x ∈ X and A2: A - –x″ ∈ FinPow(X)
  shows A ∈ FinPow(X)
proof -
  from assms have (A - –x″) ∪ –x″ ∈ FinPow(X)
    using singleton˙in˙finpow union˙finpow by simp
  moreover have A ⊆ (A - –x″) ∪ –x″ by auto
  ultimately show A ∈ FinPow(X)
    using FinPow˙def subset˙Finite by auto
qed
```

An image of a finite set is finite.

```
lemma fin˙image˙fin: assumes ∀V∈B. K(V)∈C and N ∈ FinPow(B)
  shows –K(V). V∈N″ ∈ FinPow(C)
proof -
  have –K(V). V∈0″ ∈ FinPow(C) using FinPow˙def
    by auto
  moreover have ∀A ∈ FinPow(B).
   –K(V). V∈A″ ∈ FinPow(C) ⟶ (∀a∈B. –K(V). V ∈ (A ∪ –a″)″ ∈ FinPow(C))
  proof -
    { fix A assume A ∈ FinPow(B)
      assume  –K(V). V∈A″ ∈ FinPow(C)
      fix a assume a∈B
      have  –K(V). V ∈ (A ∪ –a″)″ ∈ FinPow(C)
      proof -
 have –K(V). V ∈ (A ∪ –a″)″ = –K(V). V∈A″ ∪ –K(a)″
   by auto
 moreover note ‹–K(V). V∈A″ ∈ FinPow(C)›
 moreover from ‹∀V∈B. K(V) ∈ C›  ‹a∈B› have –K(a)″ ∈  FinPow(C)
   using singleton˙in˙finpow by simp
 ultimately show thesis using union˙finpow by simp
      qed
    } thus thesis by simp
  qed
  moreover note ‹N ∈ FinPow(B)›
  ultimately show –K(V). V∈N″ ∈ FinPow(C)
    by (rule FinPow˙induct)
qed
```

Union of a finite indexed family of finite sets is finite.

```
lemma union˙fin˙list˙fin:
  assumes A1: n ∈ nat and A2: ∀k ∈ n. N(k) ∈ FinPow(X)
  shows
  –N(k). k ∈ n″ ∈  FinPow(FinPow(X)) and (⋃k ∈ n. N(k)) ∈ FinPow(X)
proof -
  from A1 have n ∈ FinPow(n)
    using nat˙finpow˙nat fin˙finpow˙self by auto
  with A2 show –N(k). k ∈ n″ ∈  FinPow(FinPow(X))
    by (rule fin˙image˙fin)
  then show (⋃k ∈ n. N(k)) ∈ FinPow(X)
```

using fin˙union˙finpow by simp
qed

end

14 Finite sets

theory Finite1 imports ZF.EquivClass ZF.Finite func1 ZF1

begin

This theory extends Isabelle standard Finite theory. It is obsolete and should not be used for new development. Use the Finite˙ZF instead.

14.1 Finite powerset

In this section we consider various properties of Fin datatype (even though there are no datatypes in ZF set theory).

In Topology˙ZF theory we consider induced topology that is obtained by taking a subset of a topological space. To show that a topology restricted to a subset is also a topology on that subset we may need a fact that if T is a collection of sets and A is a set then every finite collection $\{V_i\}$ is of the form $V_i = U_i \cap A$, where $\{U_i\}$ is a finite subcollection of T. This is one of those trivial facts that require suprisingly long formal proof. Actually, the need for this fact is avoided by requiring intersection two open sets to be open (rather than intersection of a finite number of open sets). Still, the fact is left here as an example of a proof by induction. We will use Fin˙induct lemma from Finite.thy. First we define a property of finite sets that we want to show.

```
definition
  Prfin(T,A,M) ≡ ( (M = 0) — (∃N∈ Fin(T). ∀V∈ M. ∃ U∈ N. (V = U∩A)))
```

Now we show the main induction step in a separate lemma. This will make the proof of the theorem FinRestr below look short and nice. The premises of the ind˙step lemma are those needed by the main induction step in lemma Fin˙induct (see standard Isabelle's Finite.thy).

```
lemma ind˙step: assumes  A: ∀ V∈ TA. ∃ U∈T. V=U∩A
  and A1: W∈TA and A2: M∈ Fin(TA)
  and A3: W∉M and A4: Prfin(T,A,M)
  shows Prfin(T,A,cons(W,M))
proof -
  { assume A7: M=0 have Prfin(T, A, cons(W, M))
    proof-
      from A1 A obtain U where A5: U∈T and A6: W=U∩A by fast
      let N = –U″
```

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

Now we are ready to prove the statement we need.

```
theorem FinRestr0: assumes A: ∀ V ∈ TA. ∃ U∈ T. V=U∩A
  shows ∀ M∈ Fin(TA). Prfin(T,A,M)
proof -
  { fix M
    assume M ∈ Fin(TA)
    moreover have Prfin(T,A,0) using Prfin˙def by simp
    moreover
    { fix W M assume W∈TA M∈ Fin(TA) W∉M Prfin(T,A,M)
      with A have Prfin(T,A,cons(W,M)) by (rule ind˙step) }
    ultimately have Prfin(T,A,M) by (rule Fin˙induct)
  } thus thesis by simp
qed
```

This is a different form of the above theorem:

```
theorem ZF1FinRestr:
  assumes A1:M∈ Fin(TA) and A2: M≠0
  and A3: ∀ V∈ TA. ∃ U∈ T. V=U∩A
  shows ∃N∈ Fin(T). (∀V∈ M. ∃ U∈ N. (V = U∩A)) ∧ N≠0
proof -
  from A3 A1 have Prfin(T,A,M) using FinRestr0 by blast
  then have ∃N∈ Fin(T). ∀V∈ M. ∃ U∈ N. (V = U∩A)
    using A2 Prfin˙def by simp
  then obtain N where
    D1:N∈ Fin(T) ∧ (∀V∈ M. ∃ U∈ N. (V = U∩A)) by auto
  with A2 have N≠0 by auto
  with D1 show thesis by auto
```

qed

Purely technical lemma used in Topology˙ZF˙1 to show that if a topology is T_2, then it is T_1.

```
lemma Finite1˙L2:
  assumes A:∃U V. (U∈T ∧ V∈T ∧ x∈U ∧ y∈V ∧ U∩V=0)
  shows ∃U∈T. (x∈U ∧ y∉U)
proof -
  from A obtain U V where D1:U∈T ∧ V∈T ∧ x∈U ∧ y∈V ∧ U∩V=0 by auto
  with D1 show thesis by auto
qed
```

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

```
lemma Finite1˙L3˙IndStep:
  assumes A1:∀A B. ((A∈C ∧ B∈C) ⟶ A∪B∈C)
  and A2: A∈C and A3: N∈Fin(C) and A4:A∉N and A5:⋃N ∈ C
  shows ⋃cons(A,N) ∈ C
proof -
  have ⋃ cons(A,N) = A∪ ⋃N by blast
  with A1 A2 A5 show thesis by simp
qed
```

The lemma: a collection closed with respect to taking a union of two sets is closed under taking finite unions.

```
lemma Finite1˙L3:
  assumes A1: 0 ∈ C and A2: ∀A B. ((A∈C ∧ B∈C) ⟶ A∪B∈C) and
  A3: N∈ Fin(C)
  shows ⋃N∈C
proof -
  note A3
  moreover from A1 have ⋃0 ∈ C by simp
  moreover
  { fix A N
    assume A∈C N∈Fin(C) A∉N ⋃N ∈ C
    with A2 have ⋃cons(A,N) ∈ C by (rule Finite1˙L3˙IndStep) }
  ultimately show ⋃N∈ C by (rule Fin˙induct)
qed
```

A collection closed with respect to taking a intersection of two sets is closed under taking finite intersections. Proof by induction with the induction step formulated in a separate lemma. This is sligltly more involved than the union case in Finite1˙L3, because the intersection of empty collection is undefined (or should be treated as such). To simplify notation we define the property to be proven for finite sets as a separate notion.

```
definition
```

```
  IntPr(T,N) ≡ (N = 0 — ⋂N ∈ T)
```

The induction step.

```
lemma Finite1˙L4˙IndStep:
  assumes A1: ∀A B. ((A∈T ∧ B∈T) ⟶ A∩B∈T)
  and A2: A∈T and A3:N∈Fin(T) and A4:A∉N and A5:IntPr(T,N)
  shows IntPr(T,cons(A,N))
proof -
  { assume A6: N=0
    with A2 have IntPr(T,cons(A,N))
      using IntPr˙def by simp }
  moreover
  { assume A7: N≠0 have IntPr(T, cons(A, N))
    proof -
      from A7 A5 A2 A1 have ⋂N ∩ A ∈ T using IntPr˙def by simp
      moreover from A7 have ⋂cons(A, N) = ⋂N ∩ A by auto
      ultimately show IntPr(T, cons(A, N)) using IntPr˙def by simp
    qed }
  ultimately show thesis by auto
qed
```

The lemma.

```
lemma Finite1˙L4:
  assumes A1: ∀A B. A∈T ∧ B∈T ⟶ A∩B ∈ T
  and A2: N∈Fin(T)
  shows IntPr(T,N)
proof -
  note A2
  moreover have IntPr(T,0) using IntPr˙def by simp
  moreover
  { fix A N
    assume A∈T N∈Fin(T) A∉N IntPr(T,N)
    with A1 have  IntPr(T,cons(A,N)) by (rule Finite1˙L4˙IndStep) }
  ultimately show IntPr(T,N) by (rule Fin˙induct)
qed
```

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

```
lemma Finite1˙L5:
  assumes A1: ∀A B. ((A∈T ∧ B∈T) ⟶ A∩B∈T)
  and A2: N≠0 and A3: N∈Fin(T)
  shows ⋂N ∈ T
proof -
  from A1 A3 have IntPr(T,N) using Finite1˙L4 by simp
  with A2 show thesis using IntPr˙def by simp
qed
```

The images of finite subsets by a meta-function are finite. For example in topology if we have a finite collection of sets, then closing each of them

results in a finite collection of closed sets. This is a very useful lemma with many unexpected applications. The proof is by induction. The next lemma is the induction step.

```
lemma fin˙image˙fin˙IndStep:
  assumes ∀V∈B. K(V)∈C
  and U∈B and N∈Fin(B) and U∉N and –K(V). V∈N″∈Fin(C)
  shows –K(V). V∈cons(U,N)″ ∈ Fin(C)
  using assms by simp
```

The lemma:

```
lemma fin˙image˙fin:
  assumes A1: ∀V∈B. K(V)∈C and A2: N∈Fin(B)
  shows –K(V). V∈N″ ∈ Fin(C)
proof -
  note A2
  moreover have –K(V). V∈0″ ∈ Fin(C) by simp
  moreover
  { fix U N
    assume U∈B N∈Fin(B) U∉N –K(V). V∈N″∈Fin(C)
    with A1 have –K(V). V∈cons(U,N)″ ∈ Fin(C)
      by (rule fin˙image˙fin˙IndStep) }
  ultimately show thesis by (rule Fin˙induct)
qed
```

The image of a finite set is finite.

```
lemma Finite1˙L6A: assumes A1: f:X→Y and A2: N ∈ Fin(X)
  shows f(N) ∈ Fin(Y)
proof -
  from A1 have ∀x∈X. f(x) ∈ Y
    using apply˙type by simp
  moreover note A2
  ultimately have –f(x). x∈N″ ∈ Fin(Y)
    by (rule fin˙image˙fin)
  with A1 A2 show thesis
    using FinD func˙imagedef by simp
qed
```

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

```
lemma Finite1˙L6B:
  assumes A1: ∀x∈X. a(x) ∈ Y and A2: –b(y).y∈Y″ ∈ Fin(Z)
  shows –b(a(x)).x∈X″ ∈ Fin(Z)
proof -
  from A1 have –b(a(x)).x∈X″ ⊆ –b(y).y∈Y″ by auto
  with A2 show thesis using Fin˙subset˙lemma by blast
qed
```

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

```
lemma Finite1˙L6C:
  assumes A1: ∀y∈Y. b(y) ∈ Z and A2: –a(x). x∈X″ ∈ Fin(Y)
  shows –b(a(x)).x∈X″ ∈ Fin(Z)
proof -
  let N = –a(x). x∈X″
  from A1 A2 have –b(y). y ∈ N″ ∈ Fin(Z)
    by (rule fin˙image˙fin)
  moreover have –b(a(x)). x∈X″ = –b(y). y∈ N″
    by auto
  ultimately show thesis by simp
qed
```

Cartesian product of finite sets is finite.

```
lemma Finite1˙L12: assumes A1: A ∈ Fin(A) and A2: B ∈ Fin(B)
  shows A×B ∈ Fin(A×B)
proof -
  have T1:∀a∈A. ∀b∈B. –⟨ a,b⟩″ ∈ Fin(A×B) by simp
  have ∀a∈A. —⟨ a,b⟩″. b ∈ B″ ∈ Fin(Fin(A×B))
  proof
    fix a assume A3: a ∈ A
    with T1 have ∀b∈B. –⟨ a,b⟩″ ∈ Fin(A×B)
      by simp
    moreover note A2
    ultimately show —⟨ a,b⟩″. b ∈ B″ ∈ Fin(Fin(A×B))
      by (rule fin˙image˙fin)
  qed
  then have ∀a∈A. ⋃ —⟨ a,b⟩″. b ∈ B″ ∈ Fin(A×B)
    using Fin˙UnionI by simp
  moreover have
    ∀a∈A. ⋃ —⟨ a,b⟩″. b ∈ B″ = –a″× B by blast
  ultimately have ∀a∈A. –a″× B ∈ Fin(A×B) by simp
  moreover note A1
  ultimately have —a″× B. a∈A″ ∈ Fin(Fin(A×B))
    by (rule fin˙image˙fin)
  then have ⋃—a″× B. a∈A″ ∈ Fin(A×B)
    using Fin˙UnionI by simp
  moreover have ⋃—a″× B. a∈A″ = A×B by blast
  ultimately show thesis by simp
qed
```

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

```
definition
  Characteristic(A,default,x) ≡ (if x∈A then x else default)
```

A finite subset is a finite subset of itself.

```
lemma Finite1˙L13:
  assumes A1:A ∈ Fin(X) shows A ∈ Fin(A)
proof -
```

```
  { assume A=0 hence A ∈ Fin(A) by simp }
  moreover
  { assume A2: A≠0 then obtain c where D1:c∈A
      by auto
    then have ∀x∈X. Characteristic(A,c,x) ∈ A
      using Characteristic˙def by simp
    moreover note A1
    ultimately have
      –Characteristic(A,c,x). x∈A″ ∈ Fin(A) by (rule fin˙image˙fin)
    moreover from D1 have
      –Characteristic(A,c,x). x∈A″ = A using Characteristic˙def by simp
    ultimately have A ∈ Fin(A) by simp }
  ultimately show thesis by blast
qed
```

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

```
lemma Finite1˙L14: assumes A1: A ∈ Fin(X) B ∈ Fin(Y)
  shows A×B ∈ Fin(X×Y)
proof -
  from A1 have A×B ⊆ X×Y using FinD by auto
  then have Fin(A×B) ⊆ Fin(X×Y) using Fin˙mono by simp
  moreover from A1 have A×B ∈ Fin(A×B)
    using Finite1˙L13 Finite1˙L12 by simp
  ultimately show thesis by auto
qed
```

The next lemma is needed in the Group˙ZF˙3 theory in a couple of places.

```
lemma Finite1˙L15:
  assumes A1: –b(x). x∈A″ ∈ Fin(B)  –c(x). x∈A″ ∈ Fin(C)
  and A2: f : B×C→E
  shows –f⟨ b(x),c(x)⟩. x∈A″ ∈ Fin(E)
proof -
  from A1 have –b(x). x∈A″×–c(x). x∈A″ ∈ Fin(B×C)
    using Finite1˙L14 by simp
  moreover have
    –⟨ b(x),c(x)⟩. x∈A″ ⊆ –b(x). x∈A″×–c(x). x∈A″
    by blast
  ultimately have T0: –⟨ b(x),c(x)⟩. x∈A″ ∈ Fin(B×C)
    by (rule Fin˙subset˙lemma)
  with A2 have T1: f–⟨ b(x),c(x)⟩. x∈A″ ∈ Fin(E)
    using Finite1˙L6A by auto
  from T0 have ∀x∈A. ⟨ b(x),c(x)⟩ ∈ B×C
    using FinD by auto
  with A2 have
    f–⟨ b(x),c(x)⟩. x∈A″ = –f⟨ b(x),c(x)⟩. x∈A″
    using func1˙1˙L17 by simp
  with T1 show thesis by simp
qed
```

Singletons are in the finite powerset.

```
lemma Finite1˙L16: assumes x∈X shows –x″ ∈ Fin(X)
  using assms emptyI consI by simp
```

A special case of Finite1˙L15 where the second set is a singleton. In Group˙ZF˙3 theory this corresponds to the situation where we multiply by a constant.

```
lemma Finite1˙L16AA: assumes –b(x). x∈A″ ∈ Fin(B)
  and c∈C and f : B×C→E
  shows –f⟨ b(x),c⟩. x∈A″ ∈ Fin(E)
proof -
  from assms have
    ∀y∈B. f⟨y,c⟩ ∈ E
    –b(x). x∈A″ ∈ Fin(B)
    using apply˙funtype by auto
  then show thesis by (rule Finite1˙L6C)
qed
```

First order version of the induction for the finite powerset.

```
lemma Finite1˙L16B: assumes A1: P(0) and A2: B∈Fin(X)
  and A3: ∀A∈Fin(X).∀x∈X. x∉A ∧ P(A)⟶P(A∪–x″)
  shows P(B)
proof -
  note ‹B∈Fin(X)› and ‹P(0)›
  moreover
  { fix A x
    assume  x ∈ X  A ∈ Fin(X)  x ∉ A  P(A)
    moreover have cons(x,A) = A∪–x″ by auto
    moreover note A3
    ultimately have P(cons(x,A)) by simp }
  ultimately show  P(B) by (rule Fin˙induct)
qed
```

14.2 Finite range functions

In this section we define functions $f : X \to Y$, with the property that $f(X)$ is a finite subset of Y. Such functions play a important role in the construction of real numbers in the Real˙ZF series.

Definition of finite range functions.

```
definition
  FinRangeFunctions(X,Y) ≡ –f:X→Y. f(X) ∈ Fin(Y)″
```

Constant functions have finite range.

```
lemma Finite1˙L17: assumes c∈Y and X≠0
  shows ConstantFunction(X,c) ∈ FinRangeFunctions(X,Y)
  using assms  func1˙3˙L1 func˙imagedef func1˙3˙L2 Finite1˙L16
    FinRangeFunctions˙def by simp
```

Finite range functions have finite range.

```
lemma Finite1˙L18: assumes f ∈ FinRangeFunctions(X,Y)
  shows –f(x). x∈X″ ∈ Fin(Y)
  using assms FinRangeFunctions˙def func˙imagedef by simp
```

An alternative form of the definition of finite range functions.

```
lemma Finite1˙L19: assumes f:X→Y
  and –f(x). x∈X″ ∈ Fin(Y)
  shows f ∈ FinRangeFunctions(X,Y)
  using assms func˙imagedef FinRangeFunctions˙def by simp
```

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

```
lemma Finite1˙L20: assumes A1:f ∈ FinRangeFunctions(X,Y)
  and A2: g : Y→Z
  shows g O f ∈ FinRangeFunctions(X,Z)
proof -
  from A1 A2 have g–f(x). x∈X″ ∈ Fin(Z)
    using Finite1˙L18 Finite1˙L6A
    by simp
  with A1 A2 have –(g O f)(x). x∈X″ ∈ Fin(Z)
    using FinRangeFunctions˙def apply˙funtype
      func1˙1˙L17 comp˙fun˙apply by auto
  with A1 A2 show thesis using
    FinRangeFunctions˙def comp˙fun Finite1˙L19
    by auto
qed
```

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

```
lemma Finite1˙L21:
  assumes f ∈ FinRangeFunctions(X,Y) and A⊆X
  shows f(A) ∈ Fin(Y)
proof -
  from assms have f(X) ∈ Fin(Y)  f(A) ⊆ f(X)
    using FinRangeFunctions˙def func1˙1˙L8
    by auto
  then show f(A) ∈ Fin(Y) using Fin˙subset˙lemma
    by blast
qed

end
```

15 Finite sets 1

```
theory Finite˙ZF˙1 imports Finite1 Order˙ZF˙1a

begin
```

This theory is based on Finite1 theory and is obsolete. It contains properties

of finite sets related to order relations. See the FinOrd theory for a better approach.

15.1 Finite vs. bounded sets

The goal of this section is to show that finite sets are bounded and have maxima and minima.

Finite set has a maximum - induction step.

```
lemma Finite˙ZF˙1˙1˙L1:
  assumes A1: r –is total on″ X and A2: trans(r)
  and A3: A∈Fin(X) and A4: x∈X and A5: A=0 ∨ HasAmaximum(r,A)
  shows A∪–x″ = 0 ∨ HasAmaximum(r,A∪–x″)
proof -
  { assume A=0 then have T1: A∪–x″ = –x″ by simp
    from A1 have refl(X,r) using total˙is˙refl by simp
    with T1 A4 have A∪–x″ = 0 ∨ HasAmaximum(r,A∪–x″)
      using Order˙ZF˙4˙L8 by simp }
  moreover
  { assume A≠0
    with A1 A2 A3 A4 A5 have A∪–x″ = 0 ∨ HasAmaximum(r,A∪–x″)
      using FinD Order˙ZF˙4˙L9 by simp }
  ultimately show thesis by blast
qed
```

For total and transitive relations finite set has a maximum.

```
theorem Finite˙ZF˙1˙1˙T1A:
  assumes A1: r –is total on″ X and A2: trans(r)
  and A3: B∈Fin(X)
  shows B=0 ∨ HasAmaximum(r,B)
proof -
  have 0=0 ∨ HasAmaximum(r,0) by simp
  moreover note A3
  moreover from A1 A2 have ∀A∈Fin(X). ∀x∈X.
   x∉A ∧ (A=0 ∨ HasAmaximum(r,A)) ⟶ (A∪–x″=0 ∨ HasAmaximum(r,A∪–x″))
    using Finite˙ZF˙1˙1˙L1 by simp
  ultimately show  B=0 ∨ HasAmaximum(r,B) by (rule Finite1˙L16B)
qed
```

Finite set has a minimum - induction step.

```
lemma Finite˙ZF˙1˙1˙L2:
  assumes A1: r –is total on″ X and A2: trans(r)
  and A3: A∈Fin(X) and A4: x∈X and A5: A=0 ∨ HasAminimum(r,A)
  shows A∪–x″ = 0 ∨ HasAminimum(r,A∪–x″)
proof -
  { assume A=0 then have T1: A∪–x″ = –x″ by simp
    from A1 have refl(X,r) using total˙is˙refl by simp
    with T1 A4 have A∪–x″ = 0 ∨ HasAminimum(r,A∪–x″)
```

```
    using Order˙ZF˙4˙L8 by simp }
  moreover
  { assume A≠0
    with A1 A2 A3 A4 A5 have A∪–x″ = 0 ∨ HasAminimum(r,A∪–x″)
      using FinD Order˙ZF˙4˙L10 by simp }
  ultimately show thesis by blast
qed
```

For total and transitive relations finite set has a minimum.

```
theorem Finite˙ZF˙1˙1˙T1B:
  assumes A1: r –is total on″ X and A2: trans(r)
  and A3: B ∈ Fin(X)
  shows B=0 ∨ HasAminimum(r,B)
proof -
  have 0=0 ∨ HasAminimum(r,0) by simp
  moreover note A3
  moreover from A1 A2 have ∀A∈Fin(X). ∀x∈X.
   x∉A ∧ (A=0 ∨ HasAminimum(r,A)) ⟶ (A∪–x″=0 ∨ HasAminimum(r,A∪–x″))
    using Finite˙ZF˙1˙1˙L2 by simp
  ultimately show  B=0 ∨ HasAminimum(r,B) by (rule Finite1˙L16B)
qed
```

For transitive and total relations finite sets are bounded.

```
theorem Finite˙ZF˙1˙T1:
  assumes A1: r –is total on″ X and A2: trans(r)
  and A3: B∈Fin(X)
  shows IsBounded(B,r)
proof -
  from A1 A2 A3 have B=0 ∨ HasAminimum(r,B) B=0 ∨ HasAmaximum(r,B)
    using Finite˙ZF˙1˙1˙T1A Finite˙ZF˙1˙1˙T1B by auto
  then have
    B = 0 ∨ IsBoundedBelow(B,r) B = 0 ∨ IsBoundedAbove(B,r)
    using Order˙ZF˙4˙L7 Order˙ZF˙4˙L8A by auto
  then show IsBounded(B,r) using
    IsBounded˙def IsBoundedBelow˙def IsBoundedAbove˙def
    by simp
qed
```

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

```
theorem Finite˙ZF˙1˙T2:
  assumes A1: IsLinOrder(X,r) and A2: A ∈ Fin(X) and A3: A≠0
  shows
  Maximum(r,A) ∈ A
  Minimum(r,A) ∈ A
  ∀x∈A. ⟨x,Maximum(r,A)⟩ ∈ r
```

```
  ∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r
proof -
  from A1 have T1: r –is total on″ X trans(r) antisym(r)
    using IsLinOrder˙def by auto
  moreover from T1 A2 A3 have HasAmaximum(r,A)
    using Finite˙ZF˙1˙1˙T1A by auto
  moreover from T1 A2 A3 have HasAminimum(r,A)
    using Finite˙ZF˙1˙1˙T1B by auto
  ultimately show
    Maximum(r,A) ∈ A
    Minimum(r,A) ∈ A
    ∀x∈A. ⟨x,Maximum(r,A)⟩ ∈ r ∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r
    using Order˙ZF˙4˙L3 Order˙ZF˙4˙L4 by auto
qed
```

A special case of Finite˙ZF˙1˙T2 when the set has three elements.

```
corollary Finite˙ZF˙1˙L2A:
  assumes A1: IsLinOrder(X,r) and A2: a∈X  b∈X  c∈X
  shows
  Maximum(r,–a,b,c″) ∈ –a,b,c″
  Minimum(r,–a,b,c″) ∈ –a,b,c″
  Maximum(r,–a,b,c″) ∈ X
  Minimum(r,–a,b,c″) ∈ X
  ⟨a,Maximum(r,–a,b,c″)⟩ ∈ r
  ⟨b,Maximum(r,–a,b,c″)⟩ ∈ r
  ⟨c,Maximum(r,–a,b,c″)⟩ ∈ r
proof -
  from A2 have I: –a,b,c″ ∈ Fin(X)  –a,b,c″ ≠ 0
    by auto
  with A1 show II: Maximum(r,–a,b,c″) ∈ –a,b,c″
    by (rule Finite˙ZF˙1˙T2)
  moreover from A1 I show III: Minimum(r,–a,b,c″) ∈ –a,b,c″
    by (rule Finite˙ZF˙1˙T2)
  moreover from A2 have –a,b,c″ ⊆ X
    by auto
  ultimately show
    Maximum(r,–a,b,c″) ∈ X
    Minimum(r,–a,b,c″) ∈ X
    by auto
  from A1 I have ∀x∈–a,b,c″. ⟨x,Maximum(r,–a,b,c″)⟩ ∈ r
    by (rule Finite˙ZF˙1˙T2)
  then show
    ⟨a,Maximum(r,–a,b,c″)⟩ ∈ r
    ⟨b,Maximum(r,–a,b,c″)⟩ ∈ r
    ⟨c,Maximum(r,–a,b,c″)⟩ ∈ r
    by auto
qed
```

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

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

```
lemma Finite˙ZF˙1˙1˙L3:
  assumes A1: r –is total on″ X
  and A2: trans(r) and A3: antisym(r)
  and A4: r ⊆ X×X and A5: X≠0
  and A6: ∀x∈X. ∃a∈A. x≠a ∧ ⟨x,a⟩ ∈ r
  shows A ∉ Fin(X)
proof -
  from assms have ¬IsBounded(A,r)
    using Order˙ZF˙3˙L14 IsBounded˙def
    by simp
  with A1 A2 show A ∉ Fin(X)
    using Finite˙ZF˙1˙T1 by auto
qed

end
```

16 Finite sets and order relations

theory FinOrd˙ZF imports Finite˙ZF func˙ZF˙1

begin

This theory file contains properties of finite sets related to order relations. Part of this is similar to what is done in Finite˙ZF˙1 except that the development is based on the notion of finite powerset defined in Finite˙ZF rather the one defined in standard Isabelle Finite theory.

16.1 Finite vs. bounded sets

The goal of this section is to show that finite sets are bounded and have maxima and minima.

For total and transitive relations nonempty finite set has a maximum.

```
theorem fin˙has˙max:
  assumes A1: r –is total on″ X and A2: trans(r)
  and A3: B ∈ FinPow(X) and A4: B ≠ 0
  shows HasAmaximum(r,B)
proof -
  have 0=0 ∨ HasAmaximum(r,0) by simp
  moreover have
    ∀A ∈ FinPow(X). A=0 ∨ HasAmaximum(r,A) ⟶
    (∀x∈X. (A ∪ –x″) = 0 ∨ HasAmaximum(r,A ∪ –x″))
  proof -
    { fix A
      assume A ∈ FinPow(X)  A = 0 ∨ HasAmaximum(r,A)
```

```
      have ∀x∈X. (A ∪ –x″) = 0 ∨ HasAmaximum(r,A ∪ –x″)
      proof -
 { fix x assume x∈X
   note ‹A = 0 ∨ HasAmaximum(r,A)›
   moreover
   { assume A = 0
     then have A∪–x″ = –x″ by simp
     from A1 have refl(X,r) using total˙is˙refl
       by simp
     with ‹x∈X› ‹A∪–x″ = –x″› have HasAmaximum(r,A∪–x″)
       using Order˙ZF˙4˙L8 by simp }
   moreover
   { assume HasAmaximum(r,A)
     with A1 A2 ‹A ∈ FinPow(X)› ‹x∈X›
     have HasAmaximum(r,A∪–x″)
       using FinPow˙def Order˙ZF˙4˙L9 by simp }
   ultimately  have A ∪ –x″ = 0 ∨ HasAmaximum(r,A ∪ –x″)
     by auto
 } thus ∀x∈X. (A ∪ –x″) = 0 ∨ HasAmaximum(r,A ∪ –x″)
   by simp
      qed
    } thus thesis by simp
  qed
  moreover note A3
  ultimately have B = 0 ∨  HasAmaximum(r,B)
    by (rule FinPow˙induct)
  with A4 show HasAmaximum(r,B) by simp
qed
```

For linearly ordered nonempty finite sets the maximum is in the set and indeed it is the greatest element of the set.

```
lemma linord˙max˙props: assumes A1: IsLinOrder(X,r) and
  A2: A ∈ FinPow(X) A ≠ 0
  shows
  Maximum(r,A) ∈ A
  Maximum(r,A) ∈ X
  ∀a∈A. ⟨a,Maximum(r,A)⟩ ∈ r
proof -
  from A1 A2 show
    Maximum(r,A) ∈ A and ∀a∈A. ⟨a,Maximum(r,A)⟩ ∈ r
    using IsLinOrder˙def fin˙has˙max Order˙ZF˙4˙L3
    by auto
  with A2 show Maximum(r,A) ∈ X using FinPow˙def
    by auto
qed
```

16.2 Order isomorphisms of finite sets

In this section we eastablish that if two linearly ordered finite sets have the same number of elements, then they are order-isomorphic and the isomorphism is unique. This allows us to talk about "enumeration" of a linearly ordered finite set. We define the enumeration as the order isomorphism between the number of elements of the set (which is a natural number $n = \{0, 1, .., n-1\}$) and the set.

A really weird corner case - empty set is order isomorphic with itself.

```
lemma empty˙ord˙iso: shows ord˙iso(0,r,0,R) ≠ 0
proof -
  have 0 ≈ 0 using eqpoll˙refl by simp
  then obtain f where f ∈ bij(0,0)
    using eqpoll˙def by blast
  then show thesis using ord˙iso˙def by auto
qed
```

Even weirder than empty˙ord˙iso The order automorphism of the empty set is unique.

```
lemma empty˙ord˙iso˙uniq:
  assumes f ∈ ord˙iso(0,r,0,R)  g ∈ ord˙iso(0,r,0,R)
  shows f = g
proof -
  from assms have f : 0 → 0 and g: 0 → 0
    using ord˙iso˙def bij˙def surj˙def by auto
    moreover have ∀x∈0. f(x) = g(x) by simp
    ultimately show f = g by (rule func˙eq)
qed
```

The empty set is the only order automorphism of itself.

```
lemma empty˙ord˙iso˙empty: shows ord˙iso(0,r,0,R) = –0″
proof -
  have 0 ∈ ord˙iso(0,r,0,R)
  proof -
    have ord˙iso(0,r,0,R) ≠ 0 by (rule empty˙ord˙iso)
    then obtain f where f ∈ ord˙iso(0,r,0,R) by auto
    then show 0 ∈ ord˙iso(0,r,0,R)
      using ord˙iso˙def bij˙def surj˙def fun˙subset˙prod
      by auto
  qed
  then show ord˙iso(0,r,0,R) = –0″ using empty˙ord˙iso˙uniq
    by blast
qed
```

An induction (or maybe recursion?) scheme for linearly ordered sets. The induction step is that we show that if the property holds when the set is a singleton or for a set with the maximum removed, then it holds for the

set. The idea is that since we can build any finite set by adding elements on the right, then if the property holds for the empty set and is invariant with respect to this operation, then it must hold for all finite sets.

```
lemma fin˙ord˙induction:
  assumes A1: IsLinOrder(X,r) and A2: P(0) and
  A3: ∀ A ∈ FinPow(X). A ≠ 0 ⟶ (P(A - –Maximum(r,A)″) ⟶ P(A))
  and A4: B ∈ FinPow(X) shows P(B)
proof -
  note A2
  moreover have ∀  A ∈ FinPow(X). A ≠ 0 ⟶ (∃ a∈A. P(A-–a″) ⟶ P(A))
  proof -
    { fix A assume A ∈  FinPow(X) and A ≠ 0
      with A1 A3 have ∃ a∈A. P(A-–a″) ⟶ P(A)
 using IsLinOrder˙def fin˙has˙max
   IsLinOrder˙def Order˙ZF˙4˙L3
 by blast
    } thus thesis by simp
  qed
  moreover note A4
  ultimately show P(B) by (rule FinPow˙ind˙rem˙one)
qed
```

A sligltly more complicated version of fin˙ord˙induction that allows to prove properties that are not true for the empty set.

```
lemma fin˙ord˙ind:
  assumes A1: IsLinOrder(X,r) and A2: ∀ A ∈ FinPow(X).
  A = 0 ∨ (A = –Maximum(r,A)″ ∨ P(A - –Maximum(r,A)″) ⟶ P(A))
  and A3: B ∈  FinPow(X) and A4: B≠0
  shows P(B)
proof -
  { fix A assume A ∈  FinPow(X) and A ≠ 0
    with A1 A2 have
      ∃ a∈A. A = –a″ ∨ P(A-–a″) ⟶ P(A)
      using IsLinOrder˙def fin˙has˙max
 IsLinOrder˙def Order˙ZF˙4˙L3
      by blast
  } then have ∀ A ∈ FinPow(X).
      A = 0 ∨ (∃ a∈A. A = –a″ ∨ P(A-–a″) ⟶ P(A))
    by auto
  with A3 A4 show P(B) using  FinPow˙rem˙ind
    by simp
qed
```

Yet another induction scheme. We build a linearly ordered set by adding elements that are greater than all elements in the set.

```
lemma fin˙ind˙add˙max:
  assumes A1: IsLinOrder(X,r) and A2: P(0) and A3: ∀  A ∈ FinPow(X).
  ( ∀  x ∈ X-A. P(A) ∧ (∀ a∈A. ⟨a,x⟩ ∈ r ) ⟶ P(A ∪ –x″))
```

```
  and A4: B ∈ FinPow(X)
  shows P(B)
proof -
  note A1 A2
  moreover have
    ∀C ∈ FinPow(X). C ≠ 0 ⟶ (P(C - –Maximum(r,C)″) ⟶ P(C))
    proof -
      { fix C assume C ∈ FinPow(X) and C ≠ 0
 let x = Maximum(r,C)
 let A = C - –x″
 assume P(A)
 moreover from ‹C ∈ FinPow(X)› have A ∈ FinPow(X)
   using fin˙rem˙point˙fin by simp
 moreover from A1 ‹C ∈ FinPow(X)› ‹C ≠ 0› have
   x ∈ C and x ∈ X - A and ∀a∈A. ⟨a,x⟩ ∈ r
   using linord˙max˙props by auto
 moreover note A3
 ultimately have P(A ∪ –x″) by auto
 moreover from ‹x ∈ C› have A ∪ –x″ = C
   by auto
 ultimately have P(C) by simp
      } thus thesis by simp
    qed
    moreover note A4
  ultimately show P(B) by (rule fin˙ord˙induction)
qed
```

The only order automorphism of a linearly ordered finite set is the identity.

```
theorem fin˙ord˙auto˙id: assumes A1: IsLinOrder(X,r)
  and A2: B ∈  FinPow(X) and A3: B≠0
  shows ord˙iso(B,r,B,r) = –id(B)″
proof -
  note A1
  moreover
  { fix A assume A ∈  FinPow(X) A≠0
    let M = Maximum(r,A)
    let A0 = A - –M″
    assume A = –M″ ∨ ord˙iso(A0,r,A0,r) = –id(A0)″
    moreover
    { assume A = –M″
      have ord˙iso(–M″,r,–M″,r) = –id(–M″)″
 using id˙ord˙auto˙singleton by simp
      with ‹A = –M″› have ord˙iso(A,r,A,r) = –id(A)″
 by simp }
    moreover
    { assume ord˙iso(A0,r,A0,r) = –id(A0)″
      have ord˙iso(A,r,A,r) = –id(A)″
      proof
 show –id(A)″ ⊆ ord˙iso(A,r,A,r)
```

```
  using id˙ord˙iso by simp
{ fix f assume f ∈ ord˙iso(A,r,A,r)
  with A1 ‹A ∈  FinPow(X)› ‹A≠0› have
    restrict(f,A0) ∈ ord˙iso(A0, r, A-–f(M)″,r)
    using IsLinOrder˙def fin˙has˙max ord˙iso˙rem˙max
    by auto
  with A1 ‹A ∈  FinPow(X)› ‹A≠0› ‹f ∈ ord˙iso(A,r,A,r)›
    ‹ord˙iso(A0,r,A0,r) = –id(A0)″›
  have restrict(f,A0) = id(A0)
    using IsLinOrder˙def fin˙has˙max max˙auto˙fixpoint
    by auto
  moreover from A1 ‹f ∈ ord˙iso(A,r,A,r)›
    ‹A ∈  FinPow(X)› ‹A≠0› have
    f : A → A and M ∈ A and f(M) = M
    using ord˙iso˙def bij˙is˙fun IsLinOrder˙def
      fin˙has˙max Order˙ZF˙4˙L3 max˙auto˙fixpoint
    by auto
  ultimately have f = id(A) using id˙fixpoint˙rem
    by simp
} then show ord˙iso(A,r,A,r) ⊆ –id(A)″
  by auto
     qed
   }
   ultimately have ord˙iso(A,r,A,r) = –id(A)″
     by auto
 } then have ∀A ∈  FinPow(X). A = 0 ∨
     (A = –Maximum(r,A)″ ∨
     ord˙iso(A-–Maximum(r,A)″,r,A-–Maximum(r,A)″,r) =
     –id(A-–Maximum(r,A)″)″ ⟶  ord˙iso(A,r,A,r) = –id(A)″)
   by auto
 moreover note A2 A3
 ultimately show ord˙iso(B,r,B,r) = –id(B)″
   by (rule fin˙ord˙ind)
qed
```

Every two finite linearly ordered sets are order isomorphic. The statement is formulated to make the proof by induction on the size of the set easier, see fin˙ord˙iso˙ex for an alternative formulation.

```
lemma fin˙order˙iso:
  assumes A1: IsLinOrder(X,r)  IsLinOrder(Y,R) and
  A2: n ∈ nat
  shows ∀A ∈ FinPow(X). ∀B ∈ FinPow(Y).
  A ≈ n ∧ B ≈ n ⟶ ord˙iso(A,r,B,R) ≠ 0
proof -
  note A2
  moreover have ∀A ∈ FinPow(X). ∀B ∈ FinPow(Y).
    A ≈ 0 ∧ B ≈ 0 ⟶ ord˙iso(A,r,B,R) ≠ 0
    using eqpoll˙0˙is˙0 empty˙ord˙iso by blast
  moreover have ∀k ∈ nat.
```

```
  (∀A ∈ FinPow(X). ∀B ∈ FinPow(Y).
  A ≈ k ∧ B ≈ k ⟶ ord˙iso(A,r,B,R) ≠ 0) ⟶
  (∀C ∈ FinPow(X). ∀D ∈ FinPow(Y).
  C ≈ succ(k) ∧ D ≈ succ(k) ⟶ ord˙iso(C,r,D,R) ≠ 0)
 proof -
  { fix k assume k ∈ nat
    assume A3: ∀A ∈ FinPow(X). ∀B ∈ FinPow(Y).
A ≈ k ∧ B ≈ k ⟶ ord˙iso(A,r,B,R) ≠ 0
    have ∀C ∈ FinPow(X). ∀D ∈ FinPow(Y).
C ≈ succ(k) ∧ D ≈ succ(k) ⟶ ord˙iso(C,r,D,R) ≠ 0
    proof -
{ fix C assume C ∈ FinPow(X)
  fix D assume D ∈ FinPow(Y)
  assume C ≈ succ(k)  D ≈ succ(k)
  then have C ≠ 0 and D≠ 0
    using eqpoll˙succ˙imp˙not˙empty by auto
  let M_C = Maximum(r,C)
  let M_D = Maximum(R,D)
  let C_0 = C - –M_C″
  let D_0 = D - –M_D″
  from ‹C ∈ FinPow(X)› have C ⊆ X
    using FinPow˙def by simp
  with A1 have IsLinOrder(C,r)
    using ord˙linear˙subset by blast
  from ‹D ∈ FinPow(Y)› have D ⊆ Y
    using FinPow˙def by simp
  with A1 have IsLinOrder(D,R)
    using ord˙linear˙subset by blast
  from A1 ‹C ∈ FinPow(X)› ‹D ∈ FinPow(Y)›
    ‹C ≠ 0› ‹D≠ 0› have
    HasAmaximum(r,C) and HasAmaximum(R,D)
    using IsLinOrder˙def fin˙has˙max
    by auto
  with A1 have M_C ∈ C and M_D ∈ D
    using IsLinOrder˙def Order˙ZF˙4˙L3 by auto
  with ‹C ≈ succ(k)›  ‹D ≈ succ(k)› have
    C_0  ≈ k and D_0 ≈ k using Diff˙sing˙eqpoll by auto
  from ‹C ∈ FinPow(X)› ‹D ∈ FinPow(Y)›
  have C_0 ∈  FinPow(X) and D_0 ∈  FinPow(Y)
    using fin˙rem˙point˙fin by auto
  with A3 ‹C_0  ≈ k› ‹D_0 ≈ k› have
    ord˙iso(C_0,r,D_0,R) ≠ 0 by simp
  with ‹IsLinOrder(C,r)›  ‹IsLinOrder(D,R)›
    ‹HasAmaximum(r,C)› ‹HasAmaximum(R,D)›
  have ord˙iso(C,r,D,R) ≠ 0
    by (rule rem˙max˙ord˙iso)
} thus thesis by simp
    qed
   } thus thesis by blast
```

```
  qed
  ultimately show thesis by (rule ind˙on˙nat)
qed
```

Every two finite linearly ordered sets are order isomorphic.

```
lemma fin˙ord˙iso˙ex:
  assumes A1: IsLinOrder(X,r)  IsLinOrder(Y,R) and
  A2: A ∈ FinPow(X) B ∈ FinPow(Y) and A3: B ≈ A
  shows ord˙iso(A,r,B,R) ≠ 0
proof -
  from A2 obtain n where n ∈ nat and A ≈ n
    using finpow˙decomp by auto
  from  A3 ‹A ≈ n› have B ≈ n by (rule eqpoll˙trans)
  with A1 A2 ‹A ≈ n› ‹n ∈ nat› show ord˙iso(A,r,B,R) ≠ 0
    using fin˙order˙iso by simp
qed
```

Existence and uniqueness of order isomorphism for two linearly ordered sets with the same number of elements.

```
theorem fin˙ord˙iso˙ex˙uniq:
  assumes A1: IsLinOrder(X,r)  IsLinOrder(Y,R) and
  A2: A ∈ FinPow(X) B ∈ FinPow(Y) and A3: B ≈ A
  shows ∃!f. f ∈ ord˙iso(A,r,B,R)
proof
  from assms show ∃f. f ∈ ord˙iso(A,r,B,R)
    using fin˙ord˙iso˙ex by blast
  fix f g
  assume A4: f ∈ ord˙iso(A,r,B,R)  g ∈ ord˙iso(A,r,B,R)
  then have converse(g) ∈ ord˙iso(B,R,A,r)
    using ord˙iso˙sym by simp
  with ‹f ∈ ord˙iso(A,r,B,R)› have
    I: converse(g) O f ∈  ord˙iso(A,r,A,r)
    by (rule ord˙iso˙trans)
  { assume A ≠ 0
    with A1 A2 I have converse(g) O f = id(A)
      using fin˙ord˙auto˙id by auto
    with A4 have f = g
      using ord˙iso˙def comp˙inv˙id˙eq˙bij by auto }
  moreover
  { assume A = 0
    then have A ≈ 0 using eqpoll˙0˙iff
      by simp
    with A3 have B ≈ 0 by (rule eqpoll˙trans)
    with A4 ‹A = 0› have
      f ∈ ord˙iso(0,r,0,R) and  g ∈ ord˙iso(0,r,0,R)
      using eqpoll˙0˙iff by auto
    then have f = g by (rule empty˙ord˙iso˙uniq) }
  ultimately show f = g
    using ord˙iso˙def comp˙inv˙id˙eq˙bij
```

by auto
qed

end

17 Equivalence relations

theory EquivClass1 imports ZF.EquivClass func˙ZF ZF1

begin

In this theory file we extend the work on equivalence relations done in the standard Isabelle's EquivClass theory. That development is very good and all, but we really would prefer an approach contained within the a standard ZF set theory, without extensions specific to Isabelle. That is why this theory is written.

17.1 Congruent functions and projections on the quotient

Suppose we have a set X with a relation $r \subseteq X \times X$ and a function $f : X \to X$. The function f can be compatible (congruent) with r in the sense that if two elements x, y are related then the values $f(x), f(x)$ are also related. This is especially useful if r is an equivalence relation as it allows to "project" the function to the quotient space X/r (the set of equivalence classes of r) and create a new function F that satifies the formula $F([x]_r) = [f(x)]_r$. When f is congruent with respect to r such definition of the value of F on the equivalence class $[x]_r$ does not depend on which x we choose to represent the class. In this section we also consider binary operations that are congruent with respect to a relation. These are important in algebra - the congruency condition allows to project the operation to obtain the operation on the quotient space.

First we define the notion of function that maps equivalent elements to equivalent values. We use similar names as in the Isabelle's standard EquivClass theory to indicate the conceptual correspondence of the notions.

definition
Congruent(r,f) ≡
(∀x y. ⟨x,y⟩ ∈ r ⟶ ⟨f(x),f(y)⟩ ∈ r)

Now we will define the projection of a function onto the quotient space. In standard math the equivalence class of x with respect to relation r is usually denoted $[x]_r$. Here we reuse notation $r\{x\}$ instead. This means the image of the set $\{x\}$ with respect to the relation, which, for equivalence relations is exactly its equivalence class if you think about it.

```
definition
  ProjFun(A,r,f) ≡
  –⟨c,⋃x∈c. r–f(x)″⟩. c ∈ (A//r)″
```

Elements of equivalence classes belong to the set.

```
lemma EquivClass˙1˙L1:
  assumes A1: equiv(A,r) and A2: C ∈ A//r and A3: x∈C
  shows x∈A
proof -
  from A2 have C ⊆ ⋃ (A//r) by auto
  with A1 A3 show x∈A
    using Union˙quotient by auto
qed
```

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

```
lemma EquivClass˙1˙L1A:
  assumes A⊆X shows –r–x″. x∈A″ ⊆ X//r
  using assms quotientI by auto
```

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

```
lemma EquivClass˙1˙L2:
  assumes A1: equiv(A,r)  C ∈ A//r and A2: x∈C
  shows r–x″ = C
proof -
  from A1 A2 have x ∈ r–x″
    using EquivClass˙1˙L1  equiv˙class˙self by simp
  with A2 have I: r–x″∩C ≠ 0 by auto
  from A1 A2 have r–x″ ∈ A//r
    using EquivClass˙1˙L1 quotientI by simp
  with A1 I show thesis
    using quotient˙disj by blast
qed
```

Elements that belong to the same equivalence class are equivalent.

```
lemma EquivClass˙1˙L2A:
  assumes equiv(A,r)  C ∈ A//r  x∈C  y∈C
  shows ⟨x,y⟩ ∈ r
  using assms EquivClass˙1˙L2 EquivClass˙1˙L1 equiv˙class˙eq˙iff
  by simp
```

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

```
lemma EquivClass˙1˙L2B:
  assumes A1: equiv(A,r) and A2: y∈A and A3: x ∈ r–y″
  shows ⟨x,y⟩ ∈ r
proof -
  from A2 have  r–y″ ∈ A//r
    using quotientI by simp
```

```
  with A1 A3 show thesis using
    EquivClass˙1˙L1 equiv˙class˙self equiv˙class˙nondisjoint by blast
qed
```

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

```
lemma EquivClass˙1˙L3:
  assumes A1: equiv(A,r) and A2: Congruent(r,f)
  and A3: C ∈ A//r  x∈C  y∈C
  shows r–f(x)″ = r–f(y)″
proof -
  from A1 A3 have ⟨x,y⟩ ∈ r
    using EquivClass˙1˙L2A by simp
  with A2 have  ⟨f(x),f(y)⟩ ∈ r
    using Congruent˙def by simp
  with A1 show thesis using equiv˙class˙eq by simp
qed
```

The values of congruent functions are in the space.

```
lemma EquivClass˙1˙L4:
  assumes A1: equiv(A,r) and A2: C ∈ A//r  x∈C
  and A3: Congruent(r,f)
  shows f(x) ∈ A
proof -
  from A1 A2 have x∈A
    using EquivClass˙1˙L1 by simp
  with A1 have ⟨x,x⟩ ∈ r
    using equiv˙def refl˙def by simp
  with A3 have  ⟨f(x),f(x)⟩ ∈ r
    using Congruent˙def by simp
  with A1 show thesis using equiv˙type by auto
qed
```

Equivalence classes are not empty.

```
lemma EquivClass˙1˙L5:
  assumes A1: refl(A,r) and A2: C ∈ A//r
  shows C≠0
proof -
  from A2 obtain x where I: C = r–x″ and x∈A
    using quotient˙def by auto
  from A1 ‹x∈A› have x ∈ r–x″ using refl˙def by auto
  with I show thesis by auto
qed
```

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

```
lemma EquivClass˙1˙L6:
```

```
  assumes A1: equiv(A,r) and A2: Congruent(r,f)
  and A3: C ∈ A//r
  shows (⋃x∈C. r–f(x)″) ∈ A//r
proof -
  from A1 have refl(A,r) unfolding equiv˙def by simp
  with A3 have C≠0 using EquivClass˙1˙L5 by simp
  moreover from A2 A3 A1 have ∀x∈C. r–f(x)″ ∈ A//r
    using EquivClass˙1˙L4 quotientI by auto
  moreover from A1 A2 A3 have
    ∀x y. x∈C ∧ y∈C ⟶ r–f(x)″ = r–f(y)″
    using EquivClass˙1˙L3 by blast
  ultimately show thesis by (rule ZF1˙1˙L2)
qed
```

Congruent functions can be projected.

```
lemma EquivClass˙1˙T0:
  assumes equiv(A,r)  Congruent(r,f)
  shows ProjFun(A,r,f) : A//r → A//r
  using assms EquivClass˙1˙L6 ProjFun˙def ZF˙fun˙from˙total
  by simp
```

We now define congruent functions of two variables (binary funtions). The predicate Congruent2 corresponds to congruent2 in Isabelle's standard EquivClass theory, but uses ZF-functions rather than meta-functions.

```
definition
  Congruent2(r,f) ≡
  (∀x₁ x₂ y₁ y₂. ⟨x₁,x₂⟩ ∈ r ∧ ⟨y₁,y₂⟩ ∈ r  ⟶
  ⟨f⟨x₁,y₁⟩, f⟨x₂,y₂⟩ ⟩ ∈ r)
```

Next we define the notion of projecting a binary operation to the quotient space. This is a very important concept that allows to define quotient groups, among other things.

```
definition
  ProjFun2(A,r,f) ≡
  –⟨p,⋃ z ∈ fst(p)×snd(p). r–f(z)″⟩. p ∈ (A//r)×(A//r) ″
```

The following lemma is a two-variables equivalent of EquivClass˙1˙L3.

```
lemma EquivClass˙1˙L7:
  assumes A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: C₁ ∈ A//r  C₂ ∈ A//r
  and A4: z₁ ∈ C₁×C₂  z₂ ∈ C₁×C₂
  shows r–f(z₁)″ = r–f(z₂)″
proof -
  from A4 obtain x₁ y₁ x₂ y₂ where
    x₁∈C₁ and y₁∈C₂ and z₁ = ⟨x₁,y₁⟩ and
    x₂∈C₁ and y₂∈C₂ and z₂ = ⟨x₂,y₂⟩
    by auto
  with A1 A3 have ⟨x₁,x₂⟩ ∈ r and ⟨y₁,y₂⟩ ∈ r
```

using EquivClass˙1˙L2A by auto
with A2 have ⟨f⟨x_1,y_1⟩,f⟨x_2,y_2⟩⟩ ∈ r
using Congruent2˙def by simp
with A1 ‹z_1 = ⟨x_1,y_1⟩› ‹z_2 = ⟨x_2,y_2⟩› 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: C_1 ∈ A//r and A3: C_2 ∈ A//r
and A4: z ∈ C_1×C_2 and A5: Congruent2(r,f)
shows f(z) ∈ A
proof -
from A4 obtain x y where x∈C_1 and y∈C_2 and z = ⟨x,y⟩
by auto
with A1 A2 A3 have x∈A and y∈A
using EquivClass˙1˙L1 by auto
with A1 A4 have ⟨x,x⟩ ∈ r and ⟨y,y⟩ ∈ r
using equiv˙def refl˙def by auto
with A5 have ⟨f⟨x,y⟩, f⟨x,y⟩ ⟩ ∈ r
using Congruent2˙def by simp
with A1 ‹z = ⟨x,y⟩› show thesis using equiv˙type by auto
qed

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

lemma EquivClass˙1˙L8A:
assumes A1: equiv(A,r) and A2: x∈A y∈A
and A3: Congruent2(r,f)
shows f⟨x,y⟩ ∈ A
proof -
from A1 A2 have r–x″ ∈ A//r r–y″ ∈ A//r
⟨x,y⟩ ∈ r–x″×r–y″
using equiv˙class˙self quotientI by auto
with A1 A3 show thesis using EquivClass˙1˙L8 by simp
qed

The following lemma is a two-variables equivalent of EquivClass˙1˙L6.

lemma EquivClass˙1˙L9:
assumes A1: equiv(A,r) and A2: Congruent2(r,f)
and A3: p ∈ (A//r)×(A//r)
shows (⋃ z ∈ fst(p)×snd(p). r–f(z)″) ∈ A//r
proof -
from A3 have fst(p) ∈ A//r and snd(p) ∈ A//r
by auto
with A1 A2 have
I: ∀z ∈ fst(p)×snd(p). f(z) ∈ A

```
    using EquivClass˙1˙L8 by simp
  from A3 A1 have fst(p)×snd(p) ≠ 0
    using equiv˙def EquivClass˙1˙L5 Sigma˙empty˙iff
    by auto
  moreover from A1 I have
    ∀z ∈ fst(p)×snd(p). r–f(z)″ ∈ A//r
    using quotientI by simp
  moreover from A1 A2 ‹fst(p) ∈ A//r› ‹snd(p) ∈ A//r› have
    ∀z₁ z₂. z₁ ∈ fst(p)×snd(p) ∧ z₂ ∈ fst(p)×snd(p) ⟶
    r–f(z₁)″ = r–f(z₂)″
    using EquivClass˙1˙L7 by blast
   ultimately show thesis by (rule ZF1˙1˙L2)
qed
```

Congruent functions of two variables can be projected.

```
theorem EquivClass˙1˙T1:
  assumes equiv(A,r)  Congruent2(r,f)
  shows ProjFun2(A,r,f) : (A//r)×(A//r) → A//r
  using assms EquivClass˙1˙L9 ProjFun2˙def ZF˙fun˙from˙total
  by simp
```

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

```
lemma EquivClass˙1˙L10:
  assumes A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: x∈A  y∈A
  shows ProjFun2(A,r,f)⟨r–x″,r–y″⟩ = r–f⟨x,y⟩″
proof -
  from A3 A1 have r–x″ × r–y″ ≠ 0
    using quotientI equiv˙def EquivClass˙1˙L5 Sigma˙empty˙iff
    by auto
  moreover have
    ∀z ∈ r–x″×r–y″.  r–f(z)″ = r–f⟨x,y⟩″
  proof
    fix z assume A4: z ∈ r–x″×r–y″
    from A1 A3 have
      r–x″ ∈ A//r r–y″ ∈ A//r
      ⟨x,y⟩ ∈ r–x″×r–y″
      using quotientI equiv˙class˙self by auto
    with A1 A2 A4 show
      r–f(z)″ = r–f⟨x,y⟩″
      using EquivClass˙1˙L7 by blast
  qed
  ultimately have
    (⋃z ∈ r–x″×r–y″. r–f(z)″) =  r–f⟨x,y⟩″
    by (rule ZF1˙1˙L1)
  moreover have
    ProjFun2(A,r,f)⟨r–x″,r–y″⟩ = (⋃z ∈ r–x″×r–y″. r–f(z)″)
    proof -
```

```
    from assms have
 ProjFun2(A,r,f) : (A//r)×(A//r) → A//r
 ⟨r–x″,r–y″⟩ ∈ (A//r)×(A//r)
 using EquivClass˙1˙T1 quotientI by auto
    then show thesis using ProjFun2˙def ZF˙fun˙from˙tot˙val
 by auto
   qed
  ultimately show thesis by simp
qed
```

17.2 Projecting commutative, associative and distributive operations.

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

The projection of commutative operation is commutative.

```
lemma EquivClass˙2˙L1: assumes
  A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: f –is commutative on″ A
  and A4: c1 ∈ A//r  c2 ∈ A//r
  shows ProjFun2(A,r,f)⟨c1,c2⟩ = ProjFun2(A,r,f)⟨c2,c1⟩
proof -
  from A4 obtain x y where D1:
    c1 = r–x″  c2 = r–y″
    x∈A  y∈A
    using quotient˙def by auto
  with A1 A2 have ProjFun2(A,r,f)⟨c1,c2⟩ = r–f⟨x,y⟩″
    using EquivClass˙1˙L10 by simp
  also from A3 D1 have
    r–f⟨x,y⟩″ = r–f⟨y,x⟩″
    using IsCommutative˙def by simp
  also from A1 A2 D1 have
    r–f⟨y,x⟩″ = ProjFun2(A,r,f) ⟨c2,c1⟩
    using EquivClass˙1˙L10 by simp
  finally show thesis by simp
qed
```

The projection of commutative operation is commutative.

```
theorem EquivClass˙2˙T1:
  assumes equiv(A,r) and Congruent2(r,f)
  and f –is commutative on″ A
  shows ProjFun2(A,r,f) –is commutative on″ A//r
  using assms IsCommutative˙def EquivClass˙2˙L1 by simp
```

The projection of an associative operation is associative.

```
lemma EquivClass˙2˙L2:
  assumes A1: equiv(A,r) and A2: Congruent2(r,f)
```

```
  and A3: f –is associative on″ A
  and A4: c1 ∈ A//r  c2 ∈ A//r  c3 ∈ A//r
  and A5: g = ProjFun2(A,r,f)
  shows g⟨g⟨c1,c2⟩,c3⟩ = g⟨c1,g⟨c2,c3⟩⟩
proof -
  from A4 obtain x y z where D1:
    c1 = r–x″  c2 = r–y″  c3 = r–z″
    x∈A  y∈A  z∈A
    using quotient˙def by auto
  with A3 have T1:f⟨x,y⟩ ∈ A  f⟨y,z⟩ ∈ A
    using IsAssociative˙def apply˙type by auto
  with A1 A2 D1 A5 have
    g⟨g⟨c1,c2⟩,c3⟩ =  r–f⟨f⟨x,y⟩,z⟩″
    using EquivClass˙1˙L10 by simp
  also from D1 A3 have
    ... = r–f⟨x,f⟨y,z⟩ ⟩″
    using IsAssociative˙def by simp
  also from T1 A1 A2 D1 A5 have
    ... = g⟨c1,g⟨c2,c3⟩⟩
    using EquivClass˙1˙L10 by simp
  finally show thesis by simp
qed
```

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

```
theorem EquivClass˙2˙T2:
  assumes A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: f –is associative on″ A
  shows ProjFun2(A,r,f) –is associative on″ A//r
proof -
  let g = ProjFun2(A,r,f)
  from A1 A2 have
    g ∈ (A//r)×(A//r) → A//r
    using EquivClass˙1˙T1 by simp
  moreover from A1 A2 A3 have
    ∀c1 ∈ A//r.∀c2 ∈ A//r.∀c3 ∈ A//r.
    g⟨g⟨c1,c2⟩,c3⟩ = g⟨c1,g⟨c2,c3⟩⟩
    using EquivClass˙2˙L2 by simp
  ultimately show thesis
    using IsAssociative˙def by simp
qed
```

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

```
lemma EquivClass˙2˙L3:
  assumes A1: IsDistributive(X,A,M)
  and A2: equiv(X,r)
  and A3: Congruent2(r,A) Congruent2(r,M)
  and A4: a ∈ X//r  b ∈ X//r  c ∈ X//r
```

```
  and A5: A_p = ProjFun2(X,r,A) M_p = ProjFun2(X,r,M)
  shows M_p⟨a,A_p⟨b,c⟩⟩ = A_p⟨ M_p⟨a,b⟩,M_p⟨a,c⟩⟩ ∧
  M_p⟨ A_p⟨b,c⟩,a ⟩ = A_p⟨ M_p⟨b,a⟩, M_p⟨c,a⟩⟩
proof
  from A4 obtain x y z where x∈X  y∈X  z∈X
    a = r–x″  b = r–y″  c = r–z″
    using quotient˙def by auto
  with A1 A2 A3 A5 show
    M_p⟨a,A_p⟨b,c⟩⟩ = A_p⟨ M_p⟨a,b⟩,M_p⟨a,c⟩⟩ and
    M_p⟨ A_p⟨b,c⟩,a ⟩ = A_p⟨ M_p⟨b,a⟩, M_p⟨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 A_p = ProjFun2(X,r,A)
 let M_p = ProjFun2(X,r,M)
 from A1 A2 A3 have
   ∀a∈X//r.∀b∈X//r.∀c∈X//r.
   M_p⟨a,A_p⟨b,c⟩⟩ = A_p⟨M_p⟨a,b⟩,M_p⟨a,c⟩⟩ ∧
   M_p⟨A_p⟨b,c⟩,a⟩ = A_p⟨M_p⟨b,a⟩,M_p⟨c,a⟩⟩
   using EquivClass˙2˙L3 by simp
 then show thesis using IsDistributive˙def by simp
qed
```

17.3 Saturated sets

In this section we consider sets that are saturated with respect to an equivalence relation. A set A is saturated with respect to a relation r if $A = r^{-1}(r(A))$. For equivalence relations saturated sets are unions of equivalence classes. This makes them useful as a tool to define subsets of the quotient space using properties of representants. Namely, we often define a set $B \subseteq X/r$ by saying that $[x]_r \in B$ iff $x \in A$. If A is a saturated set, this definition is consistent in the sense that it does not depend on the choice of x to represent $[x]_r$.

The following defines the notion of a saturated set. Recall that in Isabelle r-(A) is the inverse image of A with respect to relation r. This definition is not specific to equivalence relations.

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

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

```
lemma EquivClass˙3˙L1: assumes A1: equiv(X,r)
  shows IsSaturated(r,A) ⟷ A = r(A)
proof
  assume IsSaturated(r,A)
  then have A = (converse(r) O r)(A)
    using IsSaturated˙def vimage˙def image˙comp
    by simp
  also from A1 have ... = r(A)
    using equiv˙comp˙eq by simp
  finally show A = r(A) by simp
next assume A = r(A)
  with A1 have A = (converse(r) O r)(A)
    using equiv˙comp˙eq by simp
  also have ... =  r-(r(A))
    using vimage˙def image˙comp by simp
  finally have A =  r-(r(A)) by simp
  then show IsSaturated(r,A) using IsSaturated˙def
    by simp
qed
```

For equivalence relations sets are contained in their images.

```
lemma EquivClass˙3˙L2: assumes A1: equiv(X,r) and A2: A⊆X
  shows A ⊆ r(A)
proof
  fix a assume a∈A
  with A1 A2 have a ∈ r–a″
    using equiv˙class˙self by auto
  with ‹a∈A› show a ∈ r(A) by auto
qed
```

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

```
lemma EquivClass˙3˙L3: assumes A1: equiv(X,r)
  and A2: r ⊆ X×X and A3: A⊆X
  and A4: ∀x∈A. ∀y∈X. ⟨x,y⟩ ∈ r ⟶ y∈A
  shows IsSaturated(r,A)
proof -
  from A2 A4 have r(A) ⊆ A
    using image˙iff by blast
  moreover from A1 A3 have A ⊆ r(A)
    using EquivClass˙3˙L2 by simp
  ultimately have A = r(A) by auto
  with A1 show IsSaturated(r,A) using EquivClass˙3˙L1
    by simp
qed
```

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

only one direction.

```
lemma EquivClass˙3˙L4: assumes A1: equiv(X,r)
  and A2: IsSaturated(r,A) and A3: A⊆X
  and A4: ⟨x,y⟩ ∈ r
  and A5: x∈X  y∈A
  shows x∈A
proof -
  from A1 A5 have x ∈ r–x″
    using equiv˙class˙self by simp
  with A1 A3 A4 A5 have x ∈ r(A)
    using equiv˙class˙eq equiv˙class˙self
    by auto
  with A1 A2 show x∈A
    using EquivClass˙3˙L1 by simp
qed
```

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

```
lemma EquivClass˙3˙L5: assumes A1: equiv(X,r)
  and A2: IsSaturated(r,A) and A3: A⊆X
  and A4: x∈X  y∈X
  and A5: ⟨x,y⟩ ∈ r
  shows x∈A ⟷ y∈A
proof
  assume y∈A
  with assms show x∈A using EquivClass˙3˙L4
    by simp
next assume x∈A
  from A1 A5 have ⟨y,x⟩ ∈ r
    using equiv˙is˙sym by blast
  with A1 A2 A3 A4 ‹x∈A› show y∈A
    using EquivClass˙3˙L4 by simp
qed
```

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

```
lemma EquivClass˙3˙L6: assumes A1: equiv(X,r)
  and A2: IsSaturated(r,A) and A3: A⊆X and A4: x∈X
  and A5: B = –r–x″. x∈A″
  shows x∈A ⟷ r–x″ ∈ B
proof
  assume x∈A
  with A5 show r–x″ ∈ B by auto
next assume r–x″ ∈ B
  with A5 obtain y where y ∈ A and r–x″ = r–y″
    by auto
  with A1 A3 have ⟨x,y⟩ ∈ r
    using eq˙equiv˙class by auto
  with A1 A2 A3 A4  ‹y ∈ A› show x∈A
    using EquivClass˙3˙L4 by simp
qed
```

A technical lemma involving a projection of a saturated set and a logical epression with exclusive or. Note that we don't really care what Xor is here, this is true for any predicate.

lemma EquivClass˙3˙L7: assumes equiv(X,r)
  and IsSaturated(r,A) and A⊆X
  and x∈X y∈X
  and B = –r–x″. x∈A″
  and (x∈A) Xor (y∈A)
  shows (r–x″ ∈ B) Xor (r–y″ ∈ B)
  using assms EquivClass˙3˙L6 by simp

end

18 Finite sequences

theory FiniteSeq˙ZF imports Nat˙ZF˙IML func1

begin

This theory treats finite sequences (i.e. maps $n \to X$, where $n = \{0, 1, .., n-1\}$ is a natural number) as lists. It defines and proves the properties of basic operations on lists: concatenation, appending and element etc.

18.1 Lists as finite sequences

A natural way of representing (finite) lists in set theory is through (finite) sequences. In such view a list of elements of a set X is a function that maps the set $\{0, 1, ..n-1\}$ into X. Since natural numbers in set theory are defined so that $n = \{0, 1, ..n-1\}$, a list of length n can be understood as an element of the function space $n \to X$.

We define the set of lists with values in set X as Lists(X).

definition
  Lists(X) ≡ ⋃n∈nat.(n→X)

The set of nonempty X-value listst will be called NELists(X).

definition
  NELists(X) ≡ ⋃n∈nat.(succ(n)→X)

We first define the shift that moves the second sequence to the domain $\{n, .., n + k - 1\}$, where n, k are the lengths of the first and the second sequence, resp. To understand the notation in the definitions below recall that in Isabelle/ZF pred(n) is the previous natural number and denotes the difference between natural numbers n and k.

definition

ShiftedSeq(b,n) ≡ −⟨j, b(j #- n)⟩. j ∈ NatInterval(n,domain(b))″

We define concatenation of two sequences as the union of the first sequence with the shifted second sequence. The result of concatenating lists a and b is called Concat(a,b).

definition
Concat(a,b) ≡ a ∪ ShiftedSeq(b,domain(a))

For a finite sequence we define the sequence of all elements except the first one. This corresponds to the "tail" function in Haskell. We call it Tail here as well.

definition
Tail(a) ≡ −⟨k, a(succ(k))⟩. k ∈ pred(domain(a))″

A dual notion to Tail is the list of all elements of a list except the last one. Borrowing the terminology from Haskell again, we will call this Init.

definition
Init(a) ≡ restrict(a,pred(domain(a)))

Another obvious operation we can talk about is appending an element at the end of a sequence. This is called Append.

definition
Append(a,x) ≡ a ∪ −⟨domain(a),x⟩″

If lists are modeled as finite sequences (i.e. functions on natural intervals $\{0, 1, .., n-1\} = n$) it is easy to get the first element of a list as the value of the sequence at 0. The last element is the value at $n-1$. To hide this behind a familiar name we define the Last element of a list.

definition
Last(a) ≡ a(pred(domain(a)))

Shifted sequence is a function on a the interval of natural numbers.

```
lemma shifted˙seq˙props:
  assumes A1: n ∈ nat  k ∈ nat and A2: b:k→X
  shows
  ShiftedSeq(b,n): NatInterval(n,k) → X
  ∀i ∈ NatInterval(n,k). ShiftedSeq(b,n)(i) = b(i #- n)
  ∀j∈k. ShiftedSeq(b,n)(n #+ j) = b(j)
proof -
  let I = NatInterval(n,domain(b))
  from A2 have Fact: I = NatInterval(n,k) using func1˙1˙L1 by simp
  with A1 A2 have ∀j∈ I. b(j #- n) ∈ X
    using inter˙diff˙in˙len apply˙funtype by simp
  then have
    −⟨j, b(j #- n)⟩. j ∈ I″ : I → X by (rule ZF˙fun˙from˙total)
  with Fact show thesis˙1: ShiftedSeq(b,n): NatInterval(n,k) → X
    using ShiftedSeq˙def by simp
```

```
  { fix i
    from Fact thesis˙1 have  ShiftedSeq(b,n): I → X by simp
    moreover
    assume i ∈ NatInterval(n,k)
    with Fact have i ∈ I by simp
    moreover from Fact have
      ShiftedSeq(b,n) = −⟨i, b(i #- n)⟩. i ∈ I″
      using ShiftedSeq˙def by simp
    ultimately have ShiftedSeq(b,n)(i) =  b(i #- n)
      by (rule ZF˙fun˙from˙tot˙val)
  } then show thesis1:
      ∀i ∈ NatInterval(n,k). ShiftedSeq(b,n)(i) = b(i #- n)
    by simp
  { fix j
    let i = n #+ j
    assume A3: j∈k
    with A1 have j ∈ nat using elem˙nat˙is˙nat by blast
    then have i #- n = j using diff˙add˙inverse by simp
    with A3 thesis1 have ShiftedSeq(b,n)(i) = b(j)
      using NatInterval˙def by auto
  } then show ∀j∈k. ShiftedSeq(b,n)(n #+ j) = b(j)
    by simp
qed
```

Basis properties of the contatenation of two finite sequences.

```
theorem concat˙props:
  assumes A1: n ∈ nat  k ∈ nat and A2: a:n→X   b:k→X
  shows
  Concat(a,b): n #+ k → X
  ∀i∈n. Concat(a,b)(i) = a(i)
  ∀i ∈ NatInterval(n,k). Concat(a,b)(i) =  b(i #- n)
  ∀j ∈ k. Concat(a,b)(n #+ j) = b(j)
proof -
  from A1 A2 have
    a:n→X  and I: ShiftedSeq(b,n): NatInterval(n,k) → X
    and n ∩ NatInterval(n,k) = 0
    using shifted˙seq˙props length˙start˙decomp by auto
  then have
    a ∪ ShiftedSeq(b,n): n ∪ NatInterval(n,k) → X ∪ X
    by (rule fun˙disjoint˙Un)
  with A1 A2 show Concat(a,b): n #+ k → X
    using func1˙1˙L1 Concat˙def length˙start˙decomp by auto
  { fix i assume i ∈ n
    with A1 I have i ∉ domain(ShiftedSeq(b,n))
      using length˙start˙decomp func1˙1˙L1 by auto
    with A2 have Concat(a,b)(i) = a(i)
      using func1˙1˙L1 fun˙disjoint˙apply1 Concat˙def by simp
  } thus ∀i∈n. Concat(a,b)(i) = a(i) by simp
  { fix i assume A3: i ∈ NatInterval(n,k)
```

```
    with A1 A2 have i ∉ domain(a)
      using length˙start˙decomp func1˙1˙L1 by auto
    with A1 A2 A3 have Concat(a,b)(i) =  b(i #- n)
      using func1˙1˙L1 fun˙disjoint˙apply2 Concat˙def shifted˙seq˙props
      by simp
  } thus II: ∀i ∈ NatInterval(n,k). Concat(a,b)(i) =  b(i #- n)
    by simp
  { fix j
    let i = n #+ j
    assume A3: j∈k
    with A1 have j ∈ nat using elem˙nat˙is˙nat by blast
    then have i #- n = j using diff˙add˙inverse by simp
     with A3 II have Concat(a,b)(i) = b(j)
      using NatInterval˙def by auto
  } thus ∀j ∈ k. Concat(a,b)(n #+ j) = b(j)
    by simp
qed
```

Properties of concatenating three lists.

```
lemma concat˙concat˙list:
  assumes A1: n ∈ nat  k ∈ nat  m ∈ nat and
  A2: a:n→X   b:k→X  c:m→X and
  A3: d = Concat(Concat(a,b),c)
  shows
  d : n #+k #+ m → X
  ∀j ∈ n. d(j) = a(j)
  ∀j ∈ k. d(n #+ j) = b(j)
  ∀j ∈ m. d(n #+ k #+ j) = c(j)
proof -
  from A1 A2 have I:
    n #+ k ∈ nat   m ∈ nat
    Concat(a,b): n #+ k → X   c:m→X
    using concat˙props by auto
  with A3 show d: n #+k #+ m → X
    using concat˙props by simp
  from I have II: ∀i ∈ n #+ k.
    Concat(Concat(a,b),c)(i) = Concat(a,b)(i)
    by (rule concat˙props)
  { fix j assume A4: j ∈ n
    moreover from A1 have n ⊆ n #+ k using add˙nat˙le by simp
    ultimately have j ∈ n #+ k by auto
    with A3 II have d(j) =  Concat(a,b)(j) by simp
    with A1 A2 A4 have d(j) = a(j)
      using concat˙props by simp
  } thus ∀j ∈ n. d(j) = a(j) by simp
  { fix j assume A5: j ∈ k
    with A1 A3 II have d(n #+ j) = Concat(a,b)(n #+ j)
      using add˙lt˙mono by simp
    also from A1 A2 A5 have ... = b(j)
```

```
    using concat˙props by simp
  finally have d(n #+ j) = b(j) by simp
} thus ∀j ∈ k. d(n #+ j) = b(j) by simp
from I have ∀j ∈ m. Concat(Concat(a,b),c)(n #+ k #+ j) = c(j)
  by (rule concat˙props)
with A3 show ∀j ∈ m. d(n #+ k #+ j) = c(j)
  by simp
qed
```

Properties of concatenating a list with a concatenation of two other lists.

```
lemma concat˙list˙concat:
  assumes A1: n ∈ nat  k ∈ nat  m ∈ nat and
  A2: a:n→X   b:k→X  c:m→X and
  A3: e = Concat(a, Concat(b,c))
  shows
  e : n #+k #+ m → X
  ∀j ∈ n. e(j) = a(j)
  ∀j ∈ k. e(n #+ j) = b(j)
  ∀j ∈ m. e(n #+ k #+ j) = c(j)
proof -
  from A1 A2 have I:
    n ∈ nat  k #+ m ∈ nat
    a:n→X  Concat(b,c): k #+ m → X
    using concat˙props by auto
  with A3 show  e : n #+k #+ m → X
    using concat˙props add˙assoc by simp
  from I have ∀j ∈ n. Concat(a, Concat(b,c))(j) = a(j)
    by (rule concat˙props)
  with A3 show ∀j ∈ n. e(j) = a(j) by simp
  from I have II:
    ∀j ∈ k #+ m. Concat(a, Concat(b,c))(n #+ j) = Concat(b,c)(j)
    by (rule concat˙props)
  { fix j assume A4: j ∈ k
    moreover from A1 have k ⊆ k #+ m using add˙nat˙le by simp
    ultimately have j ∈ k #+ m by auto
    with A3 II have e(n #+ j) =  Concat(b,c)(j) by simp
    also from A1 A2 A4 have ... = b(j)
      using concat˙props by simp
    finally have e(n #+ j) = b(j) by simp
  } thus ∀j ∈ k. e(n #+ j) = b(j) by simp
  { fix j assume A5: j ∈ m
    with A1 II A3 have e(n #+ k #+ j) = Concat(b,c)(k #+ j)
      using add˙lt˙mono add˙assoc by simp
    also from A1 A2 A5 have ... = c(j)
      using concat˙props by simp
    finally have e(n #+ k #+ j) = c(j) by simp
  } then show ∀j ∈ m. e(n #+ k #+ j) = c(j)
    by simp
qed
```

Concatenation is associative.

```
theorem concat˙assoc:
  assumes A1: n ∈ nat  k ∈ nat  m ∈ nat and
  A2: a:n→X   b:k→X   c:m→X
  shows Concat(Concat(a,b),c) =  Concat(a, Concat(b,c))
proof -
  let d = Concat(Concat(a,b),c)
  let e = Concat(a, Concat(b,c))
  from A1 A2 have
    d : n #+k #+ m → X and e : n #+k #+ m → X
    using concat˙concat˙list concat˙list˙concat by auto
  moreover have ∀i ∈  n #+k #+ m. d(i) = e(i)
  proof -
    { fix i assume i ∈ n #+k #+ m
      moreover from A1 have
 n #+k #+ m = n ∪ NatInterval(n,k) ∪ NatInterval(n #+ k,m)
 using adjacent˙intervals3 by simp
      ultimately have
 i ∈ n ∨ i ∈ NatInterval(n,k) ∨ i ∈ NatInterval(n #+ k,m)
 by simp
      moreover
      { assume i ∈ n
 with A1 A2 have d(i) = e(i)
 using concat˙concat˙list concat˙list˙concat by simp }
      moreover
      { assume i ∈ NatInterval(n,k)
 then obtain j where j∈k and i = n #+ j
   using NatInterval˙def by auto
 with A1 A2 have d(i) = e(i)
   using concat˙concat˙list concat˙list˙concat by simp }
      moreover
      { assume i ∈ NatInterval(n #+ k,m)
 then obtain j where j ∈ m and i = n #+ k #+ j
   using NatInterval˙def by auto
 with A1 A2 have d(i) = e(i)
   using concat˙concat˙list concat˙list˙concat by simp }
      ultimately have d(i) = e(i) by auto
    } thus thesis by simp
  qed
  ultimately show d = e by (rule func˙eq)
qed
```

Properties of Tail.

```
theorem tail˙props:
  assumes A1: n ∈ nat and A2: a: succ(n) → X
  shows
  Tail(a) : n → X
  ∀k ∈ n. Tail(a)(k) = a(succ(k))
proof -
```

```
  from A1 A2 have ∀k ∈ n. a(succ(k)) ∈ X
    using succ˙ineq apply˙funtype by simp
  then have −⟨k, a(succ(k))⟩. k ∈ n″ : n → X
    by (rule ZF˙fun˙from˙total)
  with A2 show I: Tail(a) : n → X
    using func1˙1˙L1 pred˙succ˙eq Tail˙def by simp
  moreover from A2 have Tail(a) = −⟨k, a(succ(k))⟩. k ∈ n″
    using func1˙1˙L1 pred˙succ˙eq Tail˙def by simp
  ultimately show ∀k ∈ n. Tail(a)(k) = a(succ(k))
    by (rule ZF˙fun˙from˙tot˙val0)
qed
```

Properties of Append. It is a bit surprising that the we don't need to assume that n is a natural number.

```
theorem append˙props:
  assumes A1: a: n → X and A2: x∈X and A3: b = Append(a,x)
  shows
  b : succ(n) → X
  ∀k∈n. b(k) = a(k)
  b(n) = x
proof -
  note A1
  moreover have I: n ∉ n using mem˙not˙refl by simp
  moreover from A1 A3 have II: b = a ∪ −⟨n,x⟩″
    using func1˙1˙L1 Append˙def by simp
  ultimately have b : n ∪ −n″ → X ∪ −x″
    by (rule func1˙1˙L11D)
  with A2 show b : succ(n) → X
    using succ˙explained set˙elem˙add by simp
  from A1 I II show ∀k∈n. b(k) = a(k) and b(n) = x
    using func1˙1˙L11D by auto
qed
```

A special case of append˙props: appending to a nonempty list does not change the head (first element) of the list.

```
corollary head˙of˙append:
  assumes n∈ nat and a: succ(n) → X and x∈X
  shows Append(a,x)(0) = a(0)
  using assms append˙props empty˙in˙every˙succ by auto
```

Tail commutes with Append.

```
theorem tail˙append˙commute:
  assumes A1: n ∈ nat and A2: a: succ(n) → X and A3: x∈X
  shows Append(Tail(a),x) = Tail(Append(a,x))
proof -
  let b = Append(Tail(a),x)
  let c = Tail(Append(a,x))
  from A1 A2 have I: Tail(a) : n → X using tail˙props
```

```
    by simp
  from A1 A2 A3 have
    succ(n) ∈ nat and Append(a,x) : succ(succ(n)) → X
    using append˙props by auto
  then have II: ∀k ∈ succ(n). c(k) = Append(a,x)(succ(k))
    by (rule tail˙props)
  from assms have
    b : succ(n) → X and c : succ(n) → X
    using tail˙props append˙props by auto
  moreover have ∀k ∈ succ(n). b(k) = c(k)
  proof -
    { fix k assume k ∈ succ(n)
      hence k ∈ n ∨ k = n by auto
      moreover
      { assume A4: k ∈ n
 with assms II have c(k) = a(succ(k))
   using succ˙ineq append˙props by simp
 moreover
 from A3 I have ∀k∈n. b(k) = Tail(a)(k)
   using append˙props by simp
 with A1 A2 A4 have b(k) =  a(succ(k))
   using tail˙props by simp
 ultimately have b(k) = c(k) by simp }
      moreover
      { assume A5: k = n
 with A2 A3 I II have b(k) = c(k)
   using append˙props by auto }
      ultimately have b(k) = c(k) by auto
    } thus thesis by simp
  qed
  ultimately show b = c by (rule func˙eq)
qed
```

Properties of Init.

```
theorem init˙props:
  assumes A1: n ∈ nat and A2: a: succ(n) → X
  shows
  Init(a) : n → X
  ∀k∈n. Init(a)(k) = a(k)
  a = Append(Init(a), a(n))
proof -
  have n ⊆ succ(n) by auto
  with A2 have restrict(a,n): n → X
    using restrict˙type2 by simp
  moreover from A1 A2 have I: restrict(a,n) = Init(a)
    using func1˙1˙L1 pred˙succ˙eq Init˙def by simp
  ultimately show thesis1: Init(a) : n → X by simp
  { fix k assume k∈n
    then have restrict(a,n)(k) = a(k)
```

```
    using restrict by simp
  with I have Init(a)(k) = a(k) by simp
 } then show thesis2: ∀k∈n. Init(a)(k) = a(k) by simp
 let b = Append(Init(a), a(n))
 from A2 thesis1 have II:
   Init(a) : n → X   a(n) ∈ X
   b = Append(Init(a), a(n))
   using apply˙funtype by auto
 note A2
 moreover from II have b : succ(n) → X
   by (rule append˙props)
 moreover have ∀k ∈ succ(n). a(k) = b(k)
 proof -
   { fix k assume A3: k ∈ n
     from II have ∀j∈n. b(j) = Init(a)(j)
by (rule append˙props)
     with thesis2 A3 have a(k) = b(k) by simp }
   moreover
   from II have b(n) = a(n)
     by (rule append˙props)
   hence  a(n) = b(n) by simp
   ultimately show ∀k ∈ succ(n). a(k) = b(k)
     by simp
 qed
 ultimately show a = b by (rule func˙eq)
qed
```

If we take init of the result of append, we get back the same list.

```
lemma init˙append: assumes A1: n ∈ nat and A2: a:n→X and A3: x ∈ X
  shows Init(Append(a,x)) = a
proof -
  from A2 A3 have Append(a,x): succ(n)→X using append˙props by simp
   with A1 have Init(Append(a,x)):n→X and ∀k∈n. Init(Append(a,x))(k) = Ap-
pend(a,x)(k)
    using init˙props by auto
  with A2 A3 have ∀k∈n. Init(Append(a,x))(k) = a(k) using append˙props by simp
  with ‹Init(Append(a,x)):n→X› A2 show thesis by (rule func˙eq)
qed
```

A reformulation of definition of Init.

```
lemma init˙def: assumes n ∈ nat and x:succ(n)→X
  shows Init(x) = restrict(x,n)
  using assms func1˙1˙L1 Init˙def by simp
```

A lemma about extending a finite sequence by one more value. This is just a more explicit version of append˙props.

```
lemma finseq˙extend:
  assumes  a:n→X   y∈X   b = a ∪ –⟨n,y⟩˝
  shows
```

```
  b: succ(n) → X
  ∀k∈n. b(k) = a(k)
  b(n) = y
  using assms Append˙def func1˙1˙L1 append˙props by auto
```

The next lemma is a bit displaced as it is mainly about finite sets. It is proven here because it uses the notion of Append. Suppose we have a list of element of A is a bijection. Then for every element that does not belong to A we can we can construct a bijection for the set $A \cup \{x\}$ by appending x. This is just a specialised version of lemma bij˙extend˙point from func1.thy.

```
lemma bij˙append˙point:
  assumes A1: n ∈ nat and A2: b ∈ bij(n,X) and A3: x ∉ X
  shows Append(b,x) ∈ bij(succ(n), X ∪ –x″)
proof -
  from A2 A3 have b ∪ –⟨n,x⟩″ ∈ bij(n ∪ –n″,X ∪ –x″)
    using mem˙not˙refl bij˙extend˙point by simp
  moreover have Append(b,x) = b ∪ –⟨n,x⟩″
  proof -
    from A2 have b:n→X
      using bij˙def surj˙def by simp
    then have b : n → X ∪ –x″ using func1˙1˙L1B
      by blast
    then show Append(b,x) = b ∪ –⟨n,x⟩″
      using Append˙def func1˙1˙L1 by simp
  qed
  ultimately show thesis using succ˙explained by auto
qed
```

The next lemma rephrases the definition of Last. Recall that in ZF we have $\{0, 1, 2, .., n\} = n + 1 =$succ$(n)$.

```
lemma last˙seq˙elem: assumes a: succ(n) → X shows Last(a) = a(n)
  using assms func1˙1˙L1 pred˙succ˙eq Last˙def by simp
```

If two finite sequences are the same when restricted to domain one shorter than the original and have the same value on the last element, then they are equal.

```
lemma finseq˙restr˙eq: assumes A1: n ∈ nat and
  A2: a: succ(n) → X  b: succ(n) → X and
  A3: restrict(a,n) = restrict(b,n) and
  A4: a(n) = b(n)
  shows a = b
proof -
  { fix k assume k ∈ succ(n)
    then have k ∈ n ∨ k = n by auto
    moreover
    { assume k ∈ n
      then have
 restrict(a,n)(k) = a(k) and restrict(b,n)(k) = b(k)
```

```
 using restrict by auto
      with A3 have a(k) = b(k) by simp }
    moreover
    { assume k = n
      with A4 have a(k) = b(k) by simp }
    ultimately have a(k) = b(k) by auto
  } then have ∀ k ∈ succ(n). a(k) = b(k) by simp
  with A2 show a = b by (rule func˙eq)
qed
```

Concatenating a list of length 1 is the same as appending its first (and only) element. Recall that in ZF set theory $1 = \{0\}$.

```
lemma append˙1elem: assumes A1: n ∈ nat and
  A2: a: n → X  and A3: b : 1 → X
  shows Concat(a,b) = Append(a,b(0))
proof -
  let C = Concat(a,b)
  let A = Append(a,b(0))
  from A1 A2 A3 have I:
    n ∈ nat  1 ∈ nat
    a:n→X   b:1→X by auto
  have C : succ(n) → X
  proof -
    from I have C : n #+ 1 → X
      by (rule concat˙props)
    with A1 show C : succ(n) → X by simp
  qed
  moreover from A2 A3 have A : succ(n) → X
    using apply˙funtype append˙props by simp
  moreover have ∀k ∈ succ(n). C(k) = A(k)
  proof
    fix k assume k ∈ succ(n)
    moreover
    { assume k ∈ n
      moreover from I have ∀i ∈ n. C(i) = a(i)
 by (rule concat˙props)
      moreover from A2 A3 have ∀i∈n. A(i) = a(i)
 using apply˙funtype append˙props by simp
      ultimately have C(k) =  A(k) by simp }
    moreover have C(n) = A(n)
    proof -
      from I have ∀j ∈ 1. C(n #+ j) = b(j)
 by (rule concat˙props)
      with A1 A2 A3 show C(n) = A(n)
 using apply˙funtype append˙props by simp
    qed
    ultimately show C(k) = A(k) by auto
  qed
  ultimately show C = A by (rule func˙eq)
```

qed

A simple lemma about lists of length 1.

```
lemma list˙len1˙singleton: assumes A1: x∈X
  shows –⟨0,x⟩″ : 1 → X
proof -
  from A1 have –⟨0,x⟩″ : –0″ → X using pair˙func˙singleton
    by simp
  moreover have –0″ = 1 by auto
  ultimately show thesis by simp
qed
```

A singleton list is in fact a singleton set with a pair as the only element.

```
lemma list˙singleton˙pair: assumes A1: x:1→X shows x = –⟨0,x(0)⟩″
proof -
  from A1 have x = –⟨t,x(t)⟩. t∈1″ by (rule fun˙is˙set˙of˙pairs)
  hence x = –⟨t,x(t)⟩. t∈–0″ ″ by simp
  thus thesis by simp
qed
```

When we append an element to the empty list we get a list with length 1.

```
lemma empty˙append1: assumes A1: x∈X
  shows Append(0,x): 1 → X and Append(0,x)(0) = x
proof -
  let a = Append(0,x)
  have a = –⟨0,x⟩″ using Append˙def by auto
  with A1 show a : 1 → X and a(0) = x
    using list˙len1˙singleton pair˙func˙singleton
    by auto
qed
```

Appending an element is the same as concatenating with certain pair.

```
lemma append˙concat˙pair:
  assumes n ∈ nat and a: n → X and x∈X
  shows Append(a,x) = Concat(a,–⟨0,x⟩″)
  using assms list˙len1˙singleton append˙1elem pair˙val
  by simp
```

An associativity property involving concatenation and appending. For proof we just convert appending to concatenation and use concat˙assoc.

```
lemma concat˙append˙assoc: assumes A1: n ∈ nat  k ∈ nat and
  A2: a:n→X   b:k→X and A3: x ∈ X
  shows Append(Concat(a,b),x) = Concat(a, Append(b,x))
proof -
  from A1 A2 A3 have
    n #+ k ∈ nat   Concat(a,b) : n #+ k → X   x ∈ X
    using concat˙props by auto
  then have
```

```
    Append(Concat(a,b),x) =  Concat(Concat(a,b),–⟨0,x⟩″)
    by (rule append˙concat˙pair)
  moreover
  from A1 A2 A3 have
    n ∈ nat  k ∈ nat  1 ∈ nat
     a:n→X   b:k→X  –⟨0,x⟩″ :  1 → X
    using list˙len1˙singleton by auto
  then have
    Concat(Concat(a,b),–⟨0,x⟩″) = Concat(a, Concat(b,–⟨0,x⟩″))
    by (rule concat˙assoc)
  moreover from A1 A2 A3 have Concat(b,–⟨0,x⟩″) =  Append(b,x)
    using list˙len1˙singleton append˙1elem pair˙val by simp
  ultimately show Append(Concat(a,b),x) = Concat(a, Append(b,x))
    by simp
qed
```

An identity involving concatenating with init and appending the last element.

```
lemma concat˙init˙last˙elem:
  assumes n ∈ nat  k ∈ nat and
  a: n → X  and b : succ(k) → X
  shows Append(Concat(a,Init(b)),b(k)) = Concat(a,b)
  using assms init˙props apply˙funtype concat˙append˙assoc
  by simp
```

A lemma about creating lists by composition and how Append behaves in such case.

```
lemma list˙compose˙append:
  assumes A1: n ∈ nat and A2: a : n → X and
  A3: x ∈ X and A4: c : X → Y
  shows
  c O Append(a,x) : succ(n) → Y
  c O Append(a,x) = Append(c O a, c(x))
proof -
  let b = Append(a,x)
  let d = Append(c O a, c(x))
  from A2 A4 have c O a : n → Y
    using comp˙fun by simp
  from A2 A3 have b : succ(n) → X
    using append˙props by simp
  with A4 show c O b : succ(n) → Y
    using comp˙fun by simp
  moreover from A3 A4 ‹c O a : n → Y› have
    d: succ(n) → Y
    using apply˙funtype append˙props by simp
  moreover have ∀k ∈ succ(n). (c O b) (k) = d(k)
  proof -
    { fix k assume k ∈ succ(n)
      with ‹b : succ(n) → X› have
```

```
 (c O b) (k) = c(b(k))
 using comp˙fun˙apply by simp
      with A2 A3 A4 ‹c O a : n → Y› ‹c O a : n → Y› ‹k ∈ succ(n)›
      have (c O b) (k) = d(k)
 using append˙props comp˙fun˙apply apply˙funtype
 by auto
    } thus thesis by simp
  qed
  ultimately show c O b = d by (rule func˙eq)
qed
```

A lemma about appending an element to a list defined by set comprehension.

```
lemma set˙list˙append:  assumes
  A1: ∀i ∈ succ(k). b(i) ∈ X and
  A2: a = −⟨i,b(i)⟩. i ∈ succ(k)″
  shows
  a: succ(k) → X
  −⟨i,b(i)⟩. i ∈ k″: k → X
  a = Append(−⟨i,b(i)⟩. i ∈ k″,b(k))
proof -
  from A1 have −⟨i,b(i)⟩. i ∈ succ(k)″ : succ(k) → X
    by (rule ZF˙fun˙from˙total)
  with A2 show a: succ(k) → X by simp
  from A1 have ∀i ∈ k. b(i) ∈ X
    by simp
  then show −⟨i,b(i)⟩. i ∈ k″: k → X
    by (rule ZF˙fun˙from˙total)
  with A2 show a = Append(−⟨i,b(i)⟩. i ∈ k″,b(k))
    using func1˙1˙L1 Append˙def by auto
qed
```

An induction theorem for lists.

```
lemma list˙induct: assumes A1: ∀b∈1→X. P(b) and
  A2: ∀b∈NELists(X). P(b) ⟶ (∀x∈X. P(Append(b,x))) and
  A3: d ∈ NELists(X)
  shows P(d)
proof -
  { fix n
    assume n∈nat
    moreover from A1 have ∀b∈succ(0)→X. P(b) by simp
     moreover have ∀k∈nat. ((∀b∈succ(k)→X. P(b)) ⟶ (∀c∈succ(succ(k))→X.
P(c)))
    proof -
      { fix k assume k ∈ nat assume ∀b∈succ(k)→X. P(b)
        have ∀c∈succ(succ(k))→X. P(c)
        proof
          fix c assume c: succ(succ(k))→X
          let b = Init(c)
          let x = c(succ(k))
```

```
        from ‹k ∈ nat› ‹c: succ(succ(k))→X› have b:succ(k)→X
          using init˙props by simp
        with A2 ‹k ∈ nat› ‹∀b∈succ(k)→X. P(b)› have ∀x∈X. P(Append(b,x))
          using NELists˙def by auto
          with ‹c: succ(succ(k))→X› have P(Append(b,x)) using apply˙funtype by
simp
        with ‹k ∈ nat› ‹c: succ(succ(k))→X› show P(c)
          using init˙props by simp
      qed
    } thus thesis by simp
  qed
  ultimately have ∀b∈succ(n)→X. P(b) by (rule ind˙on˙nat)
 } with A3 show thesis using NELists˙def by auto
qed
```

18.2 Lists and cartesian products

Lists of length n of elements of some set X can be thought of as a model of the cartesian product X^n which is more convenient in many applications.

There is a natural bijection between the space $(n+1) \to X$ of lists of length $n+1$ of elements of X and the cartesian product $(n \to X) \times X$.

```
lemma lists˙cart˙prod: assumes n ∈ nat
 shows −⟨x,⟨Init(x),x(n)⟩⟩. x ∈ succ(n)→X″ ∈ bij(succ(n)→X,(n→X)×X)
proof -
 let f = −⟨x,⟨Init(x),x(n)⟩⟩. x ∈ succ(n)→X″
 from assms have ∀x ∈ succ(n)→X. ⟨Init(x),x(n)⟩ ∈ (n→X)×X
   using init˙props succ˙iff apply˙funtype by simp
 then have I: f: (succ(n)→X)→((n→X)×X) by (rule ZF˙fun˙from˙total)
 moreover from assms I have ∀x∈succ(n)→X.∀y∈succ(n)→X. f(x)=f(y) ⟶ x=y
   using ZF˙fun˙from˙tot˙val init˙def finseq˙restr˙eq by auto
 moreover have ∀p∈(n→X)×X.∃x∈succ(n)→X. f(x) = p
 proof
   fix p assume p ∈ (n→X)×X
   let x = Append(fst(p),snd(p))
   from assms ‹p ∈ (n→X)×X› have x:succ(n)→X using append˙props by simp
   with I have f(x) = ⟨Init(x),x(n)⟩ using succ˙iff ZF˙fun˙from˙tot˙val by simp
   moreover from assms ‹p ∈ (n→X)×X› have Init(x) = fst(p) and x(n) = snd(p)
     using init˙append append˙props by auto
   ultimately have f(x) = ⟨fst(p),snd(p)⟩ by auto
   with ‹p ∈ (n→X)×X› ‹x:succ(n)→X› show ∃x∈succ(n)→X. f(x) = p by auto
 qed
 ultimately show thesis using inj˙def surj˙def bij˙def by auto
qed
```

We can identify a set X with lists of length one of elements of X.

```
lemma singleton˙list˙bij: shows −⟨x,x(0)⟩. x∈1→X″ ∈ bij(1→X,X)
proof -
 let f = −⟨x,x(0)⟩. x∈1→X″
```

```
  have ∀x∈1→X. x(0) ∈ X using apply˙funtype by simp
  then have I: f:(1→X)→X by (rule ZF˙fun˙from˙total)
  moreover have ∀x∈1→X.∀y∈1→X. f(x) = f(y) ⟶ x=y
  proof -
    { fix x y
      assume x:1→X y:1→X and f(x) = f(y)
      with I have x(0) = y(0) using ZF˙fun˙from˙tot˙val by auto
      moreover from ‹x:1→X› ‹y:1→X› have x = −⟨0,x(0)⟩″ and y = −⟨0,y(0)⟩″
        using list˙singleton˙pair by auto
      ultimately have x=y by simp
    } thus thesis by auto
  qed
  moreover have ∀y∈X. ∃x∈1→X. f(x)=y
  proof
    fix y assume y∈X
    let x = −⟨0,y⟩″
    from I ‹y∈X› have x:1→X and f(x) = y
      using list˙len1˙singleton ZF˙fun˙from˙tot˙val pair˙val by auto
    thus ∃x∈1→X. f(x)=y by auto
  qed
  ultimately show thesis using inj˙def surj˙def bij˙def by simp
qed
```

We can identify a set of X-valued lists of length with X.

```
lemma list˙singleton˙bij: shows
  −⟨x,−⟨0,x⟩″⟩.x∈X″ ∈ bij(X,1→X) and
  −⟨y,y(0)⟩. y∈1→X″ = converse(−⟨x,−⟨0,x⟩″⟩.x∈X″) and
  −⟨x,−⟨0,x⟩″⟩.x∈X″ = converse(−⟨y,y(0)⟩. y∈1→X″)
proof -
  let f = −⟨y,y(0)⟩. y∈1→X″
  let g = −⟨x,−⟨0,x⟩″⟩.x∈X″
  have 1 = −0″ by auto
  then have f ∈ bij(1→X,X) and g:X→(1→X)
    using singleton˙list˙bij pair˙func˙singleton ZF˙fun˙from˙total
    by auto
  moreover have ∀y∈1→X.g(f(y)) = y
  proof
    fix y assume y:1→X
    have f:(1→X)→X using singleton˙list˙bij bij˙def inj˙def by simp
    with ‹1 = −0″› ‹y:1→X› ‹g:X→(1→X)› show g(f(y)) = y
      using ZF˙fun˙from˙tot˙val apply˙funtype func˙singleton˙pair
      by simp
  qed
  ultimately show g ∈ bij(X,1→X) and f = converse(g) and g = converse(f)
    using comp˙conv˙id by auto
qed
```

What is the inverse image of a set by the natural bijection between X-valued singleton lists and X?

```
lemma singleton˙vimage: assumes U⊆X shows –x∈1→X. x(0) ∈ U″ = – –⟨0,y⟩″.
y∈U″
proof
  have 1 = –0″ by auto
  { fix x assume x ∈ –x∈1→X. x(0) ∈ U″
    with ‹1 = –0″› have x = –⟨0, x(0)⟩″ using func˙singleton˙pair by auto
  } thus –x∈1→X. x(0) ∈ U″ ⊆ – –⟨0,y⟩″. y∈U″ by auto
  { fix x assume x ∈ – –⟨0,y⟩″. y∈U″
    then obtain y where x = –⟨0,y⟩″ and y∈U by auto
    with ‹1 = –0″› assms have x:1→X using pair˙func˙singleton by auto
  } thus – –⟨0,y⟩″. y∈U″ ⊆ –x∈1→X. x(0) ∈ U″ by auto
qed
```

A technical lemma about extending a list by values from a set.

```
lemma list˙append˙from: assumes A1: n ∈ nat and A2: U ⊆ n→X and A3: V ⊆ X
  shows
  –x ∈ succ(n)→X. Init(x) ∈ U ∧ x(n) ∈ V″ = (⋃y∈V.–Append(x,y).x∈U″)
proof -
  { fix x assume x ∈ –x ∈ succ(n)→X. Init(x) ∈ U ∧ x(n) ∈ V″
    then have x ∈ succ(n)→X and Init(x) ∈ U and I: x(n) ∈ V
      by auto
    let y = x(n)
    from A1 and ‹x ∈ succ(n)→X›  have x = Append(Init(x),y)
      using init˙props by simp
    with I and ‹Init(x) ∈ U› have x ∈ (⋃y∈V.–Append(a,y).a∈U″) by auto
  }
  moreover
  { fix x assume x ∈ (⋃y∈V.–Append(a,y).a∈U″)
    then obtain a y where y∈V and a∈U and x = Append(a,y) by auto
    with A2 A3 have x: succ(n)→X using append˙props by blast
    from A2 A3 ‹y∈V› ‹a∈U› have a:n→X and y∈X by auto
    with A1 ‹a∈U›  ‹y∈V› ‹x = Append(a,y)› have Init(x) ∈ U and  x(n) ∈ V
      using append˙props init˙append by auto
    with ‹x: succ(n)→X› have x ∈ –x ∈ succ(n)→X. Init(x) ∈ U ∧ x(n) ∈ V″
      by auto
  }
  ultimately show thesis by blast
qed

end
```

19 Inductive sequences

```
theory InductiveSeq˙ZF imports Nat˙ZF˙IML FiniteSeq˙ZF

begin
```

In this theory we discuss sequences defined by conditions of the form $a_0 = x,\ a_{n+1} = f(a_n)$ and similar.

19.1 Sequences defined by induction

One way of defining a sequence (that is a function $a : \mathbb{N} \to X$) is to provide the first element of the sequence and a function to find the next value when we have the current one. This is usually called "defining a sequence by induction". In this section we set up the notion of a sequence defined by induction and prove the theorems needed to use it.

First we define a helper notion of the sequence defined inductively up to a given natural number n.

```
definition
  InductiveSequenceN(x,f,n) ≡
  THE a. a: succ(n) → domain(f) ∧ a(0) = x ∧ (∀k∈n. a(succ(k)) = f(a(k)))
```

From that we define the inductive sequence on the whole set of natural numbers. Recall that in Isabelle/ZF the set of natural numbers is denoted nat.

```
definition
  InductiveSequence(x,f) ≡ ⋃n∈nat. InductiveSequenceN(x,f,n)
```

First we will consider the question of existence and uniqueness of finite inductive sequences. The proof is by induction and the next lemma is the $P(0)$ step. To understand the notation recall that for natural numbers in set theory we have $n = \{0, 1, .., n-1\}$ and succ(n)= $\{0, 1, .., n\}$.

```
lemma indseq˙exun0: assumes A1: f: X→X and A2: x∈X
  shows
  ∃! a. a: succ(0) → X ∧ a(0) = x ∧ ( ∀k∈0. a(succ(k)) = f(a(k)) )
proof
  fix a b
  assume A3:
    a: succ(0) → X ∧ a(0) = x ∧ ( ∀k∈0. a(succ(k)) = f(a(k)) )
    b: succ(0) → X ∧ b(0) = x ∧ ( ∀k∈0. b(succ(k)) = f(b(k)) )
  moreover have succ(0) = –0″ by auto
  ultimately have a: –0″ → X  b: –0″ → X by auto
  then have a = –⟨0, a(0)⟩″   b = –⟨0, b(0)⟩″ using func˙singleton˙pair
    by auto
  with A3 show a=b by simp
next
  let a = –⟨0,x⟩″
  have a : –0″ → –x″ using singleton˙fun by simp
  moreover from A1 A2 have –x″ ⊆ X by simp
  ultimately have a : –0″ → X
    using func1˙1˙L1B by blast
  moreover have –0″ = succ(0) by auto
  ultimately have a : succ(0) → X by simp
  with A1 show
    ∃ a. a: succ(0) → X ∧ a(0) = x ∧ (∀k∈0. a(succ(k)) = f(a(k)))
    using singleton˙apply by auto
```

qed

A lemma about restricting finite sequences needed for the proof of the inductive step of the existence and uniqueness of finite inductive seqences.

lemma indseq˙restrict:
  assumes A1: f: X→X and A2: x∈X and A3: n ∈ nat and
  A4: a: succ(succ(n))→ X ∧ a(0) = x ∧ (∀k∈succ(n). a(succ(k)) = f(a(k)))
  and A5: a_r = restrict(a,succ(n))
  shows
  a_r: succ(n) → X ∧ a_r(0) = x ∧ (∀k∈n. a_r(succ(k)) = f(a_r(k)))
proof -
  from A3 have succ(n) ⊆ succ(succ(n)) by auto
  with A4 A5 have a_r: succ(n) → X using restrict˙type2 by auto
  moreover
  from A3 have 0 ∈ succ(n) using empty˙in˙every˙succ by simp
  with A4 A5 have a_r(0) = x using restrict˙if by simp
  moreover from A3 A4 A5 have ∀k∈n. a_r(succ(k)) = f(a_r(k))
    using succ˙ineq restrict˙if by auto
  ultimately show thesis by simp
qed

Existence and uniqueness of finite inductive sequences. The proof is by induction and the next lemma is the inductive step.

lemma indseq˙exun˙ind:
  assumes A1: f: X→X and A2: x∈X and A3: n ∈ nat and
  A4: ∃! a. a: succ(n) → X ∧ a(0) = x ∧ (∀k∈n. a(succ(k)) = f(a(k)))
  shows
  ∃! a. a: succ(succ(n)) → X ∧ a(0) = x ∧
  (∀k∈succ(n). a(succ(k)) = f(a(k)))
proof
  fix a b assume
    A5: a: succ(succ(n)) → X ∧ a(0) = x ∧
    (∀k∈succ(n). a(succ(k)) = f(a(k))) and
    A6: b: succ(succ(n)) → X ∧ b(0) = x ∧
    (∀k∈succ(n). b(succ(k)) = f(b(k)))
  show a = b
  proof -
    let a_r = restrict(a,succ(n))
    let b_r = restrict(b,succ(n))
    note A1 A2 A3 A5
    moreover have a_r = restrict(a,succ(n)) by simp
    ultimately have I:
      a_r: succ(n) → X ∧ a_r(0) = x ∧ (∀k∈n. a_r(succ(k)) = f(a_r(k)))
      by (rule indseq˙restrict)
    note A1 A2 A3 A6
    moreover have b_r = restrict(b,succ(n)) by simp
    ultimately have
      b_r: succ(n) → X ∧ b_r(0) = x ∧ (∀k∈n. b_r(succ(k)) = f(b_r(k)))
      by (rule indseq˙restrict)

```
    with A4 I have II: a_r = b_r by blast
    from A3 have succ(n) ∈ nat by simp
    moreover from A5 A6 have
      a: succ(succ(n)) → X and b: succ(succ(n)) → X
      by auto
    moreover note II
    moreover
    have T: n ∈ succ(n) by simp
    then have a_r(n) = a(n) and b_r(n) = b(n) using restrict
      by auto
    with A5 A6 II T have a(succ(n)) = b(succ(n)) by simp
    ultimately show a = b by (rule finseq˙restr˙eq)
  qed
next show
    ∃ a. a: succ(succ(n)) → X ∧ a(0) = x ∧
    ( ∀k∈succ(n). a(succ(k)) = f(a(k)) )
  proof -
    from A4 obtain a where III: a: succ(n) → X and IV: a(0) = x
      and V: ∀k∈n. a(succ(k)) = f(a(k)) by auto
    let b = a ∪ −⟨succ(n), f(a(n))⟩″
    from A1 III have
      VI: b : succ(succ(n)) → X and
      VII: ∀k ∈ succ(n). b(k) = a(k) and
      VIII: b(succ(n)) = f(a(n))
      using apply˙funtype finseq˙extend by auto
    from A3 have 0 ∈ succ(n) using empty˙in˙every˙succ by simp
    with IV VII have IX: b(0) = x by auto
    { fix k assume k ∈ succ(n)
      then have k∈n ∨ k = n by auto
      moreover
      { assume A7: k ∈ n
 with A3 VII have b(succ(k)) = a(succ(k))
   using succ˙ineq by auto
 also from A7 V VII have a(succ(k)) = f(b(k)) by simp
 finally have b(succ(k)) =  f(b(k)) by simp }
      moreover
      { assume A8: k = n
 with VIII have b(succ(k)) =  f(a(k)) by simp
 with A8 VII VIII have b(succ(k)) =  f(b(k)) by simp }
      ultimately have b(succ(k)) =  f(b(k)) by auto
    } then have ∀k ∈ succ(n). b(succ(k)) =  f(b(k)) by simp
    with VI IX show thesis by auto
  qed
qed
```

The next lemma combines indseq˙exun0 and indseq˙exun˙ind to show the existence and uniqueness of finite sequences defined by induction.

```
lemma indseq˙exun:
  assumes A1: f: X→X and A2: x∈X and A3: n ∈ nat
```

```
  shows
  ∃! a. a: succ(n) → X ∧ a(0) = x ∧ (∀k∈n. a(succ(k)) = f(a(k)))
proof -
  note A3
  moreover from A1 A2 have
    ∃! a. a: succ(0) → X ∧ a(0) = x ∧ ( ∀k∈0. a(succ(k)) = f(a(k)) )
    using indseq˙exun0 by simp
  moreover from A1 A2 have ∀k ∈ nat.
    ( ∃! a. a: succ(k) → X ∧ a(0) = x ∧
    ( ∀i∈k. a(succ(i)) = f(a(i)) )) ⟶
    ( ∃! a. a: succ(succ(k)) → X ∧ a(0) = x ∧
    ( ∀i∈succ(k). a(succ(i)) = f(a(i)) ) )
    using indseq˙exun˙ind by simp
  ultimately show
    ∃! a. a: succ(n) → X ∧ a(0) = x ∧ ( ∀k∈n. a(succ(k)) = f(a(k)) )
    by (rule ind˙on˙nat)
qed
```

We are now ready to prove the main theorem about finite inductive sequences.

```
theorem fin˙indseq˙props:
  assumes A1: f: X→X and A2: x∈X and A3: n ∈ nat and
  A4: a = InductiveSequenceN(x,f,n)
  shows
  a: succ(n) → X
  a(0) = x
  ∀k∈n. a(succ(k)) = f(a(k))
proof -
  let i = THE a. a: succ(n) → X ∧ a(0) = x ∧
    ( ∀k∈n. a(succ(k)) = f(a(k)) )
  from A1 A2 A3 have
    ∃! a. a: succ(n) → X ∧ a(0) = x ∧ ( ∀k∈n. a(succ(k)) = f(a(k)) )
    using indseq˙exun by simp
  then have
    i: succ(n) → X ∧ i(0) = x ∧ ( ∀k∈n. i(succ(k)) = f(i(k)) )
    by (rule theI)
  moreover from A1 A4 have a = i
    using InductiveSequenceN˙def func1˙1˙L1 by simp
  ultimately show
    a: succ(n) → X   a(0) = x   ∀k∈n. a(succ(k)) = f(a(k))
    by auto
qed
```

A corollary about the domain of a finite inductive sequence.

```
corollary fin˙indseq˙domain:
  assumes A1: f: X→X and A2: x∈X and A3: n ∈ nat
  shows domain(InductiveSequenceN(x,f,n)) = succ(n)
proof -
  from assms have InductiveSequenceN(x,f,n) : succ(n) → X
```

```
    using fin˙indseq˙props by simp
  then show thesis using func1˙1˙L1 by simp
qed
```

The collection of finite sequences defined by induction is consistent in the sense that the restriction of the sequence defined on a larger set to the smaller set is the same as the sequence defined on the smaller set.

```
lemma indseq˙consistent: assumes A1: f: X→X and A2: x∈X and
  A3: i ∈ nat  j ∈ nat and A4: i ⊆ j
  shows
  restrict(InductiveSequenceN(x,f,j),succ(i)) = InductiveSequenceN(x,f,i)
proof -
  let a = InductiveSequenceN(x,f,j)
  let b = restrict(InductiveSequenceN(x,f,j),succ(i))
  let c = InductiveSequenceN(x,f,i)
  from A1 A2 A3 have
    a: succ(j) → X  a(0) = x   ∀k∈j. a(succ(k)) = f(a(k))
    using fin˙indseq˙props by auto
  with A3 A4 have
    b: succ(i) → X ∧ b(0) = x ∧ ( ∀k∈i. b(succ(k)) = f(b(k)))
    using succ˙subset restrict˙type2 empty˙in˙every˙succ restrict succ˙ineq
    by auto
  moreover from A1 A2 A3 have
    c: succ(i) → X ∧ c(0) = x ∧ ( ∀k∈i. c(succ(k)) = f(c(k)))
    using fin˙indseq˙props by simp
  moreover from A1 A2 A3 have
    ∃! a. a: succ(i) → X ∧ a(0) = x ∧ ( ∀k∈i. a(succ(k)) = f(a(k)) )
    using indseq˙exun by simp
  ultimately show b = c by blast
qed
```

For any two natural numbers one of the corresponding inductive sequences is contained in the other.

```
lemma indseq˙subsets: assumes A1: f: X→X and A2: x∈X and
  A3: i ∈ nat  j ∈ nat and
  A4: a = InductiveSequenceN(x,f,i)  b = InductiveSequenceN(x,f,j)
  shows a ⊆ b ∨ b ⊆ a
proof -
  from A3 have i⊆j ∨ j⊆i using nat˙incl˙total by simp
  moreover
  { assume i⊆j
    with A1 A2 A3 A4 have restrict(b,succ(i)) = a
      using indseq˙consistent by simp
    moreover have restrict(b,succ(i)) ⊆ b
      using restrict˙subset by simp
    ultimately have a ⊆ b ∨ b ⊆ a by simp }
  moreover
  { assume j⊆i
    with A1 A2 A3 A4 have restrict(a,succ(j)) = b
```

```
      using indseq˙consistent by simp
    moreover have restrict(a,succ(j)) ⊆ a
      using restrict˙subset by simp
    ultimately have a ⊆ b ∨ b ⊆ a by simp }
  ultimately show  a ⊆ b ∨ b ⊆ a by auto
qed
```

The first theorem about properties of infinite inductive sequences: inductive sequence is a indeed a sequence (i.e. a function on the set of natural numbers.

```
theorem indseq˙seq: assumes  A1: f: X→X and A2: x∈X
  shows InductiveSequence(x,f) : nat → X
proof -
  let S = –InductiveSequenceN(x,f,n). n ∈ nat″
  { fix a assume a∈S
    then obtain n where n ∈ nat and a =  InductiveSequenceN(x,f,n)
      by auto
    with A1 A2 have a : succ(n)→X using fin˙indseq˙props
      by simp
    then have ∃ A B. a:A→B by auto
  } then have ∀ a ∈ S. ∃ A B. a:A→B by auto
  moreover
  { fix a b assume a∈S   b∈S
    then obtain i j where i∈nat  j∈nat and
      a = InductiveSequenceN(x,f,i)   b = InductiveSequenceN(x,f,j)
      by auto
    with A1 A2 have a⊆b ∨ b⊆a using indseq˙subsets by simp
  } then have ∀ a∈S. ∀ b∈S. a⊆b ∨ b⊆a by auto
  ultimately have ⋃S : domain(⋃S) → range(⋃S)
    using fun˙Union by simp
  with A1 A2 have I: ⋃S : nat → range(⋃S)
    using domain˙UN fin˙indseq˙domain nat˙union˙succ by simp
  moreover
  { fix k assume A3: k ∈ nat
    let y = (⋃S)(k)
    note I A3
    moreover have y = (⋃S)(k) by simp
    ultimately have ⟨k,y⟩ ∈ (⋃S) by (rule func1˙1˙L5A)
    then obtain n where n ∈ nat and II: ⟨k,y⟩ ∈ InductiveSequenceN(x,f,n)
      by auto
    with A1 A2 have InductiveSequenceN(x,f,n): succ(n) → X
      using fin˙indseq˙props by simp
    with II have y ∈ X using func1˙1˙L5 by blast
  } then have ∀ k ∈ nat.  (⋃S)(k) ∈ X by simp
  ultimately have ⋃S : nat → X using func1˙1˙L1A
    by blast
  then show InductiveSequence(x,f) : nat → X
    using InductiveSequence˙def by simp
qed
```

Restriction of an inductive sequence to a finite domain is the corresponding finite inductive sequence.

```
lemma indseq˙restr˙eq:
  assumes A1: f: X→X and A2: x∈X and A3: n ∈ nat
  shows
  restrict(InductiveSequence(x,f),succ(n)) = InductiveSequenceN(x,f,n)
proof -
  let a = InductiveSequence(x,f)
  let b = InductiveSequenceN(x,f,n)
  let S = –InductiveSequenceN(x,f,n). n ∈ nat˝
  from A1 A2 A3 have
    I: a : nat → X  and succ(n) ⊆ nat
    using indseq˙seq succnat˙subset˙nat by auto
  then have restrict(a,succ(n)) : succ(n) → X
    using restrict˙type2 by simp
  moreover from A1 A2 A3 have b : succ(n) → X
    using fin˙indseq˙props by simp
  moreover
  { fix k assume A4: k ∈ succ(n)
    from A1 A2 A3 I have
      ⋃S : nat → X   b ∈ S  b : succ(n) → X
      using InductiveSequence˙def fin˙indseq˙props by auto
    with A4 have restrict(a,succ(n))(k) = b(k)
      using fun˙Union˙apply InductiveSequence˙def restrict˙if
      by simp
  } then have ∀k ∈ succ(n). restrict(a,succ(n))(k) = b(k)
    by simp
  ultimately show thesis by (rule func˙eq)
qed
```

The first element of the inductive sequence starting at x and generated by f is indeed x.

```
theorem indseq˙valat0: assumes A1: f: X→X and A2: x∈X
  shows InductiveSequence(x,f)(0) = x
proof -
  let a = InductiveSequence(x,f)
  let b = InductiveSequenceN(x,f,0)
  have T: 0∈nat  0 ∈ succ(0) by auto
  with A1 A2 have b(0) = x
    using fin˙indseq˙props by simp
  moreover from T have restrict(a,succ(0))(0) = a(0)
    using restrict˙if by simp
  moreover from A1 A2 T have
    restrict(a,succ(0)) = b
    using indseq˙restr˙eq by simp
  ultimately show a(0) = x by simp
qed
```

An infinite inductive sequence satisfies the inductive relation that defines it.

```
theorem indseq˙vals:
  assumes A1: f: X→X and A2: x∈X  and A3: n ∈ nat
  shows
  InductiveSequence(x,f)(succ(n)) = f(InductiveSequence(x,f)(n))
proof -
  let a = InductiveSequence(x,f)
  let b = InductiveSequenceN(x,f,succ(n))
  from A3 have T:
    succ(n) ∈ succ(succ(n))
    succ(succ(n)) ∈ nat
    n ∈ succ(succ(n))
    by auto
  then have a(succ(n)) = restrict(a,succ(succ(n)))(succ(n))
    using restrict˙if by simp
  also from A1 A2 T have ... = f(restrict(a,succ(succ(n)))(n))
    using indseq˙restr˙eq fin˙indseq˙props by simp
  also from T have ... = f(a(n)) using restrict˙if by simp
  finally show a(succ(n)) = f(a(n)) by simp
qed
```

19.2 Images of inductive sequences

In this section we consider the properties of sets that are images of inductive sequences, that is are of the form $\{f^{(n)}(x) : n \in N\}$ for some x in the domain of f, where $f^{(n)}$ denotes the n'th iteration of the function f. For a function $f : X \to X$ and a point $x \in X$ such set is set is sometimes called the orbit of x generated by f.

The basic properties of orbits.

```
theorem ind˙seq˙image: assumes A1: f: X→X and A2: x∈X  and
  A3: A = InductiveSequence(x,f)(nat)
  shows x∈A and ∀y∈A. f(y) ∈ A
proof -
  let a = InductiveSequence(x,f)
  from A1 A2 have a : nat → X using indseq˙seq
    by simp
  with A3 have I: A = –a(n). n ∈ nat˝ using func˙imagedef
    by auto hence a(0) ∈ A by auto
  with A1 A2 show x∈A using indseq˙valat0 by simp
  { fix y assume y∈A
    with I obtain n where II: n ∈ nat and III: y = a(n)
      by auto
    with A1 A2 have a(succ(n)) = f(y)
      using indseq˙vals by simp
    moreover from I II have a(succ(n)) ∈ A by auto
    ultimately have f(y) ∈ A by simp
  } then show ∀y∈A. f(y) ∈ A by simp
qed
```

19.3 Subsets generated by a binary operation

In algebra we often talk about sets "generated" by an element, that is sets of the form (in multiplicative notation) $\{a^n | n \in Z\}$. This is a related to a general notion of "power" (as in $a^n = a \cdot a \cdot .. \cdot a$) or multiplicity $n \cdot a = a + a + .. + a$. The intuitive meaning of such notions is obvious, but we need to do some work to be able to use it in the formalized setting. This sections is devoted to sequences that are created by repeatedly applying a binary operation with the second argument fixed to some constant.

Basic propertes of sets generated by binary operations.

```
theorem binop˙gen˙set:
  assumes A1: f: X×Y → X and A2: x∈X  y∈Y and
  A3: a = InductiveSequence(x,Fix2ndVar(f,y))
  shows
  a : nat → X
  a(nat) ∈ Pow(X)
  x ∈ a(nat)
  ∀z ∈ a(nat). Fix2ndVar(f,y)(z) ∈ a(nat)
proof -
  let g = Fix2ndVar(f,y)
  from A1 A2 have I: g : X→X
    using fix˙2nd˙var˙fun by simp
  with A2 A3 show a : nat → X
    using indseq˙seq by simp
  then show a(nat) ∈ Pow(X) using func1˙1˙L6 by simp
  from A2 A3 I show x ∈ a(nat) using ind˙seq˙image by blast
  from A2 A3 I have
    g : X→X  x∈X  a(nat) = InductiveSequence(x,g)(nat)
    by auto
  then show ∀z ∈ a(nat). Fix2ndVar(f,y)(z) ∈ a(nat)
    by (rule ind˙seq˙image)
qed
```

A simple corollary to the theorem binop˙gen˙set: a set that contains all iterations of the application of a binary operation exists.

```
lemma binop˙gen˙set˙ex: assumes A1: f: X×Y → X and A2: x∈X  y∈Y
  shows –A ∈ Pow(X). x∈A ∧ (∀z ∈ A. f⟨z,y⟩ ∈ A) ″ ≠ 0
proof -
  let a = InductiveSequence(x,Fix2ndVar(f,y))
  let A = a(nat)
  from A1 A2 have I: A ∈ Pow(X) and x ∈ A using binop˙gen˙set
    by auto
  moreover
  { fix z assume T: z∈A
    with A1 A2 have Fix2ndVar(f,y)(z) ∈ A
      using binop˙gen˙set by simp
    moreover
```

```
    from I T have z ∈ X by auto
    with A1 A2 have Fix2ndVar(f,y)(z) = f⟨z,y⟩
      using fix˙var˙val by simp
    ultimately have f⟨z,y⟩ ∈ A by simp
  } then have ∀z ∈ A. f⟨z,y⟩ ∈ A by simp
  ultimately show thesis by auto
qed
```

A more general version of binop˙gen˙set where the generating binary operation acts on a larger set.

```
theorem binop˙gen˙set1: assumes A1: f: X×Y → X and
  A2: X1 ⊆ X and A3: x∈X1  y∈Y and
  A4: ∀t∈X1. f⟨t,y⟩ ∈ X1 and
  A5: a = InductiveSequence(x,Fix2ndVar(restrict(f,X1×Y),y))
shows
  a : nat → X1
  a(nat) ∈ Pow(X1)
  x ∈ a(nat)
  ∀z ∈ a(nat). Fix2ndVar(f,y)(z) ∈ a(nat)
  ∀z ∈ a(nat). f⟨z,y⟩ ∈ a(nat)
proof -
  let h = restrict(f,X1×Y)
  let g = Fix2ndVar(h,y)
  from A2 have X1×Y ⊆ X×Y by auto
  with A1 have I: h : X1×Y → X
    using restrict˙type2 by simp
  with A3 have II: g: X1 → X using fix˙2nd˙var˙fun by simp
  from A3 A4 I have ∀t∈X1. g(t) ∈ X1
    using restrict fix˙var˙val by simp
  with II have III: g : X1 → X1 using func1˙1˙L1A by blast
  with A3 A5 show a : nat → X1 using indseq˙seq by simp
  then show IV: a(nat) ∈ Pow(X1) using func1˙1˙L6 by simp
  from A3 A5 III show x ∈ a(nat) using ind˙seq˙image by blast
  from A3 A5 III have
    g : X1 → X1   x∈X1  a(nat) =  InductiveSequence(x,g)(nat)
    by auto
  then have ∀z ∈ a(nat). Fix2ndVar(h,y)(z) ∈ a(nat)
    by (rule ind˙seq˙image)
  moreover
  { fix z assume z ∈ a(nat)
    with IV have z ∈ X1 by auto
    with A1 A2 A3 have g(z) = Fix2ndVar(f,y)(z)
      using fix˙2nd˙var˙restr˙comm restrict by simp
  } then have ∀z ∈ a(nat). g(z) = Fix2ndVar(f,y)(z) by simp
  ultimately show ∀z ∈ a(nat). Fix2ndVar(f,y)(z) ∈ a(nat) by simp
  moreover
  { fix z assume z ∈ a(nat)
    with A2 IV have z∈X by auto
    with A1 A3 have Fix2ndVar(f,y)(z) = f⟨z,y⟩
```

```
    using fix˙var˙val by simp
  } then have ∀z ∈ a(nat). Fix2ndVar(f,y)(z) = f⟨z,y⟩
    by simp
  ultimately show ∀z ∈ a(nat). f⟨z,y⟩ ∈ a(nat)
    by simp
qed
```

A generalization of binop˙gen˙set˙ex that applies when the binary operation acts on a larger set. This is used in our Metamath translation to prove the existence of the set of real natural numbers. Metamath defines the real natural numbers as the smallest set that cantains 1 and is closed with respect to operation of adding 1.

```
lemma binop˙gen˙set˙ex1: assumes A1: f: X×Y → X and
  A2: X₁ ⊆ X and A3: x∈X₁  y∈Y and
  A4: ∀t∈X₁. f⟨t,y⟩ ∈ X₁
  shows –A ∈ Pow(X₁). x∈A ∧ (∀z ∈ A. f⟨z,y⟩ ∈ A) ″ ≠ 0
proof -
  let a = InductiveSequence(x,Fix2ndVar(restrict(f,X₁×Y),y))
  let A = a(nat)
  from A1 A2 A3 A4 have
    A ∈ Pow(X₁)   x ∈ A   ∀z ∈ A. f⟨z,y⟩ ∈ A
    using binop˙gen˙set1 by auto
  thus thesis by auto
qed
```

19.4 Inductive sequences with changing generating function

A seemingly more general form of a sequence defined by induction is a sequence generated by the difference equation $x_{n+1} = f_n(x_n)$ where $n \mapsto f_n$ is a given sequence of functions such that each maps X into inself. For example when $f_n(x) := x + x_n$ then the equation $S_{n+1} = f_n(S_n)$ describes the sequence $n \mapsto S_n = s_0 + \sum_{i=0}^{n} x_n$, i.e. the sequence of partial sums of the sequence $\{s_0, x_0, x_1, x_3, ..\}$.

The situation where the function that we iterate changes with n can be derived from the simpler case if we define the generating function appropriately. Namely, we replace the generating function in the definitions of InductiveSequenceN by the function $f : X \times n \to X \times n$, $f\langle x, k\rangle = \langle f_k(x), k+1\rangle$ if $k < n$, $\langle f_k(x), k\rangle$ otherwise. The first notion defines the expression we will use to define the generating function. To understand the notation recall that in standard Isabelle/ZF for a pair $s = \langle x, n\rangle$ we have $\mathrm{fst}(s) = x$ and $\mathrm{snd}(s) = n$.

```
definition
  StateTransfFunNMeta(F,n,s) ≡
  if (snd(s) ∈ n) then ⟨F(snd(s))(fst(s)), succ(snd(s))⟩ else s
```

Then we define the actual generating function on sets of pairs from $X \times \{0, 1, .., n\}$.

```
definition
  StateTransfFunN(X,F,n) ≡ −⟨s, StateTransfFunNMeta(F,n,s)⟩. s ∈ X×succ(n)″
```

Having the generating function we can define the expression that we cen use to define the inductive sequence generates.

```
definition
  StatesSeq(x,X,F,n) ≡
  InductiveSequenceN(⟨x,0⟩, StateTransfFunN(X,F,n),n)
```

Finally we can define the sequence given by a initial point x, and a sequence F of n functions.

```
definition
  InductiveSeqVarFN(x,X,F,n) ≡ −⟨k,fst(StatesSeq(x,X,F,n)(k))⟩. k ∈ succ(n)″
```

The state transformation function (StateTransfFunN is a function that transforms $X \times n$ into itself.

```
lemma state˙trans˙fun: assumes A1: n ∈ nat and A2: F: n → (X→X)
  shows StateTransfFunN(X,F,n): X×succ(n) → X×succ(n)
proof -
  { fix s assume A3: s ∈ X×succ(n)
    let x = fst(s)
    let k = snd(s)
    let S = StateTransfFunNMeta(F,n,s)
    from A3 have T: x ∈ X  k ∈ succ(n) and ⟨x,k⟩ = s by auto
    { assume A4: k ∈ n
      with A1 have succ(k) ∈ succ(n) using succ˙ineq by simp
      with A2 T A4 have S ∈ X×succ(n)
 using apply˙funtype StateTransfFunNMeta˙def by simp }
    with A2 A3 T have S ∈ X×succ(n)
      using apply˙funtype StateTransfFunNMeta˙def by auto
  } then have ∀s ∈ X×succ(n). StateTransfFunNMeta(F,n,s) ∈ X×succ(n)
    by simp
  then have
    −⟨s, StateTransfFunNMeta(F,n,s)⟩. s ∈ X×succ(n)″ : X×succ(n) → X×succ(n)
    by (rule ZF˙fun˙from˙total)
  then show StateTransfFunN(X,F,n): X×succ(n) → X×succ(n)
    using StateTransfFunN˙def by simp
qed
```

We can apply fin˙indseq˙props to the sequence used in the definition of InductiveSeqVarFN to get the properties of the sequence of states generated by the StateTransfFunN.

```
lemma states˙seq˙props:
  assumes A1: n ∈ nat and A2: F: n → (X→X) and A3: x∈X and
  A4: b = StatesSeq(x,X,F,n)
  shows
  b : succ(n) → X×succ(n)
  b(0) = ⟨x,0⟩
```

```
  ∀k ∈ succ(n). snd(b(k)) = k
  ∀k∈n. b(succ(k)) = ⟨F(k)(fst(b(k))), succ(k)⟩
proof -
  let f = StateTransfFunN(X,F,n)
  from A1 A2 have I: f : X×succ(n) → X×succ(n)
    using state˙trans˙fun by simp
  moreover from A1 A3 have II: ⟨x,0⟩ ∈ X×succ(n)
    using empty˙in˙every˙succ by simp
  moreover note A1
  moreover from A4 have III: b = InductiveSequenceN(⟨x,0⟩,f,n)
    using StatesSeq˙def by simp
  ultimately show IV: b : succ(n) → X×succ(n)
    by (rule fin˙indseq˙props)
  from I II A1 III show V: b(0) = ⟨x,0⟩
    by (rule fin˙indseq˙props)
  from I II A1 III have VI: ∀k∈n. b(succ(k)) = f(b(k))
    by (rule fin˙indseq˙props)
  { fix k
    note I
    moreover
    assume A5: k ∈ n hence k ∈ succ(n) by auto
    with IV have b(k) ∈  X×succ(n) using apply˙funtype by simp
    moreover have f = −⟨s, StateTransfFunNMeta(F,n,s)⟩. s ∈ X×succ(n)˝
      using StateTransfFunN˙def by simp
    ultimately have f(b(k)) =  StateTransfFunNMeta(F,n,b(k))
      by (rule ZF˙fun˙from˙tot˙val)
  } then have VII: ∀k ∈ n. f(b(k)) =  StateTransfFunNMeta(F,n,b(k))
    by simp
  { fix k assume A5: k ∈ succ(n)
    note A1 A5
    moreover from V have  snd(b(0)) = 0 by simp
    moreover from VI VII have
      ∀j∈n. snd(b(j)) = j ⟶ snd(b(succ(j))) = succ(j)
      using StateTransfFunNMeta˙def by auto
    ultimately have snd(b(k)) = k by (rule fin˙nat˙ind)
  } then show ∀k ∈ succ(n). snd(b(k)) = k by simp
  with VI VII show ∀k∈n. b(succ(k)) = ⟨F(k)(fst(b(k))), succ(k)⟩
    using StateTransfFunNMeta˙def by auto
qed
```

Basic properties of sequences defined by equation $x_{n+1} = f_n(x_n)$.

```
theorem fin˙indseq˙var˙f˙props:
  assumes A1: n ∈ nat and A2: x∈X and A3: F: n → (X→X) and
  A4: a = InductiveSeqVarFN(x,X,F,n)
  shows
  a: succ(n) → X
  a(0) = x
  ∀k∈n. a(succ(k)) = F(k)(a(k))
proof -
```

```
  let f = StateTransfFunN(X,F,n)
  let b = StatesSeq(x,X,F,n)
  from A1 A2 A3 have b : succ(n) → X×succ(n)
    using states˙seq˙props by simp
  then have ∀k ∈ succ(n). b(k) ∈ X×succ(n)
    using apply˙funtype by simp
  hence ∀k ∈ succ(n). fst(b(k)) ∈ X by auto
  then have I: −⟨k,fst(b(k))⟩. k ∈ succ(n)″ : succ(n) → X
    by (rule ZF˙fun˙from˙total)
  with A4 show II: a: succ(n) → X using InductiveSeqVarFN˙def
    by simp
  moreover from A1 have 0 ∈ succ(n) using empty˙in˙every˙succ
    by simp
  moreover from A4 have III:
    a = −⟨k,fst(StatesSeq(x,X,F,n)(k))⟩. k ∈ succ(n)″
    using InductiveSeqVarFN˙def by simp
  ultimately have a(0) = fst(b(0))
    by (rule ZF˙fun˙from˙tot˙val)
  with A1 A2 A3 show a(0) = x using states˙seq˙props by auto
  { fix k
    assume A5: k ∈ n
    with A1 have T1: succ(k) ∈ succ(n) and T2: k ∈ succ(n)
      using succ˙ineq by auto
    from II T1 III have a(succ(k)) = fst(b(succ(k)))
      by (rule ZF˙fun˙from˙tot˙val)
    with A1 A2 A3 A5 have a(succ(k)) = F(k)(fst(b(k)))
      using states˙seq˙props by simp
    moreover from II T2 III have a(k) = fst(b(k))
      by (rule ZF˙fun˙from˙tot˙val)
    ultimately have a(succ(k)) =  F(k)(a(k))
      by simp
  } then show ∀k∈n. a(succ(k)) = F(k)(a(k))
    by simp
qed
```

A consistency condition: if we make the sequence of generating functions shorter, then we get a shorter inductive sequence with the same values as in the original sequence.

```
lemma fin˙indseq˙var˙f˙restrict: assumes
  A1: n ∈ nat  i ∈ nat  x∈X  F: n → (X→X)   G: i → (X→X)
  and A2: i ⊆ n and  A3: ∀j∈i. G(j) = F(j) and A4: k ∈ succ(i)
  shows InductiveSeqVarFN(x,X,G,i)(k) = InductiveSeqVarFN(x,X,F,n)(k)
proof -
  let a = InductiveSeqVarFN(x,X,F,n)
  let b = InductiveSeqVarFN(x,X,G,i)
  from A1 A4 have i ∈ nat  k ∈ succ(i) by auto
  moreover from A1 have b(0) = a(0)
    using fin˙indseq˙var˙f˙props by simp
  moreover from A1 A2 A3 have
```

```
    ∀j∈i. b(j) = a(j) ⟶ b(succ(j)) = a(succ(j))
    using fin˙indseq˙var˙f˙props by auto
  ultimately show b(k) = a(k)
    by (rule fin˙nat˙ind)
qed
```

end

20 Folding in ZF

theory Fold˙ZF imports InductiveSeq˙ZF

begin

Suppose we have a binary operation $P : X \times X \to X$ written multiplicatively as $P\langle x, y\rangle = x \cdot y$. In informal mathematics we can take a sequence $\{x_k\}_{k\in 0..n}$ of elements of X and consider the product $x_0 \cdot x_1 \cdot .. \cdot x_n$. To do the same thing in formalized mathematics we have to define precisely what is meant by that "$\cdot .. \cdot$". The definitition we want to use is based on the notion of sequence defined by induction discussed in InductiveSeq˙ZF. We don't really want to derive the terminology for this from the word "product" as that would tie it conceptually to the multiplicative notation. This would be awkward when we want to reuse the same notions to talk about sums like $x_0 + x_1 + .. + x_n$. In functional programming there is something called "fold". Namely for a function f, initial point a and list $[b, c, d]$ the expression fold(f, a, [b,c,d]) is defined to be f(f(f(a,b),c),d) (in Haskell something like this is called foldl). If we write f in multiplicative notation we get $a \cdot b \cdot c \cdot d$, so this is exactly what we need. The notion of folds in functional programming is actually much more general that what we need here (not that I know anything about that). In this theory file we just make a slight generalization and talk about folding a list with a binary operation $f : X \times Y \to X$ with X not necessarily the same as Y.

20.1 Folding in ZF

Suppose we have a binary operation $f : X \times Y \to X$. Then every $y \in Y$ defines a transformation of X defined by $T_y(x) = f\langle x, y\rangle$. In IsarMathLib such transformation is called as Fix2ndVar(f,y). Using this notion, given a function $f : X \times Y \to X$ and a sequence $y = \{y_k\}_{k\in N}$ of elements of X we can get a sequence of transformations of X. This is defined in Seq2TransSeq below. Then we use that sequence of tranformations to define the sequence of partial folds (called FoldSeq) by means of InductiveSeqVarFN (defined in In-

ductiveSeq˙ZF theory) which implements the inductive sequence determined by a starting point and a sequence of transformations. Finally, we define the fold of a sequence as the last element of the sequence of the partial folds.

Definition that specifies how to convert a sequence a of elements of Y into a sequence of transformations of X, given a binary operation $f : X \times Y \to X$.

```
definition
  Seq2TrSeq(f,a) ≡ –⟨k,Fix2ndVar(f,a(k))⟩. k ∈ domain(a)″
```

Definition of a sequence of partial folds.

```
definition
  FoldSeq(f,x,a) ≡
  InductiveSeqVarFN(x,fstdom(f),Seq2TrSeq(f,a),domain(a))
```

Definition of a fold.

```
definition
  Fold(f,x,a) ≡ Last(FoldSeq(f,x,a))
```

If X is a set with a binary operation $f : X \times Y \to X$ then Seq2TransSeqN(f,a) converts a sequence a of elements of Y into the sequence of corresponding transformations of X.

```
lemma seq2trans˙seq˙props:
  assumes A1: n ∈ nat and A2: f : X×Y → X and A3: a:n→Y and
  A4: T = Seq2TrSeq(f,a)
  shows
  T : n → (X→X) and
  ∀k∈n. ∀x∈X. (T(k))(x) = f⟨x,a(k)⟩
proof -
  from ‹a:n→Y› have D: domain(a) = n using func1˙1˙L1 by simp
  with A2 A3 A4 show T : n → (X→X)
    using apply˙funtype fix˙2nd˙var˙fun ZF˙fun˙from˙total Seq2TrSeq˙def
    by simp
  with A4 D have I: ∀k ∈ n. T(k) = Fix2ndVar(f,a(k))
    using Seq2TrSeq˙def ZF˙fun˙from˙tot˙val0 by simp
  { fix k fix x assume A5: k∈n  x∈X
    with A1 A3 have a(k) ∈ Y using apply˙funtype
      by auto
    with A2 A5 I have (T(k))(x) = f⟨x,a(k)⟩
      using fix˙var˙val by simp
  } thus ∀k∈n. ∀x∈X. (T(k))(x) = f⟨x,a(k)⟩
    by simp
qed
```

Basic properties of the sequence of partial folds of a sequence $a = \{y_k\}_{k\in\{0,..,n\}}$.

```
theorem fold˙seq˙props:
  assumes A1: n ∈ nat and A2: f : X×Y → X and
  A3: y:n→Y and A4: x∈X and A5: Y≠0 and
```

```
  A6: F = FoldSeq(f,x,y)
  shows
  F: succ(n) → X
  F(0) = x and
  ∀k∈n. F(succ(k)) = f⟨F(k), y(k)⟩
proof -
  let T = Seq2TrSeq(f,y)
  from A1 A3 have D: domain(y) = n
    using func1˙1˙L1 by simp
  from ‹f : X×Y → X› ‹Y≠0› have I: fstdom(f) = X
    using fstdomdef by simp
  with A1 A2 A3 A4 A6 D show
    II: F: succ(n) → X  and F(0) = x
    using seq2trans˙seq˙props FoldSeq˙def fin˙indseq˙var˙f˙props
    by auto
  from A1 A2 A3 A4 A6 I D have ∀k∈n. F(succ(k)) = T(k)(F(k))
    using  seq2trans˙seq˙props FoldSeq˙def fin˙indseq˙var˙f˙props
    by simp
  moreover
  { fix k assume A5: k∈n hence k ∈ succ(n) by auto
    with A1 A2 A3 II A5 have (T(k))(F(k)) = f⟨F(k),y(k)⟩
      using apply˙funtype seq2trans˙seq˙props by simp }
  ultimately show ∀k∈n. F(succ(k)) = f⟨F(k), y(k)⟩
    by simp
qed
```

A consistency condition: if we make the list shorter, then we get a shorter sequence of partial folds with the same values as in the original sequence. This can be proven as a special case of fin˙indseq˙var˙f˙restrict but a proof using fold˙seq˙props and induction turns out to be shorter.

```
lemma foldseq˙restrict: assumes
  n ∈ nat   k ∈ succ(n) and
  i ∈ nat  f : X×Y → X  a : n → Y  b : i → Y and
  n ⊆ i   ∀j ∈ n. b(j) = a(j)   x ∈ X   Y ≠ 0
  shows FoldSeq(f,x,b)(k) = FoldSeq(f,x,a)(k)
proof -
  let P = FoldSeq(f,x,a)
  let Q = FoldSeq(f,x,b)
  from assms have
    n ∈ nat   k ∈ succ(n)
    Q(0) = P(0) and
    ∀j ∈ n. Q(j) = P(j) ⟶ Q(succ(j)) = P(succ(j))
    using fold˙seq˙props by auto
  then show  Q(k) = P(k) by (rule fin˙nat˙ind)
qed
```

A special case of foldseq˙restrict when the longer sequence is created from the shorter one by appending one element.

```
corollary fold˙seq˙append:
```

```
  assumes n ∈ nat   f : X×Y → X   a:n → Y and
  x∈X   k ∈ succ(n)   y∈Y
  shows FoldSeq(f,x,Append(a,y))(k) = FoldSeq(f,x,a)(k)
proof -
  let b = Append(a,y)
  from assms have b : succ(n) → Y  ∀j ∈ n. b(j) = a(j)
    using append˙props by auto
  with assms show thesis using foldseq˙restrict by blast
qed
```

What we really will be using is the notion of the fold of a sequence, which we define as the last element of (inductively defined) sequence of partial folds. The next theorem lists some properties of the product of the fold operation.

```
theorem fold˙props:
  assumes A1: n ∈ nat and
  A2: f : X×Y → X  a:n → Y  x∈X  Y≠0
  shows
  Fold(f,x,a) =  FoldSeq(f,x,a)(n) and
  Fold(f,x,a) ∈ X
proof -
  from assms have  FoldSeq(f,x,a) : succ(n) → X
    using  fold˙seq˙props by simp
  with A1 show
    Fold(f,x,a) =  FoldSeq(f,x,a)(n) and Fold(f,x,a) ∈ X
    using last˙seq˙elem apply˙funtype Fold˙def by auto
qed
```

A corner case: what happens when we fold an empty list?

```
theorem fold˙empty: assumes A1: f : X×Y → X and
  A2: a:0→Y  x∈X  Y≠0
  shows Fold(f,x,a) = x
proof -
  let F = FoldSeq(f,x,a)
  from assms have I:
    0 ∈ nat  f : X×Y → X  a:0→Y  x∈X  Y≠0
    by auto
  then have Fold(f,x,a) = F(0) by (rule fold˙props)
  moreover
  from I have
    0 ∈ nat  f : X×Y → X  a:0→Y  x∈X  Y≠0 and
    F = FoldSeq(f,x,a) by auto
  then have F(0) = x by (rule fold˙seq˙props)
  ultimately show Fold(f,x,a) = x by simp
qed
```

The next theorem tells us what happens to the fold of a sequence when we add one more element to it.

```
theorem fold˙append:
```

```
  assumes A1: n ∈ nat and  A2: f : X×Y → X and
  A3: a:n→Y and A4: x∈X and A5: y∈Y
  shows
  FoldSeq(f,x,Append(a,y))(n) = Fold(f,x,a) and
  Fold(f,x,Append(a,y)) = f⟨Fold(f,x,a), y⟩
proof -
  let b = Append(a,y)
  let P = FoldSeq(f,x,b)
  from A5 have I: Y ≠ 0 by auto
  with assms show thesis1: P(n) = Fold(f,x,a)
    using fold˙seq˙append fold˙props by simp
  from assms I have II:
    succ(n) ∈ nat   f : X×Y → X
    b : succ(n) → Y   x∈X  Y ≠ 0
    P = FoldSeq(f,x,b)
    using append˙props by auto
  then have
    ∀k ∈ succ(n). P(succ(k)) =  f⟨P(k), b(k)⟩
    by (rule fold˙seq˙props)
  with A3 A5 thesis1 have P(succ(n)) =  f⟨ Fold(f,x,a), y⟩
    using append˙props by auto
  moreover
  from II have P : succ(succ(n)) → X
    by (rule fold˙seq˙props)
  then have Fold(f,x,b) = P(succ(n))
    using last˙seq˙elem Fold˙def by simp
  ultimately show Fold(f,x,Append(a,y)) = f⟨Fold(f,x,a), y⟩
    by simp
qed


end
```

21 Partitions of sets

theory Partitions˙ZF imports Finite˙ZF FiniteSeq˙ZF

begin

It is a common trick in proofs that we divide a set into non-overlapping subsets. The first case is when we split the set into two nonempty disjoint sets. Here this is modeled as an ordered pair of sets and the set of such divisions of set X is called Bisections(X). The second variation on this theme is a set-valued function (aren't they all in ZF?) whose values are nonempty and mutually disjoint.

21.1 Bisections

This section is about dividing sets into two non-overlapping subsets.

The set of bisections of a given set A is a set of pairs of nonempty subsets of A that do not overlap and their union is equal to A.

```
definition
  Bisections(X) = –p ∈ Pow(X)×Pow(X).
  fst(p)≠0 ∧ snd(p)≠0 ∧ fst(p)∩snd(p) = 0 ∧ fst(p)∪snd(p) = X″
```

Properties of bisections.

```
lemma bisec˙props: assumes ⟨A,B⟩ ∈ Bisections(X) shows
  A≠0  B≠0 A ⊆ X  B ⊆ X  A ∩ B = 0  A ∪ B = X  X ≠ 0
  using assms Bisections˙def by auto
```

Kind of inverse of bisec˙props: a pair of nonempty disjoint sets form a bisection of their union.

```
lemma is˙bisec:
  assumes A≠0  B≠0  A ∩ B = 0
  shows ⟨A,B⟩ ∈ Bisections(A∪B) using assms Bisections˙def
  by auto
```

Bisection of X is a pair of subsets of X.

```
lemma bisec˙is˙pair: assumes Q ∈ Bisections(X)
  shows Q = ⟨fst(Q), snd(Q)⟩
  using assms Bisections˙def by auto
```

The set of bisections of the empty set is empty.

```
lemma bisec˙empty: shows Bisections(0) = 0
  using Bisections˙def by auto
```

The next lemma shows what can we say about bisections of a set with another element added.

```
lemma bisec˙add˙point:
  assumes A1: x ∉ X and A2: ⟨A,B⟩ ∈ Bisections(X ∪ –x″)
  shows (A = –x″ ∨ B = –x″) ∨ (⟨A - –x″, B - –x″⟩ ∈ Bisections(X))
  proof -
    { assume A ≠ –x″ and B ≠ –x″
      with A2 have A - –x″ ≠ 0 and B - –x″ ≠ 0
 using singl˙diff˙empty Bisections˙def
 by auto
      moreover have (A - –x″) ∪ (B - –x″) = X
      proof -
 have (A - –x″) ∪ (B - –x″) = (A ∪ B) - –x″
   by auto
 also from assms have (A ∪ B) - –x″ = X
   using Bisections˙def by auto
```

```
 finally show thesis by simp
      qed
      moreover from A2 have (A - –x″) ∩ (B - –x″) = 0
 using Bisections˙def by auto
      ultimately have ⟨A - –x″, B - –x″⟩ ∈ Bisections(X)
 using Bisections˙def by auto
    } thus thesis by auto
qed
```

A continuation of the lemma bisec˙add˙point that refines the case when the pair with removed point bisects the original set.

```
lemma bisec˙add˙point˙case3:
  assumes A1: ⟨A,B⟩ ∈ Bisections(X ∪ –x″)
  and A2: ⟨A - –x″, B - –x″⟩ ∈ Bisections(X)
  shows
  (⟨A, B - –x″⟩ ∈ Bisections(X) ∧ x ∈ B) ∨
  (⟨A - –x″, B⟩ ∈ Bisections(X) ∧ x ∈ A)
proof -
  from A1 have x ∈ A ∪ B
    using Bisections˙def by auto
  hence x∈A ∨ x∈B by simp
  from A1 have A - –x″ = A ∨ B - –x″ = B
    using Bisections˙def by auto
  moreover
  { assume A - –x″ = A
    with A2 ‹x ∈ A ∪ B› have
      ⟨A, B - –x″⟩ ∈ Bisections(X) ∧ x ∈ B
      using singl˙diff˙eq by simp }
  moreover
  { assume B - –x″ = B
    with A2 ‹x ∈ A ∪ B› have
      ⟨A - –x″, B⟩ ∈ Bisections(X) ∧ x ∈ A
      using singl˙diff˙eq by simp }
  ultimately show thesis by auto
qed
```

Another lemma about bisecting a set with an added point.

```
lemma point˙set˙bisec:
  assumes A1: x ∉ X and A2: ⟨–x″, A⟩ ∈ Bisections(X ∪ –x″)
  shows A = X and X ≠ 0
proof -
  from A2 have A ⊆ X using Bisections˙def by auto
  moreover
  { fix a assume a∈X
    with A2 have a ∈ –x″ ∪ A using Bisections˙def by simp
    with A1 ‹a∈X› have a ∈ A by auto }
  ultimately show A = X by auto
  with A2 show X ≠ 0 using Bisections˙def by simp
qed
```

Yet another lemma about bisecting a set with an added point, very similar to point˙set˙bisec with almost the same proof.

```
lemma set˙point˙bisec:
  assumes A1: x ∉ X and A2: ⟨A, –x″⟩ ∈ Bisections(X ∪ –x″)
  shows A = X and X ≠ 0
proof -
  from A2 have A ⊆ X using Bisections˙def by auto
  moreover
  { fix a assume a∈X
    with A2 have a ∈ A ∪ –x″ using Bisections˙def by simp
    with A1 ‹a∈X› have a ∈ A by auto }
  ultimately show A = X by auto
  with A2 show X ≠ 0 using Bisections˙def by simp
qed
```

If a pair of sets bisects a finite set, then both elements of the pair are finite.

```
lemma bisect˙fin:
  assumes A1: A ∈ FinPow(X) and A2: Q ∈ Bisections(A)
  shows fst(Q) ∈ FinPow(X) and snd(Q) ∈ FinPow(X)
proof -
  from A2 have ⟨fst(Q), snd(Q)⟩ ∈ Bisections(A)
    using bisec˙is˙pair by simp
  then have fst(Q) ⊆ A and snd(Q) ⊆ A
    using bisec˙props by auto
  with A1 show fst(Q) ∈ FinPow(X) and snd(Q) ∈ FinPow(X)
    using FinPow˙def subset˙Finite by auto
qed
```

21.2 Partitions

This sections covers the situation when we have an arbitrary number of sets we want to partition into.

We define a notion of a partition as a set valued function such that the values for different arguments are disjoint. The name is derived from the fact that such function "partitions" the union of its arguments. Please let me know if you have a better idea for a name for such notion. We would prefer to say "is a partition", but that reserves the letter "a" as a keyword(?) which causes problems.

```
definition
  Partition (˙ –is partition″ [90] 91) where
  P –is partition″ ≡ ∀x ∈ domain(P).
  P(x) ≠ 0 ∧ (∀y ∈ domain(P). x≠y ⟶ P(x) ∩ P(y) = 0)
```

A fact about lists of mutually disjoint sets.

```
lemma list˙partition: assumes A1: n ∈ nat and
  A2: a : succ(n) → X   a –is partition″
```

```
  shows (⋃i∈n. a(i)) ∩ a(n) = 0
proof -
  { assume (⋃i∈n. a(i)) ∩ a(n) ≠ 0
    then have ∃x. x ∈ (⋃i∈n. a(i)) ∩ a(n)
      by (rule nonempty˙has˙element)
    then obtain x where x ∈ (⋃i∈n. a(i)) and  I: x ∈ a(n)
      by auto
    then obtain i where i ∈ n and x ∈ a(i) by auto
    with A2 I have False
      using mem˙imp˙not˙eq func1˙1˙L1 Partition˙def
      by auto
  } thus thesis by auto
qed
```

We can turn every injection into a partition.

```
lemma inj˙partition:
  assumes A1: b ∈ inj(X,Y)
  shows
  ∀x ∈ X. −⟨x, −b(x)″⟩. x ∈ X″(x) = −b(x)″ and
  −⟨x, −b(x)″⟩. x ∈ X″ −is partition″
proof -
  let p = −⟨x, −b(x)″⟩. x ∈ X″
  { fix x assume x ∈ X
    from A1 have b : X → Y using inj˙def
      by simp
    with ‹x ∈ X› have −b(x)″ ∈ Pow(Y)
       using apply˙funtype by simp
  } hence ∀x ∈ X. −b(x)″ ∈ Pow(Y) by simp
  then have p : X → Pow(Y) using ZF˙fun˙from˙total
    by simp
  then have domain(p) = X using func1˙1˙L1
    by simp
  from ‹p : X → Pow(Y)› show I: ∀x ∈ X. p(x) = −b(x)″
    using ZF˙fun˙from˙tot˙val0 by simp
  { fix x assume x ∈ X
    with I have p(x) = −b(x)″ by simp
    hence p(x) ≠ 0 by simp
    moreover
    { fix t assume t ∈ X and x ≠ t
      with A1 ‹x ∈ X› have b(x) ≠ b(t) using inj˙def
 by auto
      with I ‹x∈X› ‹t ∈ X› have p(x) ∩ p(t) = 0
 by auto }
    ultimately have
      p(x) ≠ 0 ∧ (∀t ∈ X. x≠t ⟶ p(x) ∩ p(t) = 0)
      by simp
  } with ‹domain(p) = X› show −⟨x, −b(x)″⟩. x ∈ X″ −is partition″
    using Partition˙def by simp
qed
```

end

22 Enumerations

theory Enumeration˙ZF imports NatOrder˙ZF FiniteSeq˙ZF FinOrd˙ZF

begin

Suppose r is a linear order on a set A that has n elements, where $n \in \mathbb{N}$. In the FinOrd˙ZF theory we prove a theorem stating that there is a unique order isomorphism between $n = \{0, 1, .., n-1\}$ (with natural order) and A. Another way of stating that is that there is a unique way of counting the elements of A in the order increasing according to relation r. Yet another way of stating the same thing is that there is a unique sorted list of elements of A. We will call this list the Enumeration of A.

22.1 Enumerations: definition and notation

In this section we introduce the notion of enumeration and define a proof context (a "locale" in Isabelle terms) that sets up the notation for writing about enumarations.

We define enumeration as the only order isomorphism beween a set A and the number of its elements. We are using the formula $\bigcup\{x\} = x$ to extract the only element from a singleton. Le is the (natural) order on natural numbers, defined is Nat˙ZF theory in the standard Isabelle library.

definition
Enumeration(A,r) $\equiv \bigcup$ ord˙iso(—A—,Le,A,r)

To set up the notation we define a locale enums. In this locale we will assume that r is a linear order on some set X. In most applications this set will be just the set of natural numbers. Standard Isabelle uses $\leq$ to denote the "less or equal" relation on natural numbers. We will use the $\leq$ symbol to denote the relation r. Those two symbols usually look the same in the presentation, but they are different in the source.To shorten the notation the enumeration Enumeration(A,r) will be denoted as $\sigma(A)$. Similarly as in the Semigroup theory we will write $a \hookleftarrow x$ for the result of appending an element x to the finite sequence (list) a. Finally, $a \sqcup b$ will denote the concatenation of the lists a and b.

locale enums =

```
  fixes X r
  assumes linord: IsLinOrder(X,r)

  fixes ler (infix ≤ 70)
  defines ler˙def[simp]: x ≤ y ≡ ⟨x,y⟩ ∈ r

  fixes σ
  defines σ˙def [simp]: σ(A) ≡ Enumeration(A,r)

  fixes append (infix ↩ 72)
  defines append˙def[simp]: a ↩ x ≡ Append(a,x)

  fixes concat (infixl ⊔ 69)
  defines concat˙def[simp]: a ⊔ b ≡ Concat(a,b)
```

22.2 Properties of enumerations

In this section we prove basic facts about enumerations.

A special case of the existence and uniqueess of the order isomorphism for finite sets when the first set is a natural number.

```
lemma (in enums) ord˙iso˙nat˙fin:
  assumes A ∈ FinPow(X) and n ∈ nat and A ≈ n
  shows ∃!f. f ∈ ord˙iso(n,Le,A,r)
  using assms NatOrder˙ZF˙1˙L2 linord nat˙finpow˙nat
    fin˙ord˙iso˙ex˙uniq by simp
```

An enumeration is an order isomorhism, a bijection, and a list.

```
lemma (in enums) enum˙props: assumes A ∈ FinPow(X)
  shows
  σ(A) ∈ ord˙iso(—A—,Le, A,r)
  σ(A) ∈ bij(—A—,A)
  σ(A) : —A— → A
proof -
  from assms have
    IsLinOrder(nat,Le) and —A— ∈ FinPow(nat) and  A ≈ —A—
    using NatOrder˙ZF˙1˙L2 card˙fin˙is˙nat nat˙finpow˙nat
    by auto
  with assms show σ(A) ∈ ord˙iso(—A—,Le, A,r)
    using linord fin˙ord˙iso˙ex˙uniq singleton˙extract
      Enumeration˙def by simp
  then show σ(A) ∈ bij(—A—,A) and σ(A) : —A— → A
    using ord˙iso˙def bij˙def surj˙def
    by auto
qed
```

A corollary from enum˙props. Could have been attached as another assertion, but this slows down verification of some other proofs.

```
lemma (in enums) enum˙fun: assumes A ∈ FinPow(X)
  shows σ(A) : —A— → X
proof -
  from assms have σ(A) : —A— → A and A⊆X
    using enum˙props  FinPow˙def by auto
  then show σ(A) : —A— → X by (rule func1˙1˙L1B)
qed
```

If a list is an order isomorphism then it must be the enumeration.

```
lemma (in enums) ord˙iso˙enum: assumes A1: A ∈ FinPow(X) and
  A2: n ∈ nat and A3: f ∈ ord˙iso(n,Le,A,r)
  shows f = σ(A)
proof -
  from A3 have n ≈ A using ord˙iso˙def eqpoll˙def
    by auto
  then have A ≈ n by (rule eqpoll˙sym)
  with A1 A2 have ∃!f. f ∈ ord˙iso(n,Le,A,r)
    using ord˙iso˙nat˙fin by simp
  with assms ‹A ≈ n› show f = σ(A)
    using enum˙props card˙card by blast
qed
```

What is the enumeration of the empty set?

```
lemma (in enums) empty˙enum: shows σ(0) = 0
proof -
  have
    0 ∈ FinPow(X) and 0 ∈ nat and 0 ∈ ord˙iso(0,Le,0,r)
    using empty˙in˙finpow empty˙ord˙iso˙empty
    by auto
  then show σ(0) = 0 using ord˙iso˙enum
    by blast
qed
```

Adding a new maximum to a set appends it to the enumeration.

```
lemma (in enums) enum˙append:
  assumes A1: A ∈ FinPow(X) and A2: b ∈ X-A and
  A3: ∀a∈A. a≤b
  shows  σ(A ∪ –b″) = σ(A)↩ b
proof -
  let f = σ(A) ∪ –⟨—A—,b⟩″
  from A1 have —A— ∈ nat using card˙fin˙is˙nat
    by simp
  from A1 A2 have A ∪ –b″ ∈ FinPow(X)
    using singleton˙in˙finpow union˙finpow by simp
  moreover from this have —A ∪ –b″— ∈ nat
    using card˙fin˙is˙nat by simp
  moreover have f ∈ ord˙iso(—A ∪ –b″— , Le, A ∪ –b″ ,r)
  proof -
    from A1 A2 have
```

```
      σ(A) ∈ ord˙iso(—A—,Le, A,r) and
      —A— ∉ —A— and b ∉ A
      using enum˙props  mem˙not˙refl by auto
    moreover from ‹—A— ∈ nat› have
      ∀ k ∈ —A—. ⟨k, —A—⟩ ∈ Le
      using elem˙nat˙is˙nat by blast
    moreover from A3 have ∀ a∈A. ⟨a,b⟩ ∈ r by simp
    moreover have antisym(Le) and antisym(r)
      using linord NatOrder˙ZF˙1˙L2 IsLinOrder˙def by auto
    moreover
    from  A2 ‹—A— ∈ nat› have
      ⟨—A—,—A—⟩ ∈ Le and  ⟨b,b⟩ ∈ r
      using linord NatOrder˙ZF˙1˙L2 IsLinOrder˙def
 total˙is˙refl refl˙def by auto
    hence ⟨—A—,—A—⟩ ∈ Le ⟷ ⟨b,b⟩ ∈ r by simp
    ultimately have f ∈ ord˙iso(—A— ∪ ——A—″ , Le, A ∪ –b″ ,r)
      by (rule ord˙iso˙extend)
    with A1 A2 show f ∈ ord˙iso(—A ∪ –b″— , Le, A ∪ –b″ ,r)
      using card˙fin˙add˙one by simp
  qed
  ultimately have f = σ(A ∪ –b″)
    using ord˙iso˙enum by simp
  moreover have σ(A)↩ b = f
  proof -
    have σ(A)↩ b = σ(A) ∪ –⟨domain(σ(A)),b⟩″
      using Append˙def by simp
    moreover from A1 have domain(σ(A)) = —A—
      using enum˙props func1˙1˙L1 by blast
    ultimately show σ(A)↩ b = f by simp
  qed
  ultimately show σ(A ∪ –b″) = σ(A)↩ b by simp
qed
```

What is the enumeration of a singleton?

```
lemma (in enums) enum˙singleton:
  assumes A1: x∈X shows σ(–x″): 1 → X and σ(–x″)(0) = x
  proof -
    from A1 have
      0 ∈ FinPow(X) and x ∈ (X - 0) and ∀ a∈0. a≤x
      using empty˙in˙finpow by auto
    then have σ(0 ∪ –x″) = σ(0)↩ x by (rule enum˙append)
    with A1 show σ(–x″): 1 → X and σ(–x″)(0) = x
      using empty˙enum empty˙append1 by auto
qed


end
```

23 Semigroups

theory Semigroup˙ZF imports Partitions˙ZF Fold˙ZF Enumeration˙ZF

begin

It seems that the minimal setup needed to talk about a product of a sequence is a set with a binary operation. Such object is called "magma". However, interesting properties show up when the binary operation is associative and such alebraic structure is called a semigroup. In this theory file we define and study sequences of partial products of sequences of magma and semigroup elements.

23.1 Products of sequences of semigroup elements

Semigroup is a a magma in which the binary operation is associative. In this section we mostly study the products of sequences of elements of semigroup. The goal is to establish the fact that taking the product of a sequence is distributive with respect to concatenation of sequences, i.e for two sequences a, b of the semigroup elements we have $\prod(a \sqcup b) = (\prod a) \cdot (\prod b)$, where "$a \sqcup b$" is concatenation of a and b ($a{+}{+}b$ in Haskell notation). Less formally, we want to show that we can discard parantheses in expressions of the form $(a_0 \cdot a_1 \cdot .. \cdot a_n) \cdot (b_0 \cdot .. \cdot b_k)$.

First we define a notion similar to Fold, except that that the initial element of the fold is given by the first element of sequence. By analogy with Haskell fold we call that Fold1

definition
  Fold1(f,a) ≡ Fold(f,a(0),Tail(a))

The definition of the semigr0 context below introduces notation for writing about finite sequences and semigroup products. In the context we fix the carrier and denote it G. The binary operation on G is called f. All theorems proven in the context semigr0 will implicitly assume that f is an associative operation on G. We will use multiplicative notation for the semigroup operation. The product of a sequence a is denoted $\prod a$. We will write $a \hookleftarrow x$ for the result of appending an element x to the finite sequence (list) a. This is a bit nonstandard, but I don't have a better idea for the "append" notation. Finally, $a \sqcup b$ will denote the concatenation of the lists a and b.

locale semigr0 =

  fixes G f

  assumes assoc˙assum: f –is associative on″ G

  fixes prod (infixl · 72)

defines prod˙def [simp]: x · y ≡ f⟨x,y⟩

fixes seqprod (∏ ˙ 71)
defines seqprod˙def [simp]: ∏ a ≡ Fold1(f,a)

fixes append (infix ↩ 72)
defines append˙def [simp]: a ↩ x ≡ Append(a,x)

fixes concat (infixl ⊔ 69)
defines concat˙def [simp]: a ⊔ b ≡ Concat(a,b)

The next lemma shows our assumption on the associativity of the semigroup operation in the notation defined in in the semigr0 context.

lemma (in semigr0) semigr˙assoc: assumes x ∈ G y ∈ G z ∈ G
shows x·y·z = x·(y·z)
using assms assoc˙assum IsAssociative˙def by simp

In the way we define associativity the assumption that f is associative on G also implies that it is a binary operation on X.

lemma (in semigr0) semigr˙binop: shows f : G×G → G
using assoc˙assum IsAssociative˙def by simp

Semigroup operation is closed.

lemma (in semigr0) semigr˙closed:
assumes a∈G b∈G shows a·b ∈ G
using assms semigr˙binop apply˙funtype by simp

Lemma append˙1elem written in the notation used in the semigr0 context.

lemma (in semigr0) append˙1elem˙nice:
assumes n ∈ nat and a: n → X and b : 1 → X
shows a ⊔ b = a ↩ b(0)
using assms append˙1elem by simp

Lemma concat˙init˙last˙elem rewritten in the notation used in the semigr0 context.

lemma (in semigr0) concat˙init˙last:
assumes n ∈ nat k ∈ nat and
a: n → X and b : succ(k) → X
shows (a ⊔ Init(b)) ↩ b(k) = a ⊔ b
using assms concat˙init˙last˙elem by simp

The product of semigroup (actually, magma – we don't need associativity for this) elements is in the semigroup.

lemma (in semigr0) prod˙type:
assumes n ∈ nat and a : succ(n) → G
shows (∏ a) ∈ G
proof -

```
  from assms have
    succ(n) ∈ nat  f : G×G → G  Tail(a) : n → G
    using semigr˙binop tail˙props by auto
  moreover from assms have a(0) ∈ G and G ≠ 0
    using empty˙in˙every˙succ apply˙funtype
    by auto
  ultimately show (∏ a) ∈ G using Fold1˙def fold˙props
    by simp
qed
```

What is the product of one element list?

```
lemma (in semigr0) prod˙of˙1elem: assumes A1: a: 1 → G
  shows (∏ a) = a(0)
proof -
  have f : G×G → G using semigr˙binop by simp
  moreover from A1 have Tail(a) : 0 → G using tail˙props
    by blast
  moreover from A1 have a(0) ∈ G and G ≠ 0
    using apply˙funtype by auto
  ultimately show (∏ a) =  a(0) using fold˙empty Fold1˙def
    by simp
qed
```

What happens to the product of a list when we append an element to the list?

```
lemma (in semigr0) prod˙append: assumes A1: n ∈ nat and
  A2: a : succ(n) → G and A3: x∈G
  shows (∏ a↩x) = (∏ a) · x
proof -
  from A1 A2 have I: Tail(a) : n → G  a(0) ∈ G
    using tail˙props empty˙in˙every˙succ apply˙funtype
    by auto
  from assms have (∏ a↩x) = Fold(f,a(0),Tail(a)↩x)
    using head˙of˙append tail˙append˙commute Fold1˙def
    by simp
  also from A1 A3 I have ... = (∏ a) · x
    using semigr˙binop fold˙append Fold1˙def
    by simp
  finally show thesis by simp
qed
```

The main theorem of the section: taking the product of a sequence is distributive with respect to concatenation of sequences. The proof is by induction on the length of the second list.

```
theorem (in semigr0) prod˙conc˙distr:
  assumes A1: n ∈ nat  k ∈ nat and
  A2: a : succ(n) → G   b: succ(k) → G
  shows (∏ a) · (∏ b) = ∏ (a ⊔ b)
```

```
proof -
  from A1 have k ∈ nat by simp
  moreover have ∀b ∈ succ(0) → G. (∏ a) · (∏ b) = ∏ (a ⊔ b)
  proof -
    { fix b assume A3: b : succ(0) → G
      with A1 A2 have
 succ(n) ∈ nat  a : succ(n) → G  b : 1 → G
 by auto
      then have a ⊔ b = a ↩ b(0) by (rule append˙1elem˙nice)
      with A1 A2 A3 have (∏ a) · (∏ b) = ∏ (a ⊔ b)
 using apply˙funtype prod˙append semigr˙binop prod˙of˙1elem
 by simp
    } thus thesis by simp
  qed
  moreover have ∀j ∈ nat.
    (∀b ∈ succ(j) → G. (∏ a) · (∏ b) = ∏ (a ⊔ b)) ⟶
    (∀b ∈ succ(succ(j)) → G. (∏ a) · (∏ b) = ∏ (a ⊔ b))
  proof -
    { fix j assume A4: j ∈ nat and
      A5: (∀b ∈ succ(j) → G. (∏ a) · (∏ b) = ∏ (a ⊔ b))
      { fix b assume A6: b : succ(succ(j)) → G
 let c = Init(b)
 from A4 A6 have  T: b(succ(j)) ∈ G and
   I: c : succ(j) → G and II: b = c↩b(succ(j))
   using apply˙funtype init˙props by auto
 from A1 A2 A4 A6 have
   succ(n) ∈ nat  succ(j) ∈ nat
   a : succ(n) → G  b : succ(succ(j)) → G
   by auto
 then have III: (a ⊔ c) ↩ b(succ(j)) = a ⊔ b
   by (rule concat˙init˙last)
 from A4 I T have (∏ c↩b(succ(j))) = (∏ c) · b(succ(j))
   by (rule prod˙append)
 with II have
   (∏ a) · (∏ b) = (∏ a) · ((∏ c) · b(succ(j)))
   by simp
 moreover from A1 A2 A4 T I have
   (∏ a) ∈ G  (∏ c) ∈ G  b(succ(j)) ∈ G
   using prod˙type by auto
 ultimately have
   (∏ a) · (∏ b) =  ((∏ a) · (∏ c)) · b(succ(j))
   using semigr˙assoc by auto
 with A5 I have (∏ a) · (∏ b) = (∏ (a ⊔ c))·b(succ(j))
   by simp
 moreover
 from A1 A2 A4 I have
   T1: succ(n) ∈ nat  succ(j) ∈ nat and
   a : succ(n) → G   c : succ(j) → G
   by auto
```

```
then have Concat(a,c): succ(n) #+ succ(j) → G
  by (rule concat˙props)
with A1 A4 T have
  succ(n #+ j) ∈ nat
  a ⊔ c : succ(succ(n #+j)) → G
  b(succ(j)) ∈ G
  using succ˙plus by auto
then have
  (∏ (a ⊔ c)↩b(succ(j))) = (∏ (a ⊔ c))·b(succ(j))
  by (rule prod˙append)
with III have (∏ (a ⊔ c))·b(succ(j)) = ∏ (a ⊔ b)
  by simp
ultimately have (∏ a) · (∏ b) = ∏ (a ⊔ b)
  by simp
    } hence (∀b ∈ succ(succ(j)) → G. (∏ a) · (∏ b) = ∏ (a ⊔ b))
by simp
   } thus thesis by blast
 qed
 ultimately have ∀b ∈ succ(k) → G. (∏ a) · (∏ b) = ∏ (a ⊔ b)
   by (rule ind˙on˙nat)
 with A2 show (∏ a) · (∏ b) = ∏ (a ⊔ b) by simp
qed
```

23.2 Products over sets of indices

In this section we study the properties of expressions of the form $\prod_{i\in\Lambda} a_i = a_{i_0} \cdot a_{i_1} \cdot .. \cdot a_{i-1}$, i.e. what we denote as $\prod(\Lambda,\mathrm{a})$. Λ here is a finite subset of some set X and a is a function defined on X with values in the semigroup G.

Suppose $a : X \to G$ is an indexed family of elements of a semigroup G and $\Lambda = \{i_0, i_1, .., i_{n-1}\} \subseteq \mathbb{N}$ is a finite set of indices. We want to define $\prod_{i\in\Lambda} a_i = a_{i_0} \cdot a_{i_1} \cdot .. \cdot a_{i-1}$. To do that we use the notion of Enumeration defined in the Enumeration˙ZF theory file that takes a set of indices and lists them in increasing order, thus converting it to list. Then we use the Fold1 to multiply the resulting list. Recall that in Isabelle/ZF the capital letter "O" denotes the composition of two functions (or relations).

```
definition
  SetFold(f,a,Λ,r) = Fold1(f,a O Enumeration(Λ,r))
```

For a finite subset Λ of a linearly ordered set X we will write $\sigma(\Lambda)$ to denote the enumeration of the elements of Λ, i.e. the only order isomorphism $|\Lambda| \to \Lambda$, where $|\Lambda| \in \mathbb{N}$ is the number of elements of Λ. We also define notation for taking a product over a set of indices of some sequence of semigroup elements. The product of semigroup elements over some set $\Lambda \subseteq X$ of indices of a sequence $a : X \to G$ (i.e. $\prod_{i\in\Lambda} a_i$) is denoted $\prod(\Lambda,\mathrm{a})$. In the semigr1 context we assume that a is a function defined on some linearly

ordered set X with values in the semigroup G.

```
locale semigr1 = semigr0 +

  fixes X r
  assumes linord: IsLinOrder(X,r)

  fixes a
  assumes a˙is˙fun: a : X → G

  fixes σ
  defines σ˙def [simp]: σ(A) ≡ Enumeration(A,r)

  fixes setpr (∏)
  defines setpr˙def [simp]: ∏(Λ,b) ≡ SetFold(f,b,Λ,r)
```

We can use the enums locale in the semigr0 context.

```
lemma (in semigr1) enums˙valid˙in˙semigr1: shows enums(X,r)
  using linord enums˙def by simp
```

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

```
lemma (in semigr1) setproddef:
  shows ∏(Λ,a) = ∏ (a O σ(Λ))
  using SetFold˙def by simp
```

A composition of enumeration of a nonempty finite subset of $\mathbb{N}$ with a sequence of elements of G is a nonempty list of elements of G. This implies that a product over set of a finite set of indices belongs to the (carrier of) semigroup.

```
lemma (in semigr1) setprod˙type: assumes
  A1: Λ ∈ FinPow(X) and A2: Λ≠0
  shows
  ∃n ∈ nat . —Λ— = succ(n) ∧ a O σ(Λ) : succ(n) → G
  and ∏(Λ,a) ∈ G
proof -
  from assms obtain n where n ∈ nat and —Λ— = succ(n)
    using card˙non˙empty˙succ by auto
  from A1 have σ(Λ) : —Λ— → Λ
    using enums˙valid˙in˙semigr1 enums.enum˙props
    by simp
  with A1 have a O σ(Λ): —Λ— → G
    using a˙is˙fun FinPow˙def comp˙fun˙subset
    by simp
  with ‹n ∈ nat› and ‹—Λ— = succ(n)› show
    ∃n ∈ nat . —Λ— = succ(n) ∧ a O σ(Λ) : succ(n) → G
    by auto
  from  ‹n ∈ nat› ‹—Λ— = succ(n)› ‹a O σ(Λ): —Λ— → G›
  show ∏(Λ,a) ∈ G using prod˙type setproddef
    by auto
```

qed

The enum˙append lemma from the Enemeration theory specialized for natural numbers.

```
lemma (in semigr1) semigr1˙enum˙append:
  assumes Λ ∈ FinPow(X) and
  n ∈ X - Λ and ∀k∈Λ. ⟨k,n⟩ ∈ r
  shows σ(Λ ∪ –n″) = σ(Λ)↩ n
  using assms  FinPow˙def enums˙valid˙in˙semigr1
    enums.enum˙append by simp
```

What is product over a singleton?

```
lemma (in semigr1) gen˙prod˙singleton:
  assumes A1: x ∈ X
  shows  ∏(–x″,a) = a(x)
proof -
  from A1 have σ(–x″): 1 → X and  σ(–x″)(0) = x
    using enums˙valid˙in˙semigr1 enums.enum˙singleton
    by auto
  then show ∏(–x″,a) = a(x)
    using a˙is˙fun comp˙fun setproddef prod˙of˙1elem
      comp˙fun˙apply by simp
qed
```

A generalization of prod˙append to the products over sets of indices.

```
lemma (in semigr1) gen˙prod˙append:
  assumes
  A1: Λ ∈ FinPow(X) and A2: Λ ≠ 0 and
  A3: n ∈ X -  Λ and
  A4: ∀k∈Λ. ⟨k,n⟩ ∈ r
  shows ∏(Λ ∪ –n″, a) = (∏(Λ,a)) · a(n)
proof -
  have ∏(Λ ∪ –n″, a) =  ∏  (a O σ(Λ ∪ –n″))
    using setproddef by simp
  also from A1 A3 A4 have . . . = ∏  (a O (σ(Λ)↩ n))
    using semigr1˙enum˙append by simp
  also have . . . = ∏  ((a O σ(Λ))↩ a(n))
  proof -
    from A1 A3 have
      —Λ— ∈ nat and σ(Λ) : —Λ— → X and n ∈ X
      using card˙fin˙is˙nat enums˙valid˙in˙semigr1 enums.enum˙fun
      by auto
    then show thesis using a˙is˙fun list˙compose˙append
      by simp
  qed
  also from assms have . . . = (∏  (a O σ(Λ)))·a(n)
    using a˙is˙fun setprod˙type apply˙funtype prod˙append
    by blast
  also have . . . = (∏(Λ,a)) · a(n)
```

```
    using SetFold˙def by simp
  finally show ∏(Λ ∪ –n˝, a) = (∏(Λ,a)) · a(n)
    by simp
qed
```

Very similar to gen˙prod˙append: a relation between a product over a set of indices and the product over the set with the maximum removed.

```
lemma (in semigr1) gen˙product˙rem˙point:
  assumes A1: A ∈ FinPow(X) and
  A2: n ∈ A and  A4: A - –n˝ ≠ 0 and
  A3: ∀k∈A. ⟨k, n⟩ ∈ r
  shows
  (∏(A - –n˝,a)) · a(n) = ∏(A, a)
proof -
  let Λ = A - –n˝
  from A1 A2 have Λ ∈ FinPow(X) and n ∈ X -  Λ
    using fin˙rem˙point˙fin FinPow˙def by auto
  with A3 A4 have ∏(Λ ∪ –n˝, a) = (∏(Λ,a)) · a(n)
    using a˙is˙fun gen˙prod˙append by blast
  with A2 show thesis using rem˙add˙eq by simp
qed
```

23.3 Commutative semigroups

Commutative semigroups are those whose operation is commutative, i.e. $a \cdot b = b \cdot a$. This implies that for any permutation $s : n \to n$ we have $\prod_{j=0}^{n} a_j = \prod_{j=0}^{n} a_{s(j)}$, or, closer to the notation we are using in the semigr0 context, $\prod a = \prod(a \circ s)$. Maybe one day we will be able to prove this, but for now the goal is to prove something simpler: that if the semigroup operation is commutative taking the product of a sequence is distributive with respect to the operation: $\prod_{j=0}^{n}(a_j \cdot b_j) = \left(\prod_{j=0}^{n} a_j)\right) \left(\prod_{j=0}^{n} b_j)\right)$. Many of the rearrangements (namely those that don't use the inverse) proven in the AbelianGroup˙ZF theory hold in fact in semigroups. Some of them will be reproven in this section.

A rearrangement with 3 elements.

```
lemma (in semigr0) rearr3elems:
  assumes f –is commutative on˝ G and a∈G  b∈G  c∈G
  shows a·b·c = a·c·b
  using assms semigr˙assoc IsCommutative˙def by simp
```

A rearrangement of four elements.

```
lemma (in semigr0) rearr4elems:
  assumes A1: f –is commutative on˝ G and
  A2: a∈G  b∈G  c∈G  d∈G
  shows a·b·(c·d) = a·c·(b·d)
proof -
```

```
  from A2 have a·b·(c·d) = a·b·c·d
    using semigr˙closed semigr˙assoc by simp
  also have a·b·c·d =  a·c·(b·d)
  proof -
    from A1 A2 have a·b·c·d = c·(a·b)·d
      using IsCommutative˙def semigr˙closed
      by simp
    also from A2 have ... =  c·a·b·d
      using semigr˙closed semigr˙assoc
      by simp
    also from A1 A2 have ... = a·c·b·d
      using IsCommutative˙def semigr˙closed
      by simp
    also from A2 have ... = a·c·(b·d)
      using semigr˙closed semigr˙assoc
      by simp
    finally show  a·b·c·d =  a·c·(b·d) by simp
  qed
  finally show  a·b·(c·d) = a·c·(b·d)
    by simp
qed
```

We start with a version of prod˙append that will shorten a bit the proof of the main theorem.

```
lemma (in semigr0) shorter˙seq: assumes A1: k ∈ nat and
  A2: a ∈ succ(succ(k)) → G
  shows (∏ a) = (∏ Init(a)) · a(succ(k))
proof -
  let x = Init(a)
  from assms have
    a(succ(k)) ∈ G and x : succ(k) → G
    using apply˙funtype init˙props by auto
  with A1 have (∏ x↩a(succ(k))) = (∏ x) · a(succ(k))
    using prod˙append by simp
  with assms show thesis using init˙props
    by simp
qed
```

A lemma useful in the induction step of the main theorem.

```
lemma (in semigr0) prod˙distr˙ind˙step:
  assumes A1: k ∈ nat and
  A2: a : succ(succ(k)) → G and
  A3: b : succ(succ(k)) → G and
  A4: c : succ(succ(k)) → G and
  A5: ∀j∈succ(succ(k)). c(j) = a(j) · b(j)
  shows
  Init(a) : succ(k) → G
  Init(b) : succ(k) → G
  Init(c) : succ(k) → G
```

```
  ∀j∈succ(k). Init(c)(j) = Init(a)(j) · Init(b)(j)
proof -
  from A1 A2 A3 A4 show
    Init(a) : succ(k) → G
    Init(b) : succ(k) → G
    Init(c) : succ(k) → G
    using init˙props by auto
  from A1 have T: succ(k) ∈ nat by simp
  from T A2 have ∀j∈succ(k). Init(a)(j) = a(j)
    by (rule init˙props)
  moreover from T A3 have ∀j∈succ(k). Init(b)(j) = b(j)
     by (rule init˙props)
   moreover from T A4 have ∀j∈succ(k). Init(c)(j) = c(j)
     by (rule init˙props)
   moreover from A5 have ∀j∈succ(k). c(j) = a(j) · b(j)
     by simp
   ultimately show ∀j∈succ(k). Init(c)(j) = Init(a)(j) · Init(b)(j)
     by simp
qed
```

For commutative operations taking the product of a sequence is distributive with respect to the operation. This version will probably not be used in applications, it is formulated in a way that is easier to prove by induction. For a more convenient formulation see prod˙comm˙distrib. The proof by induction on the length of the sequence.

```
theorem (in semigr0) prod˙comm˙distr:
  assumes A1: f –is commutative on″ G and A2: n∈nat
  shows ∀ a b c.
  (a : succ(n)→G ∧ b : succ(n)→G ∧ c : succ(n)→G ∧
  (∀j∈succ(n). c(j) = a(j) · b(j))) ⟶
  (∏ c) = (∏ a) · (∏ b)
proof -
  note A2
  moreover have ∀ a b c.
    (a : succ(0)→G ∧ b : succ(0)→G ∧ c : succ(0)→G ∧
    (∀j∈succ(0). c(j) = a(j) · b(j))) ⟶
    (∏ c) = (∏ a) · (∏ b)
  proof -
    { fix a b c
      assume a : succ(0)→G ∧ b : succ(0)→G ∧ c : succ(0)→G ∧
 (∀j∈succ(0). c(j) = a(j) · b(j))
      then have
 I: a : 1→G  b : 1→G  c : 1→G and
 II: c(0) = a(0) · b(0) by auto
      from I have
 (∏ a) = a(0) and (∏ b) = b(0) and (∏ c) = c(0)
 using prod˙of˙1elem by auto
      with II have (∏ c) = (∏ a) · (∏ b) by simp
    } then show thesis using Fold1˙def by simp
```

```
qed
moreover have ∀k ∈ nat.
  (∀ a b c.
  (a : succ(k)→G ∧ b : succ(k)→G ∧ c : succ(k)→G ∧
  (∀j∈succ(k). c(j) = a(j) · b(j))) ⟶
  (∏ c) = (∏ a) · (∏ b)) ⟶
  (∀ a b c.
  (a : succ(succ(k))→G ∧ b : succ(succ(k))→G ∧ c : succ(succ(k))→G ∧
  (∀j∈succ(succ(k)). c(j) = a(j) · b(j))) ⟶
  (∏ c) = (∏ a) · (∏ b))
proof
  fix k assume k ∈ nat
  show (∀a b c.
    a ∈ succ(k) → G ∧
    b ∈ succ(k) → G ∧ c ∈ succ(k) → G ∧
    (∀j∈succ(k). c(j) = a(j) · b(j)) ⟶
    (∏ c) = (∏ a) · (∏ b)) ⟶
    (∀a b c.
    a ∈ succ(succ(k)) → G ∧
    b ∈ succ(succ(k)) → G ∧
    c ∈ succ(succ(k)) → G ∧
    (∀j∈succ(succ(k)). c(j) = a(j) · b(j)) ⟶
    (∏ c) = (∏ a) · (∏ b))
  proof
    assume A3: ∀a b c.
a ∈ succ(k) → G ∧
b ∈ succ(k) → G ∧ c ∈ succ(k) → G ∧
(∀j∈succ(k). c(j) = a(j) · b(j)) ⟶
(∏ c) = (∏ a) · (∏ b)
    show ∀a b c.
a ∈ succ(succ(k)) → G ∧
b ∈ succ(succ(k)) → G ∧
c ∈ succ(succ(k)) → G ∧
(∀j∈succ(succ(k)). c(j) = a(j) · b(j)) ⟶
(∏ c) = (∏ a) · (∏ b)
    proof -
{ fix a b c
  assume
    a ∈ succ(succ(k)) → G ∧
    b ∈ succ(succ(k)) → G ∧
    c ∈ succ(succ(k)) → G ∧
    (∀j∈succ(succ(k)). c(j) = a(j) · b(j))
  with ‹k ∈ nat› have I:
    a : succ(succ(k)) → G
    b : succ(succ(k)) → G
    c : succ(succ(k)) → G
    and II: ∀j∈succ(succ(k)). c(j) = a(j) · b(j)
    by auto
  let x = Init(a)
```

```
        let y = Init(b)
        let z = Init(c)
  from ‹k ∈ nat› I have III:
    (∏ a) = (∏ x) · a(succ(k))
    (∏ b) = (∏ y) · b(succ(k)) and
    IV: (∏ c) = (∏ z) · c(succ(k))
    using  shorter˙seq by auto
  moreover
  from  ‹k ∈ nat› I II have
    x : succ(k) → G
    y : succ(k) → G
    z : succ(k) → G and
    ∀j∈succ(k). z(j) = x(j) · y(j)
    using prod˙distr˙ind˙step by auto
  with A3 II IV have
    (∏ c) = (∏ x)·(∏ y)·(a(succ(k)) · b(succ(k)))
    by simp
  moreover from A1 ‹k ∈ nat› I III have
    (∏ x)·(∏ y)·(a(succ(k)) · b(succ(k)))=
    (∏ a) · (∏ b)
    using init˙props prod˙type apply˙funtype
      rearr4elems by simp
  ultimately have (∏ c) = (∏ a) · (∏ b)
    by simp
 } thus thesis by auto
     qed
   qed
 qed
 ultimately show thesis by (rule ind˙on˙nat)
qed
```

A reformulation of prod˙comm˙distr that is more convenient in applications.

```
theorem (in semigr0) prod˙comm˙distrib:
  assumes  f –is commutative on″ G and n∈nat and
  a : succ(n)→G  b : succ(n)→G  c : succ(n)→G and
  ∀j∈succ(n). c(j) = a(j) · b(j)
  shows (∏ c) = (∏ a) · (∏ b)
  using assms prod˙comm˙distr by simp
```

A product of two products over disjoint sets of indices is the product over the union.

```
lemma (in semigr1) prod˙bisect:
  assumes  A1: f –is commutative on″ G  and A2: Λ ∈ FinPow(X)
  shows
  ∀P ∈  Bisections(Λ). ∏(Λ,a) = (∏(fst(P),a))·(∏(snd(P),a))
proof -
  have IsLinOrder(X,r) using linord by simp
  moreover have
    ∀P ∈  Bisections(0). ∏(0,a) = (∏(fst(P),a))·(∏(snd(P),a))
```

using bisec˙empty by simp
moreover have ∀ A ∈ FinPow(X).
(∀ n ∈ X - A.
(∀P ∈ Bisections(A). ∏(A,a) = (∏(fst(P),a))·(∏(snd(P),a)))
∧ (∀k∈A. ⟨k,n⟩ ∈ r) ⟶
(∀Q ∈ Bisections(A ∪ −n″).
∏(A ∪ −n″,a) = (∏(fst(Q),a))·(∏(snd(Q),a))))
proof -
{ fix A assume A ∈ FinPow(X)
fix n assume n ∈ X - A
have (∀P ∈ Bisections(A).
∏(A,a) = (∏(fst(P),a))·(∏(snd(P),a)))
∧ (∀k∈A. ⟨k,n⟩ ∈ r) ⟶
(∀Q ∈ Bisections(A ∪ −n″).
∏(A ∪ −n″,a) = (∏(fst(Q),a))·(∏(snd(Q),a)))
proof -
{ assume I:
∀P ∈ Bisections(A). ∏(A,a) = (∏(fst(P),a))·(∏(snd(P),a))
and II: ∀k∈A. ⟨k,n⟩ ∈ r
have ∀Q ∈ Bisections(A ∪ −n″).
∏(A ∪ −n″,a) = (∏(fst(Q),a))·(∏(snd(Q),a))
proof -
{ fix Q assume Q ∈ Bisections(A ∪ −n″)
let Q_0 = fst(Q)
let Q_1 = snd(Q)
from ‹A ∈ FinPow(X)› ‹n ∈ X - A› have A ∪ −n″ ∈ FinPow(X)
using singleton˙in˙finpow union˙finpow by auto
with ‹Q ∈ Bisections(A ∪ −n″)› have
Q_0 ∈ FinPow(X) Q_0 ≠ 0 and Q_1 ∈ FinPow(X) Q_1 ≠ 0
using bisect˙fin bisec˙is˙pair Bisections˙def by auto
then have ∏(Q_0,a) ∈ G and ∏(Q_1,a) ∈ G
using a˙is˙fun setprod˙type by auto
from ‹Q ∈ Bisections(A ∪ −n″)› ‹A ∈ FinPow(X)› ‹n ∈ X-A›
have refl(X,r) Q_0 ⊆ A ∪ −n″ Q_1 ⊆ A ∪ −n″
A ⊆ X and n ∈ X
using linord IsLinOrder˙def total˙is˙refl Bisections˙def
FinPow˙def by auto
from ‹refl(X,r)› ‹Q_0 ⊆ A ∪ −n″› ‹A ⊆ X› ‹n ∈ X› II
have III: ∀k ∈ Q_0. ⟨k, n⟩ ∈ r by (rule refl˙add˙point)
from ‹refl(X,r)› ‹Q_1 ⊆ A ∪ −n″› ‹A ⊆ X› ‹n ∈ X› II
have IV: ∀k ∈ Q_1. ⟨k, n⟩ ∈ r by (rule refl˙add˙point)
from ‹n ∈ X - A› ‹Q ∈ Bisections(A ∪ −n″)› have
Q_0 = −n″ ∨ Q_1 = −n″ ∨ ⟨Q_0 - −n″,Q_1-−n″⟩ ∈ Bisections(A)
using bisec˙is˙pair bisec˙add˙point by simp
moreover
{ assume Q_1 = −n″
from ‹n ∈ X - A› have n ∉ A by auto
moreover
from ‹Q ∈ Bisections(A ∪ −n″)›

have $\langle Q_0,Q_1 \rangle \in$ Bisections(A $\cup$ –n″)
using bisec˙is˙pair by simp
with ‹Q_1 = –n″› have $\langle Q_0$, –n″$\rangle \in$ Bisections(A $\cup$ –n″)
by simp
ultimately have Q_0 = A and A $\neq$ 0
using set˙point˙bisec by auto
with ‹A $\in$ FinPow(X)› ‹n $\in$ X - A› II ‹Q_1 = –n″›
have $\prod$(A $\cup$ –n″,a) = ($\prod$(Q_0,a))·$\prod$(Q_1,a)
using a˙is˙fun gen˙prod˙append gen˙prod˙singleton
by simp }
moreover
{ assume Q_0 = –n″
from ‹n $\in$ X - A› have n $\in$ X by auto
then have –n″ $\in$ FinPow(X) and –n″ $\neq$ 0
using singleton˙in˙finpow by auto
from ‹n $\in$ X - A› have n $\notin$ A by auto
moreover
from ‹Q $\in$ Bisections(A $\cup$ –n″)›
have $\langle Q_0$, $Q_1\rangle \in$ Bisections(A $\cup$ –n″)
using bisec˙is˙pair by simp
with ‹Q_0 = –n″› have $\langle$–n″, $Q_1\rangle \in$ Bisections(A $\cup$ –n″)
by simp
ultimately have Q_1 = A and A $\neq$ 0 using point˙set˙bisec
by auto
with A1 ‹A $\in$ FinPow(X)› ‹n $\in$ X - A› II
‹–n″ $\in$ FinPow(X)› ‹–n″ $\neq$ 0› ‹Q_0 = –n″›
have $\prod$(A $\cup$ –n″,a) = ($\prod$(Q_0,a))·($\prod$(Q_1,a))
using a˙is˙fun gen˙prod˙append gen˙prod˙singleton
setprod˙type IsCommutative˙def by auto }
moreover
{ assume A4: $\langle Q_0$ - –n″,Q_1 - –n″$\rangle \in$ Bisections(A)
with ‹A $\in$ FinPow(X)› have
Q_0 - –n″ $\in$ FinPow(X) Q_0 - –n″ $\neq$ 0 and
Q_1 - –n″ $\in$ FinPow(X) Q_1 - –n″ $\neq$ 0
using FinPow˙def Bisections˙def by auto
with ‹n $\in$ X - A› have
$\prod$(Q_0 - –n″,a) $\in$ G $\prod$(Q_1 - –n″,a) $\in$ G and
T: a(n) $\in$ G
using a˙is˙fun setprod˙type apply˙funtype by auto
from ‹Q $\in$ Bisections(A $\cup$ –n″)› A4 have
($\langle Q_0$, Q_1 - –n″$\rangle \in$ Bisections(A) $\wedge$ n $\in$ Q_1) $\vee$
($\langle Q_0$ - –n″, $Q_1\rangle \in$ Bisections(A) $\wedge$ n $\in$ Q_0)
using bisec˙is˙pair bisec˙add˙point˙case3 by auto
moreover
{ assume $\langle Q_0$, Q_1 - –n″$\rangle \in$ Bisections(A) and n $\in$ Q_1
then have A $\neq$ 0 using bisec˙props by simp
with A2 ‹A $\in$ FinPow(X)› ‹n $\in$ X - A› I II T IV
‹$\langle Q_0$, Q_1 - –n″$\rangle \in$ Bisections(A)› ‹$\prod$(Q_0,a) $\in$ G›
‹$\prod$(Q_1 - –n″,a) $\in$ G› ‹Q_1 $\in$ FinPow(X)›

```
      ‹n ∈ Q1› ‹Q1 - –n″ ≠ 0›
    have ∏(A ∪ –n″,a) = (∏(Q0,a))·(∏(Q1,a))
      using gen˙prod˙append semigr˙assoc gen˙product˙rem˙point
      by simp }
  moreover
  { assume ⟨Q0 - –n″, Q1⟩ ∈ Bisections(A) and n ∈ Q0
    then have A ≠ 0 using bisec˙props by simp
    with A1 A2 ‹A ∈ FinPow(X)› ‹n ∈ X - A› I II III T
      ‹⟨Q0 - –n″, Q1⟩∈Bisections(A)› ‹∏(Q0 - –n″,a)∈G›
      ‹∏(Q1,a) ∈ G› ‹Q0 ∈ FinPow(X)› ‹n ∈ Q0› ‹Q0-–n″≠0›
    have ∏(A ∪ –n″,a) = (∏(Q0,a))·(∏(Q1,a))
      using gen˙prod˙append rearr3elems gen˙product˙rem˙point
        by simp }
  ultimately have
    ∏(A ∪ –n″,a) = (∏(Q0,a))·(∏(Q1,a))
    by auto }
       ultimately have ∏(A ∪ –n″,a) = (∏(Q0,a))·(∏(Q1,a))
  by auto
     } thus thesis by simp
   qed
 } thus thesis by simp
      qed
    } thus thesis by simp
  qed
  moreover note A2
  ultimately show thesis by (rule fin˙ind˙add˙max)
qed
```

A better looking reformulation of prod˙bisect.

```
theorem (in semigr1) prod˙disjoint: assumes
  A1: f –is commutative on″ G  and
  A2: A ∈ FinPow(X) A ≠ 0 and
  A3: B ∈ FinPow(X) B ≠ 0 and
  A4: A ∩ B = 0
  shows ∏(A∪B,a) = (∏(A,a))·(∏(B,a))
proof -
  from A2 A3 A4 have ⟨A,B⟩ ∈ Bisections(A∪B)
    using is˙bisec by simp
  with A1 A2 A3 show thesis
    using a˙is˙fun union˙finpow prod˙bisect by simp
qed
```

A generalization of prod˙disjoint.

```
lemma (in semigr1) prod˙list˙of˙lists: assumes
  A1: f –is commutative on″ G  and A2: n ∈ nat
  shows ∀M ∈ succ(n) → FinPow(X).
  M –is partition″ ⟶
  (∏ –⟨i,∏(M(i),a)⟩. i ∈ succ(n)″) =
  (∏(⋃i ∈ succ(n). M(i),a))
```

```
proof -
  note A2
  moreover have ∀ M ∈ succ(0) → FinPow(X).
    M –is partition″ ⟶
    (∏ –⟨i,∏(M(i),a)⟩. i ∈ succ(0)″) = (∏(⋃i ∈ succ(0). M(i),a))
    using a˙is˙fun func1˙1˙L1 Partition˙def apply˙funtype setprod˙type
      list˙len1˙singleton prod˙of˙1elem
    by simp
  moreover have ∀ k ∈ nat.
    (∀ M ∈ succ(k) → FinPow(X).
    M –is partition″ ⟶
    (∏ –⟨i,∏(M(i),a)⟩. i ∈ succ(k)″) =
    (∏(⋃i ∈ succ(k). M(i),a))) ⟶
    (∀ M ∈ succ(succ(k)) → FinPow(X).
    M –is partition″ ⟶
    (∏ –⟨i,∏(M(i),a)⟩. i ∈ succ(succ(k))″) =
    (∏(⋃i ∈ succ(succ(k)). M(i),a)))
  proof -
    { fix k assume k ∈ nat
      assume A3: ∀ M ∈ succ(k) → FinPow(X).
 M –is partition″ ⟶
 (∏ –⟨i,∏(M(i),a)⟩. i ∈ succ(k)″) =
 (∏(⋃i ∈ succ(k). M(i),a))
      have (∀ N ∈ succ(succ(k)) → FinPow(X).
 N –is partition″ ⟶
 (∏ –⟨i,∏(N(i),a)⟩. i ∈ succ(succ(k))″) =
 (∏(⋃i ∈ succ(succ(k)). N(i),a)))
      proof -
 { fix N assume A4: N : succ(succ(k)) → FinPow(X)
   assume A5: N –is partition″
   with A4 have I: ∀ i ∈ succ(succ(k)). N(i) ≠ 0
     using func1˙1˙L1 Partition˙def by simp
   let b = –⟨i,∏(N(i),a)⟩. i ∈ succ(succ(k))″
   let c = –⟨i,∏(N(i),a)⟩. i ∈ succ(k)″
   have II: ∀ i ∈ succ(succ(k)). ∏(N(i),a) ∈ G
   proof
     fix i assume i ∈ succ(succ(k))
     with A4 I have N(i) ∈ FinPow(X) and N(i) ≠ 0
       using apply˙funtype by auto
     then show ∏(N(i),a) ∈ G using setprod˙type
       by simp
   qed
   hence ∀ i ∈ succ(k).  ∏(N(i),a) ∈ G by auto
   then have c : succ(k) → G by (rule ZF˙fun˙from˙total)
   have b = –⟨i,∏(N(i),a)⟩. i ∈ succ(succ(k))″
     by simp
   with II have b = Append(c,∏(N(succ(k)),a))
     by (rule set˙list˙append)
   with  II  ‹k ∈ nat› ‹c : succ(k) → G›
```

```
  have (∏ b) = (∏ c)·(∏(N(succ(k)),a))
    using prod˙append by simp
  also have
    ... =  (∏(⋃i ∈ succ(k). N(i),a))·(∏(N(succ(k)),a))
  proof -
    let M = restrict(N,succ(k))
    have succ(k) ⊆ succ(succ(k)) by auto
    with ‹N : succ(succ(k)) → FinPow(X)›
    have M : succ(k) → FinPow(X) and
      III: ∀i ∈ succ(k). M(i) = N(i)
      using restrict˙type2 restrict apply˙funtype
      by auto
    with A5 ‹M : succ(k) → FinPow(X)›have M –is partition″
      using func1˙1˙L1 Partition˙def by simp
    with A3 ‹M : succ(k) → FinPow(X)› have
      (∏ –⟨i,∏(M(i),a)⟩. i ∈ succ(k)″) =
      (∏(⋃i ∈ succ(k). M(i),a))
      by blast
    with III show thesis by simp
  qed
  also have ... = (∏(⋃i ∈ succ(succ(k)). N(i),a))
  proof -
    let A = ⋃i ∈ succ(k). N(i)
    let B = N(succ(k))
    from A4 ‹k ∈ nat› have succ(k) ∈ nat and
      ∀i ∈ succ(k). N(i) ∈ FinPow(X)
      using apply˙funtype by auto
    then have A ∈ FinPow(X) by (rule union˙fin˙list˙fin)
    moreover from I have A ≠ 0 by auto
    moreover from A4 I have
      N(succ(k)) ∈ FinPow(X) and N(succ(k)) ≠ 0
      using apply˙funtype by auto
    moreover from ‹succ(k) ∈ nat› A4 A5 have A ∩ B = 0
      by (rule list˙partition)
    moreover note A1
    ultimately have ∏(A∪B,a) = (∏(A,a))·(∏(B,a))
      using prod˙disjoint by simp
    moreover have A ∪ B = (⋃i ∈ succ(succ(k)). N(i))
      by auto
    ultimately show thesis by simp
  qed
  finally have (∏ –⟨i,∏(N(i),a)⟩. i ∈ succ(succ(k))″) =
    (∏(⋃i ∈ succ(succ(k)). N(i),a))
    by simp
  } thus thesis by auto
qed
} thus thesis by simp
 qed
 ultimately show thesis by (rule ind˙on˙nat)
```

qed

A more convenient reformulation of prod˙list˙of˙lists.

```
theorem (in semigr1) prod˙list˙of˙sets:
  assumes A1: f –is commutative on″ G  and
  A2: n ∈ nat  n ≠ 0 and
  A3: M : n → FinPow(X)   M –is partition″
  shows
  (∏ –⟨i,∏(M(i),a)⟩. i ∈ n″) = (∏(⋃i ∈ n. M(i),a))
proof -
  from A2 obtain k where k ∈ nat and n = succ(k)
    using Nat˙ZF˙1˙L3 by auto
  with A1 A3 show thesis using prod˙list˙of˙lists
    by simp
qed
```

The definition of the product ∏(A,a) ≡ SetFold(f,a,A,r) of a some (finite) set of semigroup elements requires that r is a linear order on the set of indices A. This is necessary so that we know in which order we are multiplying the elements. The product over A is defined so that we have $\prod_A a = \prod a \circ \sigma(A)$ where $\sigma : |A| \to A$ is the enumeration of A (the only order isomorphism between the number of elements in A and A), see lemma setproddef. However, if the operation is commutative, the order is irrelevant. The next theorem formalizes that fact stating that we can replace the enumeration $\sigma(A)$ by any bijection between $|A|$ and A. In a way this is a generalization of setproddef. The proof is based on application of prod˙list˙of˙sets to the finite collection of singletons that comprise A.

```
theorem (in semigr1) prod˙order˙irr:
  assumes A1: f –is commutative on″ G and
  A2: A ∈ FinPow(X) A ≠ 0 and
  A3: b ∈ bij(—A—,A)
  shows (∏ (a O b)) = ∏(A,a)
proof -
  let n = —A—
  let M = –⟨k, –b(k)″⟩. k ∈ n″
  have (∏ (a O b)) = (∏ –⟨i,∏(M(i),a)⟩. i ∈ n″)
  proof -
    have ∀i ∈ n. ∏(M(i),a) = (a O b)(i)
    proof
      fix i assume i ∈ n
      with A2 A3 ‹i ∈ n› have b(i) ∈ X
 using bij˙def inj˙def apply˙funtype FinPow˙def
 by auto
      then have ∏(–b(i)″,a) = a(b(i))
 using gen˙prod˙singleton by simp
      with A3 ‹i ∈ n› have ∏(–b(i)″,a) = (a O b)(i)
 using bij˙def inj˙def comp˙fun˙apply by auto
      with ‹i ∈ n› A3 show ∏(M(i),a) = (a O b)(i)
```

```
using bij˙def inj˙partition by auto
  qed
  hence –⟨i,∏(M(i),a)⟩. i ∈ n″ = –⟨i,(a O b)(i)⟩. i ∈ n″
    by simp
  moreover have –⟨i,(a O b)(i)⟩. i ∈ n″ = a O b
  proof -
    from A3 have b : n → A using bij˙def inj˙def by simp
    moreover from A2 have A ⊆ X using FinPow˙def by simp
    ultimately have b : n → X by (rule func1˙1˙L1B)
    then have a O b: n → G using a˙is˙fun comp˙fun
by simp
    then show –⟨i,(a O b)(i)⟩. i ∈ n″ = a O b
using fun˙is˙set˙of˙pairs by simp
  qed
  ultimately show thesis by simp
 qed
 also have ... = (∏(⋃i ∈ n. M(i),a))
 proof -
  note A1
  moreover from A2 have n ∈ nat and n ≠ 0
    using card˙fin˙is˙nat card˙non˙empty˙non˙zero by auto
  moreover have M : n → FinPow(X) and M –is partition″
  proof -
    from A2 A3 have ∀k ∈ n. –b(k)″ ∈  FinPow(X)
using bij˙def inj˙def apply˙funtype FinPow˙def
 singleton˙in˙finpow by auto
    then show M : n → FinPow(X) using ZF˙fun˙from˙total
by simp
    from A3 show M –is partition″ using bij˙def inj˙partition
by auto
  qed
  ultimately show
    (∏ –⟨i,∏(M(i),a)⟩. i ∈ n″) = (∏(⋃i ∈ n. M(i),a))
    by (rule prod˙list˙of˙sets)
 qed
 also from A3 have (∏(⋃i ∈ n. M(i),a)) = ∏(A,a)
  using bij˙def inj˙partition surj˙singleton˙image
  by auto
 finally show thesis by simp
qed
```

Another way of expressing the fact that the product dos not depend on the order.

```
corollary (in semigr1) prod˙bij˙same:
 assumes f –is commutative on″ G and
 A ∈ FinPow(X) A ≠ 0 and
 b ∈ bij(—A—,A) c ∈ bij(—A—,A)
 shows (∏ (a O b)) = (∏ (a O c))
 using assms prod˙order˙irr by simp
```

end

24 Commutative Semigroups

theory CommutativeSemigroup˙ZF imports Semigroup˙ZF

begin

In the Semigroup theory we introduced a notion of SetFold(f,a,Λ,r) that represents the sum of values of some function a valued in a semigroup where the arguments of that function vary over some set Λ. Using the additive notation something like this would be expressed as $\sum_{x\in\Lambda} f(x)$ in informal mathematics. This theory considers an alternative to that notion that is more specific to commutative semigroups.

24.1 Sum of a function over a set

The r parameter in the definition of SetFold(f,a,Λ,r) (from Semigroup˙ZF) represents a linear order relation on Λ that is needed to indicate in what order we are summing the values $f(x)$. If the semigroup operation is commutative the order does not matter and the relation r is not needed. In this section we define a notion of summing up values of some function $a : X \to G$ over a finite set of indices $\Gamma \subseteq X$, without using any order relation on X.

We define the sum of values of a function $a : X \to G$ over a set Λ as the only element of the set of sums of lists that are bijections between the number of values in Λ (which is a natural number $n = \{0, 1, .., n-1\}$ if Λ is finite) and Λ. The notion of Fold1(f,c) is defined in Semigroup˙ZF as the fold (sum) of the list c starting from the first element of that list. The intention is to use the fact that since the result of summing up a list does not depend on the order, the set –Fold1(f,a O b). b ∈ bij(—Λ—, Λ)″ is a singleton and we can extract its only value by taking its union.

definition
CommSetFold(f,a,Λ) = ⋃–Fold1(f,a O b). b ∈ bij(—Λ—, Λ)″

the next locale sets up notation for writing about summation in commutative semigroups. We define two kinds of sums. One is the sum of elements of a list (which are just functions defined on a natural number) and the second one represents a more general notion the sum of values of a semigroup valued function over some set of arguments. Since those two types of sums are different notions they are represented by different symbols. However in the presentations they are both intended to be printed as $\sum$.

locale commsemigr =

```
fixes G f

assumes csgassoc: f –is associative on″ G

assumes csgcomm: f –is commutative on″ G

fixes csgsum (infixl + 69)
defines csgsum˙def[simp]: x + y ≡ f⟨x,y⟩

fixes X a
assumes csgaisfun: a : X → G

fixes csglistsum (∑ ˙ 70)
defines csglistsum˙def[simp]: ∑k ≡ Fold1(f,k)

fixes csgsetsum (∑)
defines csgsetsum˙def[simp]: ∑(A,h) ≡ CommSetFold(f,h,A)
```

Definition of a sum of function over a set in notation defined in the commsemigr locale.

```
lemma (in commsemigr) CommSetFolddef:
  shows (∑(A,a)) = (⋃–∑ (a O b). b ∈ bij(—A—, A)″)
  using CommSetFold˙def by simp
```

The next lemma states that the result of a sum does not depend on the order we calculate it. This is similar to lemma prod˙order˙irr in the Semigroup theory, except that the semigr1 locale assumes that the domain of the function we sum up is linearly ordered, while in commsemigr we don't have this assumption.

```
lemma (in commsemigr) sum˙over˙set˙bij:
  assumes A1: A ∈ FinPow(X) A ≠ 0 and A2: b ∈ bij(—A—,A)
  shows (∑(A,a)) = (∑ (a O b))
proof -
  have
    ∀c ∈ bij(—A—,A). ∀ d ∈ bij(—A—,A). (∑ (a O c)) = (∑ (a O d))
  proof -
    { fix c assume c ∈ bij(—A—,A)
      fix d assume d ∈ bij(—A—,A)
      let r = InducedRelation(converse(c), Le)
      have semigr1(G,f,A,r,restrict(a, A))
      proof -
 have semigr0(G,f) using csgassoc semigr0˙def by simp
 moreover from A1 ‹c ∈ bij(—A—,A)› have IsLinOrder(A,r)
   using bij˙converse˙bij card˙fin˙is˙nat
           natord˙lin˙on˙each˙nat ind˙rel˙pres˙lin by simp
 moreover from A1 have restrict(a, A) : A → G
   using FinPow˙def csgaisfun restrict˙fun by simp
 ultimately show thesis using semigr1˙axioms.intro semigr1˙def
```

```
  by simp
    qed
    moreover have f –is commutative on″ G using csgcomm
 by simp
    moreover from A1 have A ∈  FinPow(A) A ≠ 0
 using FinPow˙def by auto
    moreover note ‹c ∈ bij(—A—,A)› ‹d ∈ bij(—A—,A)›
    ultimately have
 Fold1(f,restrict(a,A) O c) = Fold1(f,restrict(a,A) O d)
 by (rule semigr1.prod˙bij˙same)
    hence (∑ (restrict(a,A) O c)) = (∑ (restrict(a,A) O d))
 by simp
    moreover from A1 ‹c ∈ bij(—A—,A)› ‹d ∈ bij(—A—,A)›
    have
 restrict(a,A) O c = a O c and restrict(a,A) O d = a O d
 using bij˙def surj˙def csgaisfun FinPow˙def comp˙restrict
 by auto
    ultimately have (∑(a O c)) = (∑(a O d)) by simp
    } thus thesis by blast
   qed
  with A2 have (⋃–∑(a O b). b ∈ bij(—A—, A)″) = (∑ (a O b))
   by (rule singleton˙comprehension)
  then show thesis using CommSetFolddef by simp
qed
```

The result of a sum is in the semigroup. Also, as the second assertion we show that every semigroup valued function generates a homomorphism between the finite subsets of a semigroup and the semigroup. Adding an element to a set coresponds to adding a value.

```
lemma (in commsemigr) sum˙over˙set˙add˙point:
  assumes  A1: A ∈ FinPow(X)  A ≠ 0
  shows ∑(A,a) ∈ G and
  ∀x ∈ X-A. ∑(A ∪ –x″,a) = (∑(A,a)) + a(x)
proof -
  from A1 obtain b where b ∈ bij(—A—,A)
    using fin˙bij˙card by auto
  with A1 have ∑(A,a) = (∑ (a O b))
    using sum˙over˙set˙bij by simp
  from A1 have —A— ∈ nat using card˙fin˙is˙nat by simp
  have semigr0(G,f) using csgassoc semigr0˙def by simp
  moreover
  from A1 obtain n where n ∈ nat and —A— = succ(n)
    using card˙non˙empty˙succ by auto
  with A1  ‹b ∈ bij(—A—,A)› have
    n ∈ nat and a O b : succ(n) → G
    using bij˙def inj˙def FinPow˙def comp˙fun˙subset csgaisfun
    by auto
  ultimately have Fold1(f,a O b) ∈ G by (rule semigr0.prod˙type)
  with ‹∑(A,a) = (∑ (a O b))› show ∑(A,a) ∈ G
```

```
    by simp
  { fix x assume x ∈ X-A
    with A1 have (A ∪ –x″) ∈ FinPow(X) and A ∪ –x″ ≠ 0
      using singleton˙in˙finpow union˙finpow by auto
    moreover have Append(b,x) ∈ bij(—A ∪ –x″—, A ∪ –x″)
    proof -
      note ‹—A— ∈ nat› ‹b ∈ bij(—A—,A)›
      moreover from ‹x ∈ X-A› have x ∉ A by simp
      ultimately have Append(b,x) ∈ bij(succ(—A—), A ∪ –x″)
 by (rule bij˙append˙point)
      with A1 ‹x ∈ X-A› show thesis
 using card˙fin˙add˙one by auto
    qed
    ultimately have (∑(A ∪ –x″,a)) =  (∑  (a O Append(b,x)))
      using sum˙over˙set˙bij by simp
    also have . . . = (∑  Append(a O b, a(x)))
    proof -
      note ‹—A— ∈ nat›
      moreover
      from A1 ‹b ∈ bij(—A—, A)› have
 b : —A— → A and A ⊆ X
 using bij˙def inj˙def using FinPow˙def by auto
      then have b : —A— → X by (rule func1˙1˙L1B)
      moreover from ‹x ∈ X-A› have x ∈ X and a : X → G
 using csgaisfun by auto
      ultimately show thesis using list˙compose˙append
 by simp
    qed
    also have . . . =  (∑(A,a)) + a(x)
    proof -
      note ‹semigr0(G,f)›  ‹n ∈ nat›  ‹a O b : succ(n) → G›
      moreover from ‹x ∈ X-A› have a(x) ∈ G
 using csgaisfun apply˙funtype by simp
      ultimately have
 Fold1(f,Append(a O b, a(x))) = f⟨Fold1(f,a O b),a(x)⟩
 by (rule semigr0.prod˙append)
      with A1 ‹b ∈ bij(—A—,A)› show thesis
 using sum˙over˙set˙bij by simp
    qed
    finally have (∑(A ∪ –x″,a)) = (∑(A,a)) + a(x)
      by simp
  } thus ∀x ∈ X-A. ∑(A ∪ –x″,a) = (∑(A,a)) + a(x)
    by simp
qed

end
```

25 Monoids

theory Monoid˙ZF imports func˙ZF

begin

This theory provides basic facts about monoids.

25.1 Definition and basic properties

In this section we talk about monoids. The notion of a monoid is similar to the notion of a semigroup except that we require the existence of a neutral element. It is also similar to the notion of group except that we don't require existence of the inverse.

Monoid is a set G with an associative operation and a neutral element. The operation is a function on $G \times G$ with values in G. In the context of ZF set theory this means that it is a set of pairs $\langle x, y\rangle$, where $x \in G \times G$ and $y \in G$. In other words the operation is a certain subset of $(G \times G) \times G$. We express all this by defing a predicate IsAmonoid(G,f). Here G is the "carrier" of the group and f is the binary operation on it.

```
definition
  IsAmonoid(G,f) ≡
  f –is associative on″ G ∧
  (∃e∈G. (∀ g∈G. ( (f(⟨e,g⟩) = g) ∧ (f(⟨g,e⟩) = g))))
```

The next locale called "monoid0" defines a context for theorems that concern monoids. In this contex we assume that the pair (G, f) is a monoid. We will use the $\oplus$ symbol to denote the monoid operation (for no particular reason).

```
locale monoid0 =
  fixes G
  fixes 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 assms monoidAsssum IsAmonoid˙def IsAssociative˙def apply˙funtype
  by auto
```

There is only one neutral element in a monoid.

```
lemma (in monoid0) group0˙1˙L2: shows
  ∃!e. e∈G ∧ (∀ g∈G. ( (e⊕g = g) ∧ g⊕e = g))
proof
```

```
  fix e y
  assume e ∈ G ∧ (∀g∈G. e ⊕ g = g ∧ g ⊕ e = g)
    and y ∈ G ∧ (∀g∈G. y ⊕ g = g ∧ g ⊕ y = g)
  then have y⊕e = y y⊕e = e by auto
  thus e = y by simp
next from 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.

```
definition
 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) unit˙is˙neutral:
  assumes A1: e = TheNeutralElement(G,f)
  shows e ∈ G ∧ (∀g∈G. e ⊕ g = g ∧ g ⊕ e = g)
proof -
  let n = THE b. b∈ G ∧ (∀ g∈G. b⊕g = g ∧ g⊕b = g)
  have ∃!b. b∈ G ∧ (∀ g∈G. b⊕g = g ∧ g⊕b = g)
    using group0˙1˙L2 by simp
  then have n∈ G ∧ (∀ g∈G. n⊕g = g ∧ g⊕n = g)
    by (rule theI)
  with A1 show thesis
    using TheNeutralElement˙def by simp
qed
```

The monoid carrier is not empty.

```
lemma (in monoid0) group0˙1˙L3A: shows G≠0
proof -
  have TheNeutralElement(G,f) ∈ G using unit˙is˙neutral
    by simp
  thus thesis by auto
qed
```

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

```
lemma (in monoid0) group0˙1˙L3B: shows range(f) = G
proof
  from monoidAsssum have f : G×G→G
     using IsAmonoid˙def IsAssociative˙def by simp
  then show range(f) ⊆ G
    using func1˙1˙L5B by simp
  show G ⊆ range(f)
  proof
    fix g assume A1: g∈G
```

```
    let e = TheNeutralElement(G,f)
    from A1 have ⟨e,g⟩ ∈ G×G g = f⟨e,g⟩
      using unit˙is˙neutral by auto
    with ‹f : G×G→G› show g ∈ range(f)
      using func1˙1˙L5A by blast
  qed
qed
```

Another way to state that the range of the monoid operation is the whole monoid carrier.

```
lemma (in monoid0) range˙carr: shows f(G×G) = G
  using monoidAsssum IsAmonoid˙def IsAssociative˙def
    group0˙1˙L3B range˙image˙domain by auto
```

In a monoid any neutral element is the neutral element.

```
lemma (in monoid0) group0˙1˙L4:
  assumes A1: e ∈ G ∧ (∀g∈G. e ⊕ g = g ∧ g ⊕ e = g)
  shows e = TheNeutralElement(G,f)
proof -
  let n = THE b. b∈ G ∧ (∀ g∈G. b⊕g = g ∧ g⊕b = g)
  have ∃!b. b∈ G ∧ (∀ g∈G. b⊕g = g ∧ g⊕b = g)
    using group0˙1˙L2 by simp
  moreover note A1
  ultimately have n = e by (rule the˙equality2)
  then show thesis using TheNeutralElement˙def by simp
qed
```

The next lemma shows that if the if we restrict the monoid operation to a subset of G that contains the neutral element, then the neutral element of the monoid operation is also neutral with the restricted operation.

```
lemma (in monoid0) group0˙1˙L5:
  assumes A1: ∀x∈H.∀y∈H. x⊕y ∈ H
  and A2: H⊆G
  and A3: e = TheNeutralElement(G,f)
  and A4: g = restrict(f,H×H)
  and A5: e∈H
  and A6: h∈H
  shows g⟨e,h⟩ = h ∧ g⟨h,e⟩ = h
proof -
  from A4 A6 A5 have
    g⟨e,h⟩ = e⊕h ∧ g⟨h,e⟩ = h⊕e
    using restrict˙if by simp
  with A3 A4 A6 A2 show
    g⟨e,h⟩ = h ∧ g⟨h,e⟩ = h
    using  unit˙is˙neutral by auto
qed
```

The next theorem shows that if the monoid operation is closed on a subset

of G then this set is a (sub)monoid (although we do not define this notion). This fact will be useful when we study subgroups.

```
theorem (in monoid0) group0˙1˙T1:
  assumes A1: H –is closed under″ f
  and A2: H⊆G
  and A3: TheNeutralElement(G,f) ∈ H
  shows  IsAmonoid(H,restrict(f,H×H))
proof -
  let g = restrict(f,H×H)
  let e = TheNeutralElement(G,f)
  from 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 -
  let e = TheNeutralElement(G,f)
  let g = restrict(f,H×H)
  from assms have monoid0(H,g)
    using monoid0˙def monoid0.group0˙1˙T1
    by simp
  moreover have
    e ∈ H ∧ (∀h∈H. g⟨e,h⟩ = h ∧ g⟨h,e⟩ = h)
  proof -
    { fix h assume h ∈ H
      with assms have
 monoid0(G,f)  ∀x∈H.∀y∈H. f⟨x,y⟩ ∈ H
 H⊆G  e = TheNeutralElement(G,f)  g = restrict(f,H×H)
 e ∈ H  h ∈ H
 using monoid0˙def IsOpClosed˙def by auto
      then have g⟨e,h⟩ = h ∧ g⟨h,e⟩ = h
 by (rule monoid0.group0˙1˙L5)
    } hence ∀h∈H. g⟨e,h⟩ = h ∧ g⟨h,e⟩ = h by simp
    with A4 show thesis by simp
  qed
  ultimately have e =  TheNeutralElement(H,g)
    by (rule monoid0.group0˙1˙L4)
  thus thesis by simp
qed
```

If a sum of two elements is not zero, then at least one has to be nonzero.

```
lemma (in monoid0) sum˙nonzero˙elmnt˙nonzero:
  assumes a ⊕ b ≠ TheNeutralElement(G,f)
  shows a ≠ TheNeutralElement(G,f) ∨ b ≠ TheNeutralElement(G,f)
  using assms unit˙is˙neutral by auto

end
```

26 Groups - introduction

```
theory Group˙ZF imports Monoid˙ZF

begin
```

This theory file covers basics of group theory.

26.1 Definition and basic properties of groups

In this section we define the notion of a group and set up the notation for discussing groups. We prove some basic theorems about groups.

To define a group we take a monoid and add a requirement that the right inverse needs to exist for every element of the group.

```
definition
  IsAgroup(G,f) ≡
  (IsAmonoid(G,f) ∧ (∀ g∈G. ∃ b∈G. f⟨g,b⟩ = TheNeutralElement(G,f)))
```

We define the group inverse as the set $\{\langle x, y\rangle \in G \times G : x \cdot y = e\}$, where e is the neutral element of the group. This set (which can be written as $(\cdot)^{-1}\{e\}$) is a certain relation on the group (carrier). Since, as we show later, for every $x \in G$ there is exactly one $y \in G$ such that $x \cdot y = e$ this relation is in fact a function from G to G.

```
definition
  GroupInv(G,f) ≡ −⟨x,y⟩ ∈ G×G. f⟨x,y⟩ = TheNeutralElement(G,f)″
```

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

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

  fixes neut (1)
  defines neut˙def[simp]: 1 ≡ TheNeutralElement(G,P)

  fixes groper (infixl · 70)
  defines groper˙def[simp]: a · b ≡ P⟨a,b⟩

  fixes inv (˙⁻¹ [90] 91)
  defines inv˙def[simp]: x⁻¹ ≡ GroupInv(G,P)(x)
```

First we show a lemma that says that we can use theorems proven in the monoid0 context (locale).

```
lemma (in group0) group0˙2˙L1: shows monoid0(G,P)
  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 assms IsAgroup˙def by simp
```

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

```
lemma (in group0) group0˙2˙L2:
  shows 1∈G ∧ (∀g∈G.(1·g = g ∧ g·1 = g))
  using group0˙2˙L1 monoid0.unit˙is˙neutral by simp
```

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

```
lemma (in group0) group˙op˙closed: assumes a∈G  b∈G
  shows a·b ∈ G using assms 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 assms IsAgroup˙def IsAmonoid˙def
    IsAssociative˙def group˙op˙closed by simp
```

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

```
lemma (in group0) group˙oper˙assocA: shows P : G×G→G
  using groupAssum IsAgroup˙def IsAmonoid˙def IsAssociative˙def
  by simp
```

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

```
theorem (in group0) group0˙2˙T1:
  assumes A1: g∈G and A2: b∈G and A3: g·b = 1
  shows b·g = 1
proof -
  from A2 groupAssum obtain c where I: c ∈ G ∧ b·c = 1
    using IsAgroup˙def by auto
  then have c∈G by simp
  have 1∈G using group0˙2˙L2 by simp
  with A1 A2 I have b·g =  b·(g·(b·c))
    using group˙op˙closed group0˙2˙L2 group˙oper˙assoc
    by simp
  also from  A1 A2 ‹c∈G› have b·(g·(b·c)) = b·(g·b·c)
    using group˙oper˙assoc by simp
  also from A3 A2 I have b·(g·b·c)= 1 using group0˙2˙L2 by simp
  finally show b·g = 1 by simp
qed
```

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

```
lemma (in group0) group0˙2˙L4:
  assumes A1: x∈G shows ∃!y. y∈G ∧ x·y = 1
```

```
proof
  from A1 groupAssum show ∃y. y∈G ∧  x·y = 1
    using IsAgroup˙def by auto
  fix y n
  assume A2: y∈G ∧  x·y = 1 and A3:n∈G ∧ x·n = 1 show y=n
  proof -
    from A1 A2 have T1: y·x = 1
      using group0˙2˙T1 by simp
    from A2 A3 have y = y·(x·n)
      using group0˙2˙L2 by simp
    also from A1 A2 A3 have ... = (y·x)·n
      using group˙oper˙assoc by blast
    also from T1 A3 have ... = n
      using group0˙2˙L2 by simp
    finally show y=n by simp
  qed
qed
```

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

```
theorem group0˙2˙T2:
  assumes A1: IsAgroup(G,f) shows GroupInv(G,f) : G→G
proof -
  have GroupInv(G,f) ⊆ G×G using GroupInv˙def by auto
  moreover from A1 have
    ∀x∈G. ∃!y. y∈G ∧ ⟨x,y⟩ ∈ GroupInv(G,f)
    using group0˙def group0.group0˙2˙L4 GroupInv˙def by simp
  ultimately show thesis using func1˙1˙L11 by simp
qed
```

We can think about the group inverse (the function) as the inverse image of the neutral element. Recall that in Isabelle f-(A) denotes the inverse image of the set A.

```
theorem (in group0) group0˙2˙T3: shows P-–1″ = GroupInv(G,P)
proof -
  from groupAssum have P : G×G → G
    using IsAgroup˙def IsAmonoid˙def IsAssociative˙def
    by simp
  then show P-–1″ = GroupInv(G,P)
    using func1˙1˙L14 GroupInv˙def by auto
qed
```

The inverse is in the group.

```
lemma (in group0) inverse˙in˙group: assumes A1: x∈G shows x⁻¹∈G
proof -
  from groupAssum have GroupInv(G,P) : G→G using group0˙2˙T2 by simp
  with A1 show thesis using apply˙type by simp
qed
```

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

```
lemma (in group0) group0˙2˙L6:
  assumes A1: x∈G shows x·x⁻¹ = 1 ∧ x⁻¹·x = 1
proof
  from groupAssum have GroupInv(G,P) : G→G
    using group0˙2˙T2 by simp
  with A1 have ⟨x,x⁻¹⟩ ∈  GroupInv(G,P)
    using apply˙Pair by simp
  then show x·x⁻¹ = 1 using GroupInv˙def by simp
  with A1 show x⁻¹·x = 1 using inverse˙in˙group group0˙2˙T1
    by blast
qed
```

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

```
lemma (in group0) group0˙2˙L7:
  assumes A1: a∈G and A2: b∈G and A3: a·b = a
  shows b=1
proof -
  from A3 have a⁻¹ · (a·b) = a⁻¹·a by simp
  with A1 A2 show thesis using
    inverse˙in˙group group˙oper˙assoc group0˙2˙L6 group0˙2˙L2
    by simp
qed
```

See the comment to group0˙2˙L7.

```
lemma (in group0) group0˙2˙L8:
  assumes A1: a∈G and A2: b∈G and A3: a·b = b
  shows a=1
proof -
  from A3 have (a·b)·b⁻¹  = b·b⁻¹ by simp
  with A1 A2 have a·(b·b⁻¹)  = b·b⁻¹ using
    inverse˙in˙group group˙oper˙assoc by simp
  with A1 A2 show thesis
    using group0˙2˙L6 group0˙2˙L2 by simp
qed
```

The inverse of the neutral element is the neutral element.

```
lemma (in group0) group˙inv˙of˙one: shows 1⁻¹ = 1
  using group0˙2˙L2 inverse˙in˙group group0˙2˙L6 group0˙2˙L7 by blast
```

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

```
lemma (in group0) group0˙2˙L8A:
  assumes A1: a∈G and A2: a⁻¹ = 1
  shows a = 1
proof -
  from A1 have a·a⁻¹ = 1 using group0˙2˙L6 by simp
  with A1 A2 show a = 1 using group0˙2˙L2 by simp
qed
```

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

```
lemma (in group0) group0˙2˙L8B:
  assumes a∈G and a ≠ 1
  shows a⁻¹ ≠ 1 using assms group0˙2˙L8A by auto
```

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

```
lemma (in group0) group0˙2˙L8C:
  assumes a∈G and a⁻¹ ≠ 1
  shows a≠1
  using assms group0˙2˙L8A group˙inv˙of˙one by auto
```

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

```
lemma (in group0) group0˙2˙L9:
  assumes A1: a∈G and A2: b∈G and A3: a·b = 1
  shows a = b⁻¹ and b = a⁻¹
proof -
  from A3 have a·b·b⁻¹ = 1·b⁻¹ by simp
  with A1 A2 have a·(b·b⁻¹) = 1·b⁻¹ using
    inverse˙in˙group group˙oper˙assoc by simp
  with A1 A2 show a = b⁻¹ using
    group0˙2˙L6 inverse˙in˙group group0˙2˙L2 by simp
  from A3 have a⁻¹·(a·b) = a⁻¹·1 by simp
  with A1 A2 show b = a⁻¹ using
    inverse˙in˙group group˙oper˙assoc group0˙2˙L6 group0˙2˙L2
    by simp
qed
```

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

```
lemma (in group0) group0˙2˙L9A:
  assumes A1: ∀g∈G. b(g) ∈ G ∧ g·b(g) = 1
  shows ∀g∈G. b(g) = g⁻¹
proof
  fix g assume g∈G
  moreover from A1 ‹g∈G› have b(g) ∈ G by simp
  moreover from A1 ‹g∈G› have g·b(g) = 1 by simp
  ultimately show b(g) = g⁻¹ by (rule group0˙2˙L9)
qed
```

What is the inverse of a product?

```
lemma (in group0) group˙inv˙of˙two:
  assumes A1: a∈G and A2: b∈G
  shows  b⁻¹·a⁻¹ = (a·b)⁻¹
proof -
  from A1 A2 have
    b⁻¹∈G  a⁻¹∈G  a·b∈G  b⁻¹·a⁻¹ ∈ G
```

```
    using inverse˙in˙group group˙op˙closed
    by auto
  from A1 A2 ‹b⁻¹·a⁻¹ ∈ G›  have a·b·(b⁻¹·a⁻¹) = a·(b·(b⁻¹·a⁻¹))
    using group˙oper˙assoc by simp
  moreover from A2 ‹b⁻¹∈G› ‹a⁻¹∈G› have b·(b⁻¹·a⁻¹) = b·b⁻¹·a⁻¹
    using group˙oper˙assoc by simp
  moreover from A2 ‹a⁻¹∈G› have b·b⁻¹·a⁻¹ = a⁻¹
     using group0˙2˙L6 group0˙2˙L2 by simp
  ultimately have a·b·(b⁻¹·a⁻¹) = a·a⁻¹
    by simp
  with A1 have a·b·(b⁻¹·a⁻¹) = 1
    using group0˙2˙L6 by simp
  with ‹a·b ∈ G›  ‹b⁻¹·a⁻¹ ∈ G› show b⁻¹·a⁻¹ = (a·b)⁻¹
    using group0˙2˙L9 by simp
qed
```

What is the inverse of a product of three elements?

```
lemma (in group0) group˙inv˙of˙three:
  assumes A1: a∈G  b∈G  c∈G
  shows
  (a·b·c)⁻¹ = c⁻¹·(a·b)⁻¹
  (a·b·c)⁻¹ = c⁻¹·(b⁻¹·a⁻¹)
  (a·b·c)⁻¹ = c⁻¹·b⁻¹·a⁻¹
proof -
  from A1 have T:
    a·b ∈ G  a⁻¹ ∈ G  b⁻¹ ∈ G   c⁻¹ ∈ G
    using group˙op˙closed inverse˙in˙group by auto
  with A1 show
    (a·b·c)⁻¹ = c⁻¹·(a·b)⁻¹ and (a·b·c)⁻¹ = c⁻¹·(b⁻¹·a⁻¹)
     using group˙inv˙of˙two by auto
   with T show (a·b·c)⁻¹ = c⁻¹·b⁻¹·a⁻¹ using group˙oper˙assoc
     by simp
qed
```

The inverse of the inverse is the element.

```
lemma (in group0) group˙inv˙of˙inv:
  assumes a∈G shows a = (a⁻¹)⁻¹
  using assms inverse˙in˙group group0˙2˙L6 group0˙2˙L9
  by simp
```

Group inverse is nilpotent, therefore a bijection and involution.

```
lemma (in group0) group˙inv˙bij:
  shows GroupInv(G,P) O GroupInv(G,P) = id(G) and GroupInv(G,P) ∈ bij(G,G)
and
  GroupInv(G,P) = converse(GroupInv(G,P))
proof -
  have I: GroupInv(G,P): G→G using groupAssum group0˙2˙T2 by simp
  then have GroupInv(G,P) O GroupInv(G,P): G→G and id(G):G→G
    using comp˙fun id˙type by auto
```

```
  moreover
  { fix g assume g∈G
    with I have (GroupInv(G,P) O GroupInv(G,P))(g) = id(G)(g)
      using comp˙fun˙apply group˙inv˙of˙inv id˙conv by simp
  } hence ∀g∈G. (GroupInv(G,P) O GroupInv(G,P))(g) = id(G)(g) by simp
  ultimately show GroupInv(G,P) O GroupInv(G,P) = id(G)
    by (rule func˙eq)
  with I show GroupInv(G,P) ∈ bij(G,G) using nilpotent˙imp˙bijective
    by simp
  with ‹GroupInv(G,P) O GroupInv(G,P) = id(G)› show
    GroupInv(G,P) = converse(GroupInv(G,P)) using comp˙id˙conv by simp
qed
```

A set comprehension form of the image of a set under the group inverse.

```
lemma (in group0) ginv˙image: assumes V⊆G
  shows GroupInv(G,P)(V) ⊆ G and GroupInv(G,P)(V) = –g⁻¹. g ∈ V″
proof -
  from assms have I: GroupInv(G,P)(V) = –GroupInv(G,P)(g). g∈V″
    using groupAssum group0˙2˙T2 func˙imagedef by blast
  thus GroupInv(G,P)(V) = –g⁻¹. g ∈ V″  by simp
  show GroupInv(G,P)(V) ⊆ G using groupAssum group0˙2˙T2 func1˙1˙L6(2) by
blast
qed
```

Inverse of an element that belongs to the inverse of the set belongs to the set.

```
lemma (in group0) ginv˙image˙el: assumes V⊆G g ∈ GroupInv(G,P)(V)
  shows g⁻¹ ∈ V
  using assms ginv˙image group˙inv˙of˙inv by auto
```

For the group inverse the image is the same as inverse image.

```
lemma (in group0) inv˙image˙vimage: shows GroupInv(G,P)(V) = GroupInv(G,P)-(V)
  using group˙inv˙bij vimage˙converse by simp
```

If the unit is in a set then it is in the inverse of that set.

```
lemma (in group0) neut˙inv˙neut: assumes A⊆G and 1∈A
  shows 1 ∈ GroupInv(G,P)(A)
proof -
  have GroupInv(G,P):G→G using groupAssum group0˙2˙T2 by simp
  with assms have 1⁻¹ ∈ GroupInv(G,P)(A) using func˙imagedef by auto
  then show thesis using group˙inv˙of˙one by simp
qed
```

The group inverse is onto.

```
lemma (in group0) group˙inv˙surj: shows GroupInv(G,P)(G) = G
  using group˙inv˙bij bij˙def surj˙range˙image˙domain by auto
```

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

```
lemma (in group0) group0˙2˙L11:
  assumes A1: a∈G  b∈G and A2: a⁻¹·b = 1
  shows a=b
proof -
  from A1 A2 have a⁻¹ ∈ G  b∈G  a⁻¹·b = 1
    using inverse˙in˙group by auto
  then have b = (a⁻¹)⁻¹ by (rule group0˙2˙L9)
  with A1 show a=b using group˙inv˙of˙inv by simp
qed
```

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

```
lemma (in group0) group0˙2˙L11A:
  assumes A1: a∈G  b∈G and A2: a·b⁻¹ = 1
  shows a=b
proof -
  from A1 A2 have a ∈ G  b⁻¹∈G  a·b⁻¹ = 1
    using inverse˙in˙group by auto
  then have a = (b⁻¹)⁻¹ by (rule group0˙2˙L9)
  with A1 show a=b using group˙inv˙of˙inv by simp
qed
```

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

```
lemma (in group0) group0˙2˙L11B:
  assumes A1: a∈G and A2: b⁻¹ ≠ a
  shows a⁻¹ ≠ b
proof -
  { assume a⁻¹ = b
    then have (a⁻¹)⁻¹ = b⁻¹ by simp
    with A1 A2 have False using group˙inv˙of˙inv
      by simp
  } then show a⁻¹ ≠ b by auto
qed
```

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

```
lemma (in group0) group0˙2˙L12:
  assumes A1: a∈G  b∈G
  shows
  (a·b⁻¹)⁻¹ = b·a⁻¹
  (a⁻¹·b)⁻¹ = b⁻¹·a
proof -
  from A1 have
    (a·b⁻¹)⁻¹ = (b⁻¹)⁻¹· a⁻¹ and (a⁻¹·b)⁻¹ = b⁻¹·(a⁻¹)⁻¹
    using inverse˙in˙group group˙inv˙of˙two by auto
  with A1 show  (a·b⁻¹)⁻¹ = b·a⁻¹  (a⁻¹·b)⁻¹ = b⁻¹·a
    using group˙inv˙of˙inv by auto
qed
```

A couple useful rearrangements with three elements: we can insert a $b \cdot b^{-1}$

between two group elements (another version) and one about a product of an element and inverse of a product, and two others.

```
lemma (in group0) group0˙2˙L14A:
  assumes A1: a∈G  b∈G  c∈G
  shows
  a·c⁻¹= (a·b⁻¹)·(b·c⁻¹)
  a⁻¹·c = (a⁻¹·b)·(b⁻¹·c)
  a·(b·c)⁻¹ = a·c⁻¹·b⁻¹
  a·(b·c⁻¹) = a·b·c⁻¹
  (a·b⁻¹·c⁻¹)⁻¹ = c·b·a⁻¹
  a·b·c⁻¹·(c·b⁻¹) = a
  a·(b·c)·c⁻¹ = a·b
proof -
  from A1 have T:
    a⁻¹ ∈ G  b⁻¹∈G  c⁻¹∈G
    a⁻¹·b ∈ G  a·b⁻¹ ∈ G  a·b ∈ G
    c·b⁻¹ ∈ G  b·c ∈ G
    using inverse˙in˙group group˙op˙closed
    by auto
   from A1 T have
     a·c⁻¹ =  a·(b⁻¹·b)·c⁻¹
     a⁻¹·c =  a⁻¹·(b·b⁻¹)·c
    using group0˙2˙L2 group0˙2˙L6 by auto
   with A1 T show
     a·c⁻¹= (a·b⁻¹)·(b·c⁻¹)
     a⁻¹·c = (a⁻¹·b)·(b⁻¹·c)
     using group˙oper˙assoc by auto
  from A1 have a·(b·c)⁻¹ = a·(c⁻¹·b⁻¹)
    using group˙inv˙of˙two by simp
  with A1 T show a·(b·c)⁻¹ =a·c⁻¹·b⁻¹
    using group˙oper˙assoc by simp
  from A1 T show a·(b·c⁻¹) = a·b·c⁻¹
    using group˙oper˙assoc by simp
  from A1 T show  (a·b⁻¹·c⁻¹)⁻¹ = c·b·a⁻¹
    using group˙inv˙of˙three  group˙inv˙of˙inv
    by simp
  from T have a·b·c⁻¹·(c·b⁻¹) = a·b·(c⁻¹·(c·b⁻¹))
    using group˙oper˙assoc by simp
  also from A1 T have ... =  a·b·b⁻¹
    using group˙oper˙assoc group0˙2˙L6 group0˙2˙L2
    by simp
  also from A1 T have ... = a·(b·b⁻¹)
    using group˙oper˙assoc by simp
  also from A1 have ... = a
    using group0˙2˙L6 group0˙2˙L2 by simp
  finally show a·b·c⁻¹·(c·b⁻¹) = a by simp
  from A1 T have a·(b·c)·c⁻¹ =  a·(b·(c·c⁻¹))
    using group˙oper˙assoc by simp
  also from A1 T have ... = a·b
```

```
    using  group0˙2˙L6 group0˙2˙L2 by simp
  finally show a·(b·c)·c⁻¹ = a·b
    by simp
qed
```

A simple equation to solve

```
lemma (in group0) simple˙equation0:
  assumes a∈G  b∈G c∈G a·b⁻¹ = c⁻¹
  shows c = b·a⁻¹
proof -
  from assms(4) have (a·b⁻¹)⁻¹ = (c⁻¹)⁻¹ by simp
  with assms(1,2,3) show c = b·a⁻¹ using group0˙2˙L12(1) group˙inv˙of˙inv by simp
qed
```

Another simple equation

```
lemma (in group0) simple˙equation1:
  assumes a∈G  b∈G c∈G a⁻¹·b = c⁻¹
  shows c = b⁻¹·a
proof -
  from assms(4) have (a⁻¹·b)⁻¹ = (c⁻¹)⁻¹ by simp
  with assms(1,2,3) show c = b⁻¹·a using group0˙2˙L12(2) group˙inv˙of˙inv by simp
qed
```

Another lemma about rearranging a product of four group elements.

```
lemma (in group0) group0˙2˙L15:
  assumes A1: a∈G  b∈G  c∈G  d∈G
  shows (a·b)·(c·d)⁻¹ = a·(b·d⁻¹)·a⁻¹·(a·c⁻¹)
proof -
  from A1 have T1:
    d⁻¹∈G  c⁻¹∈G a·b∈G a·(b·d⁻¹)∈G
    using inverse˙in˙group group˙op˙closed
    by auto
  with A1 have (a·b)·(c·d)⁻¹ = (a·b)·(d⁻¹·c⁻¹)
    using group˙inv˙of˙two by simp
  also from A1 T1 have ... = a·(b·d⁻¹)·c⁻¹
    using group˙oper˙assoc by simp
  also from A1 T1 have ... = a·(b·d⁻¹)·a⁻¹·(a·c⁻¹)
    using group0˙2˙L14A by blast
  finally show thesis by simp
qed
```

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

```
lemma (in group0) inv˙cancel˙two:
  assumes A1: a∈G  b∈G
  shows
  a·b⁻¹·b = a
  a·b·b⁻¹ = a
  a⁻¹·(a·b) = b
```

```
  a·(a⁻¹·b) = b
proof -
  from A1 have
    a·b⁻¹·b = a·(b⁻¹·b)   a·b·b⁻¹ = a·(b·b⁻¹)
    a⁻¹·(a·b) = a⁻¹·a·b   a·(a⁻¹·b) = a·a⁻¹·b
    using inverse˙in˙group group˙oper˙assoc by auto
  with A1 show
    a·b⁻¹·b = a
    a·b·b⁻¹ = a
    a⁻¹·(a·b) = b
    a·(a⁻¹·b) = b
    using group0˙2˙L6 group0˙2˙L2 by auto
qed
```

Another lemma about cancelling with two group elements.

```
lemma (in group0) group0˙2˙L16A:
  assumes A1: a∈G  b∈G
  shows a·(b·a)⁻¹ = b⁻¹
proof -
  from A1 have (b·a)⁻¹ = a⁻¹·b⁻¹  b⁻¹ ∈ G
    using group˙inv˙of˙two inverse˙in˙group by auto
  with A1 show a·(b·a)⁻¹ = b⁻¹ using inv˙cancel˙two
    by simp
qed
```

Some other identities with three element and cancelling.

```
lemma (in group0) cancel˙middle:
  assumes a∈G  b∈G c∈G
  shows
    (a·b)⁻¹·(a·c) = b⁻¹·c
    (a·b)·(c·b)⁻¹ = a·c⁻¹
    a⁻¹·(a·b·c)·c⁻¹ = b
    a·(b·c⁻¹)·c = a·b
    a·b⁻¹·(b·c⁻¹) = a·c⁻¹
proof -
  from assms have (a·b)⁻¹·(a·c) = b⁻¹·(a⁻¹·(a·c))
   using group˙inv˙of˙two inverse˙in˙group group˙oper˙assoc group˙op˙closed by auto
  with assms(1,3) show (a·b)⁻¹·(a·c) = b⁻¹·c using inv˙cancel˙two(3) by simp
  from assms have (a·b)·(c·b)⁻¹ = a·(b·(b⁻¹·c⁻¹))
   using group˙inv˙of˙two inverse˙in˙group group˙oper˙assoc group˙op˙closed by auto
  with assms show (a·b)·(c·b)⁻¹ =a·c⁻¹ using inverse˙in˙group inv˙cancel˙two(4) by
simp
  from assms have a⁻¹·(a·b·c)·c⁻¹ = (a⁻¹·a)·b·(c·c⁻¹)
    using inverse˙in˙group group˙oper˙assoc group˙op˙closed by auto
  with assms show a⁻¹·(a·b·c)·c⁻¹ = b using group0˙2˙L6 group0˙2˙L2 by simp
  from assms have a·(b·c⁻¹)·c = a·b·(c⁻¹·c) using inverse˙in˙group group˙oper˙assoc
group˙op˙closed
    by simp
  with assms show a·(b·c⁻¹)·c = a·b using group˙op˙closed group0˙2˙L6 group0˙2˙L2
```

by simp
from assms have $a \cdot b^{-1} \cdot (b \cdot c^{-1}) = a \cdot (b^{-1} \cdot b) \cdot c^{-1}$ using inverse˙in˙group group˙oper˙assoc group˙op˙closed
by simp
with assms show $a \cdot b^{-1} \cdot (b \cdot c^{-1}) = a \cdot c^{-1}$ using group0˙2˙L6 group0˙2˙L2 by simp
qed

Adding a neutral element to a set that is closed under the group operation results in a set that is closed under the group operation.

lemma (in group0) group0˙2˙L17:
assumes $H \subseteq G$
and H –is closed under˝ P
shows ($H \cup$ –**1**˝) –is closed under˝ P
using assms IsOpClosed˙def group0˙2˙L2 by auto

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

lemma (in group0) group0˙2˙L18:
assumes A1: $a \in G$ $b \in G$
and A2: $c = a \cdot b$
shows $c \cdot b^{-1} = a$ $a^{-1} \cdot c = b$
proof-
from A2 A1 have $c \cdot b^{-1} = a \cdot (b \cdot b^{-1})$ $a^{-1} \cdot c = (a^{-1} \cdot a) \cdot b$
using inverse˙in˙group group˙oper˙assoc by auto
moreover from A1 have $a \cdot (b \cdot b^{-1}) = a$ $(a^{-1} \cdot a) \cdot b = b$
using group0˙2˙L6 group0˙2˙L2 by auto
ultimately show $c \cdot b^{-1} = a$ $a^{-1} \cdot c = b$
by auto
qed

We can cancel an element on the right from both sides of an equation.

lemma (in group0) cancel˙right: assumes $a \in G$ $b \in G$ $c \in G$ $a \cdot b = c \cdot b$
shows $a = c$
proof -
from assms(4) have $a \cdot b \cdot b^{-1} = c \cdot b \cdot b^{-1}$ by simp
with assms(1,2,3) show thesis using inv˙cancel˙two(2) by simp
qed

We can cancel an element on the left from both sides of an equation.

lemma (in group0) cancel˙left: assumes $a \in G$ $b \in G$ $c \in G$ $a \cdot b = a \cdot c$
shows $b = c$
proof -
from assms(4) have $a^{-1} \cdot (a \cdot b) = a^{-1} \cdot (a \cdot c)$ by simp
with assms(1,2,3) show thesis using inv˙cancel˙two(3) by simp
qed

Multiplying different group elements by the same factor results in different group elements.

lemma (in group0) group0˙2˙L19:

```
  assumes A1: a∈G  b∈G  c∈G and A2: a≠b
  shows a·c ≠ b·c and c·a ≠ c·b
proof -
  { assume a·c = b·c ∨ c·a =c·b
    then have a·c·c⁻¹ = b·c·c⁻¹ ∨ c⁻¹·(c·a) = c⁻¹·(c·b)
      by auto
    with A1 A2 have False using inv˙cancel˙two by simp
  } then show a·c ≠ b·c and c·a ≠ c·b by auto
qed
```

26.2 Subgroups

There are two common ways to define subgroups. One requires that the group operation is closed in the subgroup. The second one defines subgroup as a subset of a group which is itself a group under the group operations. We use the second approach because it results in shorter definition.

The rest of this section is devoted to proving the equivalence of these two definitions of the notion of a subgroup.

A pair (H, P) is a subgroup if H forms a group with the operation P restricted to $H \times H$. It may be surprising that we don't require H to be a subset of G. This however can be inferred from the definition if the pair (G, P) is a group, see lemma group0˙3˙L2.

```
definition
  IsAsubgroup(H,P) ≡ IsAgroup(H, restrict(P,H×H))
```

Formally the group operation in a subgroup is different than in the group as they have different domains. Of course we want to use the original operation with the associated notation in the subgroup. The next couple of lemmas will allow for that.

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

```
lemma group0˙3˙L1:
  assumes A1: IsAsubgroup(H,f)
  and A2: n = TheNeutralElement(H,restrict(f,H×H))
  shows n ∈ H
  ∀h∈H. restrict(f,H×H)⟨n,h ⟩ = h
  ∀h∈H. restrict(f,H×H)⟨h,n⟩ = h
proof -
  let b = restrict(f,H×H)
  let e = TheNeutralElement(H,restrict(f,H×H))
  from A1 have group0(H,b)
    using IsAsubgroup˙def group0˙def by simp
  then have I:
```

```
    e ∈ H ∧ (∀h∈H. (b⟨e,h ⟩ = h ∧ b⟨h,e⟩ = h))
    by (rule group0.group0˙2˙L2)
  with A2 show n ∈ H by simp
  from A2 I show ∀h∈H. b⟨n,h⟩ = h and ∀h∈H. b⟨h,n⟩ = h
    by auto
qed
```

A subgroup is contained in the group.

```
lemma (in group0) group0˙3˙L2:
  assumes A1: IsAsubgroup(H,P)
  shows H ⊆ G
proof
  fix h assume h∈H
  let b = restrict(P,H×H)
  let n = TheNeutralElement(H,restrict(P,H×H))
   from A1 have b ∈ H×H→H
    using IsAsubgroup˙def IsAgroup˙def
      IsAmonoid˙def IsAssociative˙def by simp
  moreover from A1 ‹h∈H› have ⟨ n,h⟩ ∈ H×H
    using group0˙3˙L1 by simp
  moreover from A1 ‹h∈H› have h = b⟨n,h ⟩
    using group0˙3˙L1 by simp
  ultimately have ⟨⟨n,h⟩,h⟩ ∈ b
    using func1˙1˙L5A by blast
  then have ⟨⟨n,h⟩,h⟩ ∈ P using restrict˙subset by auto
  moreover from groupAssum have P:G×G→G
    using IsAgroup˙def IsAmonoid˙def IsAssociative˙def
    by simp
  ultimately show h∈G using func1˙1˙L5
    by blast
qed
```

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

```
lemma (in group0) group0˙3˙L3:
  assumes IsAsubgroup(H,P)
  shows ∀h∈H. 1·h = h ∧ h·1 = h
  using assms groupAssum group0˙3˙L2 group0˙2˙L2
  by auto
```

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

```
lemma (in group0) group0˙3˙L4: assumes A1: IsAsubgroup(H,P)
  shows TheNeutralElement(H,restrict(P,H×H)) = 1
proof -
  let n = TheNeutralElement(H,restrict(P,H×H))
  from A1 have n ∈ H using group0˙3˙L1 by simp
  with groupAssum A1 have n∈G using  group0˙3˙L2 by auto
  with A1 ‹n ∈ H› show thesis using
     group0˙3˙L1 restrict˙if group0˙2˙L7 by simp
```

```
qed
```

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

```
lemma (in group0) group0˙3˙L5: assumes A1: IsAsubgroup(H,P)
  shows 1 ∈ H
proof -
  from A1 show 1∈H using group0˙3˙L1 group0˙3˙L4
    by fast
qed
```

Subgroups are closed with respect to the group operation.

```
lemma (in group0) group0˙3˙L6: assumes A1: IsAsubgroup(H,P)
  and A2: a∈H b∈H
  shows a·b ∈ H
proof -
  let f = restrict(P,H×H)
  from A1 have monoid0(H,f) using
    IsAsubgroup˙def IsAgroup˙def monoid0˙def by simp
  with A2 have f (⟨a,b⟩) ∈ H using monoid0.group0˙1˙L1
    by blast
 with A2 show a·b ∈ H using restrict˙if by simp
qed
```

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

```
lemma group0˙3˙L7A:
  assumes A1: IsAgroup(G,f)
  and A2: IsAsubgroup(H,f) and A3: g = restrict(f,H×H)
  shows GroupInv(G,f) ∩ H×H = GroupInv(H,g)
proof -
  let e = TheNeutralElement(G,f)
  let e₁ = TheNeutralElement(H,g)
  from A1 have group0(G,f) using group0˙def by simp
  from A2 A3 have group0(H,g)
    using IsAsubgroup˙def group0˙def by simp
  from ‹group0(G,f)› A2 A3  have GroupInv(G,f) = f-‒e₁″
    using group0.group0˙3˙L4 group0.group0˙2˙T3
    by simp
  moreover have g-‒e₁″ = f-‒e₁″ ∩ H×H
  proof -
    from A1 have f ∈ G×G→G
      using IsAgroup˙def IsAmonoid˙def IsAssociative˙def
      by simp
    moreover from A2 ‹group0(G,f)› have H×H ⊆ G×G
      using group0.group0˙3˙L2 by auto
    ultimately show g-‒e₁″ = f-‒e₁″ ∩ H×H
      using A3 func1˙2˙L1 by simp
```

```
  qed
  moreover from A3 ‹group0(H,g)› have GroupInv(H,g) = g-`e1`
    using group0.group0˙2˙T3 by simp
  ultimately show thesis by simp
qed
```

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

```
theorem (in group0) group0˙3˙T1:
  assumes A1: IsAsubgroup(H,P)
  and A2: g = restrict(P,H×H)
  shows GroupInv(H,g) = restrict(GroupInv(G,P),H)
proof -
  from groupAssum have GroupInv(G,P) : G→G
    using group0˙2˙T2 by simp
  moreover from A1 A2 have GroupInv(H,g) : H→H
    using IsAsubgroup˙def group0˙2˙T2 by simp
  moreover from A1 have H ⊆ G
    using group0˙3˙L2 by simp
  moreover from groupAssum A1 A2 have
    GroupInv(G,P) ∩ H×H = GroupInv(H,g)
    using group0˙3˙L7A by simp
  ultimately show thesis
    using func1˙2˙L3 by simp
qed
```

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

```
theorem (in group0) group0˙3˙T2:
  assumes IsAsubgroup(H,P)
  and g = restrict(P,H×H)
  shows ∀h∈H. GroupInv(H,g)(h) = h^-1
  using assms group0˙3˙T1 restrict˙if by simp
```

Subgroups are closed with respect to taking the group inverse.

```
theorem (in group0) group0˙3˙T3A:
  assumes A1: IsAsubgroup(H,P) and A2: h∈H
  shows h^-1∈ H
proof -
  let g = restrict(P,H×H)
  from A1 have  GroupInv(H,g) ∈ H→H
    using IsAsubgroup˙def group0˙2˙T2 by simp
  with A2 have GroupInv(H,g)(h) ∈ H
    using apply˙type by simp
  with A1 A2 show h^-1∈ H using group0˙3˙T2 by simp
qed
```

The next theorem states that a nonempty subset of a group G that is closed under the group operation and taking the inverse is a subgroup of the group.

```
theorem (in group0) group0˙3˙T3:
  assumes A1: H≠0
  and A2: H⊆G
  and A3: H –is closed under″ P
  and A4: ∀x∈H. x⁻¹ ∈ H
  shows IsAsubgroup(H,P)
proof -
  let g = restrict(P,H×H)
  let n = TheNeutralElement(H,g)
  from A3 have I: ∀x∈H.∀y∈H. x·y ∈ H
    using IsOpClosed˙def by simp
  from A1 obtain x where x∈H by auto
  with A4 I A2 have 1∈H
    using group0˙2˙L6 by blast
  with A3 A2 have T2: IsAmonoid(H,g)
    using group0˙2˙L1 monoid0.group0˙1˙T1
    by simp
  moreover have ∀h∈H.∃b∈H. g⟨h,b⟩ = n
  proof
    fix h assume h∈H
    with A4 A2 have h·h⁻¹ = 1
      using group0˙2˙L6 by auto
    moreover from groupAssum A2 A3 ‹1∈H› have 1 = n
      using IsAgroup˙def group0˙1˙L6 by auto
    moreover from A4 ‹h∈H› have g⟨h,h⁻¹⟩ = h·h⁻¹
      using restrict˙if by simp
    ultimately have g⟨h,h⁻¹⟩ = n by simp
    with A4 ‹h∈H› show ∃b∈H. g⟨h,b⟩ = n by auto
  qed
  ultimately show IsAsubgroup(H,P) using
    IsAsubgroup˙def IsAgroup˙def by simp
qed
```

Intersection of subgroups is a subgroup. This lemma is obsolete and should be replaced by subgroup˙inter.

```
lemma group0˙3˙L7:
  assumes A1: IsAgroup(G,f)
  and A2: IsAsubgroup(H₁,f)
  and A3: IsAsubgroup(H₂,f)
  shows IsAsubgroup(H₁∩H₂,restrict(f,H₁×H₁))
proof -
  let e = TheNeutralElement(G,f)
  let g = restrict(f,H₁×H₁)
  from A1 have I: group0(G,f)
    using group0˙def by simp
  from A2 have group0(H₁,g)
    using IsAsubgroup˙def group0˙def by simp
  moreover have H₁∩H₂ ≠ 0
  proof -
```

```
    from A1 A2 A3 have e ∈ H1∩H2
      using group0˙def group0.group0˙3˙L5 by simp
    thus thesis by auto
  qed
  moreover have H1∩H2 ⊆ H1 by auto
  moreover from A2 A3 I ‹H1∩H2 ⊆ H1› have
    H1∩H2 –is closed under″ g
    using group0.group0˙3˙L6 IsOpClosed˙def
      func˙ZF˙4˙L7 func˙ZF˙4˙L5 by simp
  moreover from A2 A3 I have
    ∀x ∈ H1∩H2. GroupInv(H1,g)(x) ∈ H1∩H2
    using group0.group0˙3˙T2 group0.group0˙3˙T3A
    by simp
  ultimately show thesis
    using group0.group0˙3˙T3 by simp
qed
```

Intersection of subgroups is a subgroup.

```
lemma (in group0) subgroup˙inter: assumes IsAsubgroup(H1,P) and IsAsubgroup(H2,P)
  shows IsAsubgroup(H1∩H2,P)
proof -
  from assms have H1∩H2 ≠ 0 using group0˙3˙L5 by auto
  moreover from assms have H1∩H2 ⊆ G using group0˙3˙L2 by auto
  moreover from assms have H1∩H2 –is closed under″ P
    unfolding IsOpClosed˙def using group0˙3˙L6 func˙ZF˙4˙L7 func˙ZF˙4˙L5 by simp
  moreover from assms have ∀x ∈ H1∩H2. x⁻¹ ∈ H1∩H2
    using group0˙3˙T2 group0˙3˙T3A by simp
  ultimately show thesis using group0˙3˙T3 by auto
qed
```

The range of the subgroup operation is the whole subgroup.

```
lemma image˙subgr˙op: assumes A1: IsAsubgroup(H,P)
  shows restrict(P,H×H)(H×H) = H
proof -
  from A1 have monoid0(H,restrict(P,H×H))
    using IsAsubgroup˙def IsAgroup˙def monoid0˙def
    by simp
  then show thesis by (rule monoid0.range˙carr)
qed
```

If we restrict the inverse to a subgroup, then the restricted inverse is onto the subgroup.

```
lemma (in group0) restr˙inv˙onto: assumes A1: IsAsubgroup(H,P)
  shows restrict(GroupInv(G,P),H)(H) = H
proof -
  from A1 have GroupInv(H,restrict(P,H×H))(H) = H
    using IsAsubgroup˙def group0˙def group0.group˙inv˙surj
    by simp
  with A1 show thesis using group0˙3˙T1 by simp
```

qed

A union of two subgroups is a subgroup iff one of the subgroups is a subset of the other subgroup.

```
lemma (in group0) union˙subgroups:
  assumes IsAsubgroup(H1,P) and IsAsubgroup(H2,P)
  shows IsAsubgroup(H1∪H2,P) ⟷ (H1⊆H2 ∨ H2⊆H1)
proof
  assume H1⊆H2 ∨ H2⊆H1 show IsAsubgroup(H1∪H2,P)
  proof -
    from ‹H1⊆H2 ∨ H2⊆H1› have H2 = H1∪H2 ∨ H1 = H1∪H2 by auto
    with assms show IsAsubgroup(H1∪H2,P) by auto
  qed
next
  assume IsAsubgroup(H1∪H2, P) show H1⊆H2 ∨ H2⊆H1
  proof -
    { assume ¬ H1⊆H2
      then obtain x where x∈H1 and x∉H2 by auto
      with assms(1) have x^-1 ∈ H1 using group0˙3˙T3A by simp
      { fix y assume y∈H2
        let z = x·y
        from ‹x∈H1› ‹y∈H2› have x ∈ H1∪H2 and y ∈ H1∪H2 by auto
        with ‹IsAsubgroup(H1∪H2,P)› have z ∈ H1∪H2 using group0˙3˙L6 by blast
        from assms ‹x ∈ H1∪H2› ‹y∈H2› have x∈G y∈G and y^-1∈H2
          using group0˙3˙T3A group0˙3˙L2 by auto
        then have z·y^-1 = x and x^-1·z = y using inv˙cancel˙two(2,3) by auto
        { assume z ∈ H2
          with ‹IsAsubgroup(H2,P)› ‹y^-1∈H2› have z·y^-1 ∈ H2 using group0˙3˙L6 by
simp
          with ‹z·y^-1 = x› ‹x∉H2› have False by auto
        } hence z ∉ H2 by auto
       with assms(1) ‹x^-1 ∈ H1› ‹z ∈ H1∪H2› have x^-1·z ∈ H1 using group0˙3˙L6 by
simp
        with ‹x^-1·z = y› have y∈H1 by simp
      } hence H2⊆H1 by blast
    } thus thesis by blast
  qed
qed
```

Transitivity for "is a subgroup of" relation. The proof (probably) uses the lemma restrict˙restrict from standard Isabelle/ZF library which states that restrict(restrict(f,A),B) = restrict(f,A∩B). That lemma is added to the simplifier, so it does not have to be referenced explicitly in the proof below.

```
lemma subgroup˙transitive:
 assumes IsAgroup(G3,P) IsAsubgroup(G2,P) IsAsubgroup(G1,restrict(P,G2×G2))
  shows IsAsubgroup(G1,P)
proof -
   from assms(2) have group0(G2,restrict(P,G2×G2)) unfolding IsAsubgroup˙def
group0˙def by simp
```

```
  with assms(3) have G1⊆G2 using group0.group0˙3˙L2 by simp
  hence G2×G2 ∩ G1×G1 = G1×G1 by auto
  with assms(3) show IsAsubgroup(G1,P) unfolding IsAsubgroup˙def by simp
qed
```

end

27 Groups 1

theory Group˙ZF˙1 imports Group˙ZF

begin

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

27.1 Translations

In this section we consider translations. Translations are maps $T : G \to G$ of the form $T_g(a) = g \cdot a$ or $T_g(a) = a \cdot g$. We also consider two-dimensional translations $T_g : G \times G \to G \times G$, where $T_g(a, b) = (a \cdot g, b \cdot g)$ or $T_g(a, b) = (g \cdot a, g \cdot b)$.

For an element $a \in G$ the right translation is defined a function (set of pairs) such that its value (the second element of a pair) is the value of the group operation on the first element of the pair and g. This looks a bit strange in the raw set notation, when we write a function explicitely as a set of pairs and value of the group operation on the pair $\langle a, b\rangle$ as P⟨a,b⟩ instead of the usual infix $a \cdot b$ or $a + b$.

```
definition
  RightTranslation(G,P,g) ≡ −⟨ a,b⟩ ∈ G×G. P⟨a,g⟩ = b″
```

A similar definition of the left translation.

```
definition
  LeftTranslation(G,P,g) ≡ −⟨a,b⟩ ∈ G×G. P⟨g,a⟩ = b″
```

Translations map G into G. Two dimensional translations map $G \times G$ into itself.

```
lemma (in group0) group0˙5˙L1: assumes A1: g∈G
  shows RightTranslation(G,P,g) : G→G and  LeftTranslation(G,P,g) : G→G
proof -
  from A1 have ∀a∈G. a·g ∈ G and ∀a∈G. g·a ∈ G
    using group˙oper˙assocA apply˙funtype by auto
  then show
    RightTranslation(G,P,g) : G→G
    LeftTranslation(G,P,g) : G→G
    using RightTranslation˙def LeftTranslation˙def func1˙1˙L11A
```

```
  by auto
qed
```

The values of the translations are what we expect.

```
lemma (in group0) group0˙5˙L2: assumes g∈G a∈G
 shows
 RightTranslation(G,P,g)(a) = a·g
 LeftTranslation(G,P,g)(a) = g·a
 using assms group0˙5˙L1 RightTranslation˙def LeftTranslation˙def
   func1˙1˙L11B by auto
```

Composition of left translations is a left translation by the product.

```
lemma (in group0) group0˙5˙L4: assumes A1: g∈G h∈G a∈G and
 A2: T_g = LeftTranslation(G,P,g)  T_h = LeftTranslation(G,P,h)
 shows
 T_g(T_h(a)) = g·h·a
 T_g(T_h(a)) = LeftTranslation(G,P,g·h)(a)
proof -
 from A1 have I: 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 I show
   T_g(T_h(a)) = LeftTranslation(G,P,g·h)(a)
   using group0˙5˙L2 group˙oper˙assoc by simp
qed
```

Composition of right translations is a right translation by the product.

```
lemma (in group0) group0˙5˙L5: assumes A1: g∈G h∈G a∈G and
 A2: T_g = RightTranslation(G,P,g)  T_h = RightTranslation(G,P,h)
 shows
 T_g(T_h(a)) = a·h·g
 T_g(T_h(a)) = RightTranslation(G,P,h·g)(a)
proof -
 from A1 have I: 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 I show
   T_g(T_h(a)) = RightTranslation(G,P,h·g)(a)
   using group0˙5˙L2 group˙oper˙assoc by simp
qed
```

Point free version of group0˙5˙L4 and group0˙5˙L5.

```
lemma (in group0) trans˙comp: assumes g∈G h∈G shows
 RightTranslation(G,P,g) O RightTranslation(G,P,h) = RightTranslation(G,P,h·g)
 LeftTranslation(G,P,g) O LeftTranslation(G,P,h) = LeftTranslation(G,P,g·h)
proof -
```

```
  let T_g = RightTranslation(G,P,g)
  let T_h = RightTranslation(G,P,h)
  from assms have T_g:G→G and T_h:G→G
    using group0˙5˙L1 by auto
  then have T_g O T_h:G→G using comp˙fun by simp
  moreover from assms have RightTranslation(G,P,h·g):G→G
    using group˙op˙closed group0˙5˙L1 by simp
  moreover from assms ‹T_h:G→G› have
    ∀a∈G. (T_g O T_h)(a) = RightTranslation(G,P,h·g)(a)
    using comp˙fun˙apply group0˙5˙L5 by simp
  ultimately show T_g O T_h =  RightTranslation(G,P,h·g)
    by (rule func˙eq)
next
  let T_g = LeftTranslation(G,P,g)
  let T_h = LeftTranslation(G,P,h)
  from assms have T_g:G→G and T_h:G→G
    using group0˙5˙L1 by auto
  then have T_g O T_h:G→G using comp˙fun by simp
  moreover from assms have LeftTranslation(G,P,g·h):G→G
    using group˙op˙closed group0˙5˙L1 by simp
  moreover from assms ‹T_h:G→G› have
    ∀a∈G. (T_g O T_h)(a) = LeftTranslation(G,P,g·h)(a)
    using comp˙fun˙apply group0˙5˙L4 by simp
  ultimately show T_g O T_h =  LeftTranslation(G,P,g·h)
    by (rule func˙eq)
qed
```

The image of a set under a composition of translations is the same as the image under translation by a product.

```
lemma (in group0) trans˙comp˙image: assumes A1: g∈G h∈G and
  A2: T_g = LeftTranslation(G,P,g)  T_h = LeftTranslation(G,P,h)
shows T_g(T_h(A)) = LeftTranslation(G,P,g·h)(A)
proof -
  from A2 have T_g(T_h(A)) = (T_g O T_h)(A)
    using image˙comp by simp
  with assms show thesis using trans˙comp by simp
qed
```

Another form of the image of a set under a composition of translations

```
lemma (in group0) group0˙5˙L6:
  assumes A1: g∈G h∈G and A2: A⊆G and
  A3: T_g = RightTranslation(G,P,g)  T_h = RightTranslation(G,P,h)
  shows T_g(T_h(A)) = −a·h·g. a∈A″
proof -
  from A2 have ∀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 assms ‹∀a∈A. a∈G› show
    T_g(T_h(A)) = −a·h·g. a∈A″
```

using func1˙1˙L15C group0˙5˙L5 by auto
qed

The translation by neutral element is the identity on group.

lemma (in group0) trans˙neutral: shows
RightTranslation(G,P,**1**) = id(G) and LeftTranslation(G,P,**1**) = id(G)
proof -
have RightTranslation(G,P,**1**):G→G and ∀ a∈G. RightTranslation(G,P,**1**)(a) = a
using group0˙2˙L2 group0˙5˙L1 group0˙5˙L2 by auto
then show RightTranslation(G,P,**1**) = id(G) by (rule indentity˙fun)
have LeftTranslation(G,P,**1**):G→G and ∀ a∈G. LeftTranslation(G,P,**1**)(a) = a
using group0˙2˙L2 group0˙5˙L1 group0˙5˙L2 by auto
then show LeftTranslation(G,P,**1**) = id(G) by (rule indentity˙fun)
qed

Translation by neutral element does not move sets.

lemma (in group0) trans˙neutral˙image: assumes V⊆G
shows RightTranslation(G,P,**1**)(V) = V and LeftTranslation(G,P,**1**)(V) = V
using assms trans˙neutral image˙id˙same by auto

Composition of translations by an element and its inverse is identity.

lemma (in group0) trans˙comp˙id: assumes g∈G shows
RightTranslation(G,P,g) O RightTranslation(G,P,g^{-1}) = id(G) and
RightTranslation(G,P,g^{-1}) O RightTranslation(G,P,g) = id(G) and
LeftTranslation(G,P,g) O LeftTranslation(G,P,g^{-1}) = id(G) and
LeftTranslation(G,P,g^{-1}) O LeftTranslation(G,P,g) = id(G)
using assms inverse˙in˙group trans˙comp group0˙2˙L6 trans˙neutral by auto

Translations are bijective.

lemma (in group0) trans˙bij: assumes g∈G shows
RightTranslation(G,P,g) ∈ bij(G,G) and LeftTranslation(G,P,g) ∈ bij(G,G)
proof-
from assms have
RightTranslation(G,P,g):G→G and
RightTranslation(G,P,g^{-1}):G→G and
RightTranslation(G,P,g) O RightTranslation(G,P,g^{-1}) = id(G)
RightTranslation(G,P,g^{-1}) O RightTranslation(G,P,g) = id(G)
using inverse˙in˙group group0˙5˙L1 trans˙comp˙id by auto
then show RightTranslation(G,P,g) ∈ bij(G,G) using fg˙imp˙bijective by simp
from assms have
LeftTranslation(G,P,g):G→G and
LeftTranslation(G,P,g^{-1}):G→G and
LeftTranslation(G,P,g) O LeftTranslation(G,P,g^{-1}) = id(G)
LeftTranslation(G,P,g^{-1}) O LeftTranslation(G,P,g) = id(G)
using inverse˙in˙group group0˙5˙L1 trans˙comp˙id by auto
then show LeftTranslation(G,P,g) ∈ bij(G,G) using fg˙imp˙bijective by simp
qed

Converse of a translation is translation by the inverse.

```
lemma (in group0) trans˙conv˙inv: assumes g∈G shows
  converse(RightTranslation(G,P,g)) = RightTranslation(G,P,g⁻¹) and
  converse(LeftTranslation(G,P,g)) = LeftTranslation(G,P,g⁻¹) and
  LeftTranslation(G,P,g) = converse(LeftTranslation(G,P,g⁻¹)) and
  RightTranslation(G,P,g) = converse(RightTranslation(G,P,g⁻¹))
proof -
  from assms have
    RightTranslation(G,P,g) ∈ bij(G,G)  RightTranslation(G,P,g⁻¹) ∈ bij(G,G) and
    LeftTranslation(G,P,g) ∈ bij(G,G)  LeftTranslation(G,P,g⁻¹) ∈ bij(G,G)
    using trans˙bij inverse˙in˙group by auto
  moreover from assms have
    RightTranslation(G,P,g⁻¹) O RightTranslation(G,P,g) = id(G) and
    LeftTranslation(G,P,g⁻¹) O LeftTranslation(G,P,g) = id(G) and
    LeftTranslation(G,P,g) O LeftTranslation(G,P,g⁻¹) = id(G) and
    LeftTranslation(G,P,g⁻¹) O LeftTranslation(G,P,g) = id(G)
    using trans˙comp˙id by auto
  ultimately show
    converse(RightTranslation(G,P,g)) = RightTranslation(G,P,g⁻¹) and
    converse(LeftTranslation(G,P,g)) = LeftTranslation(G,P,g⁻¹) and
    LeftTranslation(G,P,g) = converse(LeftTranslation(G,P,g⁻¹)) and
    RightTranslation(G,P,g) = converse(RightTranslation(G,P,g⁻¹))
    using comp˙id˙conv by auto
qed
```

The image of a set by translation is the same as the inverse image by by the inverse element translation.

```
lemma (in group0) trans˙image˙vimage: assumes g∈G shows
  LeftTranslation(G,P,g)(A) = LeftTranslation(G,P,g⁻¹)-(A) and
  RightTranslation(G,P,g)(A) = RightTranslation(G,P,g⁻¹)-(A)
  using assms trans˙conv˙inv vimage˙converse by auto
```

Another way of looking at translations is that they are sections of the group operation.

```
lemma (in group0) trans˙eq˙section: assumes g∈G shows
  RightTranslation(G,P,g) = Fix2ndVar(P,g) and
  LeftTranslation(G,P,g) =  Fix1stVar(P,g)
proof -
  let T = RightTranslation(G,P,g)
  let F = Fix2ndVar(P,g)
  from assms have T: G→G and F: G→G
    using group0˙5˙L1 group˙oper˙assocA fix˙2nd˙var˙fun by auto
  moreover from assms have ∀a∈G. T(a) = F(a)
    using group0˙5˙L2 group˙oper˙assocA fix˙var˙val by simp
  ultimately show T = F by (rule func˙eq)
next
  let T = LeftTranslation(G,P,g)
  let F = Fix1stVar(P,g)
  from assms have T: G→G and F: G→G
    using group0˙5˙L1 group˙oper˙assocA fix˙1st˙var˙fun by auto
```

```
  moreover from assms have ∀a∈G. T(a) = F(a)
    using group0˙5˙L2 group˙oper˙assocA fix˙var˙val by simp
  ultimately show T = F by (rule func˙eq)
qed
```

A lemma demonstrating what is the left translation of a set

```
lemma (in group0) ltrans˙image: assumes A1: V⊆G and A2: x∈G
  shows LeftTranslation(G,P,x)(V) = –x·v. v∈V″
proof -
  from assms have LeftTranslation(G,P,x)(V) = –LeftTranslation(G,P,x)(v). v∈V″
    using group0˙5˙L1 func˙imagedef by blast
  moreover from assms have ∀v∈V. LeftTranslation(G,P,x)(v) = x·v
    using group0˙5˙L2 by auto
  ultimately show thesis by auto
qed
```

A lemma demonstrating what is the right translation of a set

```
lemma (in group0) rtrans˙image: assumes A1: V⊆G and A2: x∈G
  shows RightTranslation(G,P,x)(V) = –v·x. v∈V″
proof -
   from assms have RightTranslation(G,P,x)(V) = –RightTranslation(G,P,x)(v).
v∈V″
    using group0˙5˙L1 func˙imagedef by blast
  moreover from assms have ∀v∈V. RightTranslation(G,P,x)(v) = v·x
    using group0˙5˙L2 by auto
  ultimately show thesis by auto
qed
```

Right and left translations of a set are subsets of the group. Interestingly, we do not have to assume the set is a subset of the group.

```
lemma (in group0) lrtrans˙in˙group: assumes x∈G
  shows LeftTranslation(G,P,x)(V) ⊆ G and RightTranslation(G,P,x)(V) ⊆ G
proof -
 from assms have LeftTranslation(G,P,x):G→G and RightTranslation(G,P,x):G→G
    using group0˙5˙L1 by auto
  then show LeftTranslation(G,P,x)(V) ⊆ G and RightTranslation(G,P,x)(V) ⊆ G
    using func1˙1˙L6(2) by auto
qed
```

A technical lemma about solving equations with translations.

```
lemma (in group0) ltrans˙inv˙in: assumes A1: V⊆G and A2: y∈G and
  A3: x ∈ LeftTranslation(G,P,y)(GroupInv(G,P)(V))
  shows y ∈ LeftTranslation(G,P,x)(V)
proof -
  have x∈G
  proof -
    from A2 have LeftTranslation(G,P,y):G→G using group0˙5˙L1 by simp
    then have LeftTranslation(G,P,y)(GroupInv(G,P)(V)) ⊆ G
```

```
    using func1˙1˙L6 by simp
  with A3 show x∈G by auto
qed
have ∃v∈V. x = y·v⁻¹
proof -
  have GroupInv(G,P): G→G using groupAssum group0˙2˙T2
    by simp
  with assms obtain z where z ∈ GroupInv(G,P)(V) and x = y·z
    using func1˙1˙L6 ltrans˙image by auto
  with A1 ‹GroupInv(G,P): G→G› show thesis using func˙imagedef by auto
qed
then obtain v where v∈V and x = y·v⁻¹ by auto
with A1 A2 have y = x·v using inv˙cancel˙two by auto
with assms ‹x∈G› ‹v∈V› show thesis using ltrans˙image by auto
qed
```

We can look at the result of interval arithmetic operation as union of left translated sets.

```
lemma (in group0) image˙ltrans˙union: assumes A⊆G B⊆G shows
  (P –lifted to subsets of″ G)⟨A,B⟩ = (⋃a∈A. LeftTranslation(G,P,a)(B))
proof
  from assms have I: (P –lifted to subsets of″ G)⟨A,B⟩ = –a·b . ⟨a,b⟩ ∈ A×B″
    using group˙oper˙assocA lift˙subsets˙explained by simp
  { fix c assume c ∈ (P –lifted to subsets of″ G)⟨A,B⟩
    with I obtain a b where c = a·b and a∈A  b∈B by auto
    hence c ∈ –a·b. b∈B″ by auto
    moreover from assms ‹a∈A› have
      LeftTranslation(G,P,a)(B) = –a·b. b∈B″ using ltrans˙image by auto
    ultimately have c ∈ LeftTranslation(G,P,a)(B) by simp
    with ‹a∈A› have c ∈ (⋃a∈A. LeftTranslation(G,P,a)(B)) by auto
  } thus (P –lifted to subsets of″ G)⟨A,B⟩ ⊆ (⋃a∈A. LeftTranslation(G,P,a)(B))
    by auto
  { fix c assume c ∈ (⋃a∈A. LeftTranslation(G,P,a)(B))
    then obtain a where a∈A and c ∈ LeftTranslation(G,P,a)(B)
      by auto
    moreover from assms ‹a∈A› have LeftTranslation(G,P,a)(B) = –a·b. b∈B″
      using ltrans˙image by auto
    ultimately obtain b where b∈B and c = a·b by auto
    with I ‹a∈A› have c ∈ (P –lifted to subsets of″ G)⟨A,B⟩ by auto
  } thus (⋃a∈A. LeftTranslation(G,P,a)(B)) ⊆ (P –lifted to subsets of″ G)⟨A,B⟩
    by auto
qed
```

The right translation version of image˙ltrans˙union The proof follows the same schema.

```
lemma (in group0) image˙rtrans˙union: assumes A⊆G B⊆G shows
  (P –lifted to subsets of″ G)⟨A,B⟩ = (⋃b∈B. RightTranslation(G,P,b)(A))
proof
  from assms have I: (P –lifted to subsets of″ G)⟨A,B⟩ = –a·b . ⟨a,b⟩ ∈ A×B″
```

```
    using group˙oper˙assocA lift˙subsets˙explained by simp
  { fix c assume c ∈ (P –lifted to subsets of″ G)⟨A,B⟩
    with I obtain a b where c = a·b and a∈A  b∈B by auto
    hence c ∈ –a·b. a∈A″ by auto
    moreover from assms ‹b∈B› have
      RightTranslation(G,P,b)(A) = –a·b. a∈A″ using rtrans˙image by auto
    ultimately have c ∈ RightTranslation(G,P,b)(A) by simp
    with ‹b∈B› have c ∈ (⋃b∈B.  RightTranslation(G,P,b)(A)) by auto
  } thus (P –lifted to subsets of″ G)⟨A,B⟩ ⊆ (⋃b∈B.  RightTranslation(G,P,b)(A))
    by auto
  { fix c assume c ∈ (⋃b∈B.  RightTranslation(G,P,b)(A))
    then obtain b where b∈B and c ∈ RightTranslation(G,P,b)(A)
      by auto
    moreover from assms ‹b∈B› have RightTranslation(G,P,b)(A) = –a·b. a∈A″
      using rtrans˙image by auto
    ultimately obtain a where a∈A and c = a·b by auto
    with I ‹b∈B› have c ∈ (P –lifted to subsets of″ G)⟨A,B⟩ by auto
  } thus (⋃b∈B. RightTranslation(G,P,b)(A)) ⊆ (P –lifted to subsets of″ G)⟨A,B⟩
    by auto
qed
```

If the neutral element belongs to a set, then an element of group belongs the translation of that set.

```
lemma (in group0) neut˙trans˙elem:
  assumes A1: A⊆G g∈G and A2: 1∈A
  shows g ∈ LeftTranslation(G,P,g)(A) g ∈ RightTranslation(G,P,g)(A)
proof -
  from assms have g·1 ∈ LeftTranslation(G,P,g)(A)
    using ltrans˙image by auto
  with A1 show g ∈ LeftTranslation(G,P,g)(A) using group0˙2˙L2 by simp
  from assms have 1·g ∈ RightTranslation(G,P,g)(A)
    using rtrans˙image by auto
  with A1 show g ∈ RightTranslation(G,P,g)(A) using group0˙2˙L2 by simp
qed
```

The neutral element belongs to the translation of a set by the inverse of an element that belongs to it.

```
lemma (in group0) elem˙trans˙neut: assumes A1: A⊆G and A2: g∈A
  shows 1 ∈ LeftTranslation(G,P,g⁻¹)(A) 1 ∈ RightTranslation(G,P,g⁻¹)(A)
proof -
  from assms have ginv:g⁻¹ ∈ G using inverse˙in˙group by auto
  with assms have g⁻¹·g ∈ LeftTranslation(G,P,g⁻¹)(A)
    using ltrans˙image by auto
  moreover from assms have g⁻¹·g = 1 using group0˙2˙L6 by auto
  ultimately show 1 ∈ LeftTranslation(G,P,g⁻¹)(A) by simp
  from ginv assms have g·g⁻¹ ∈ RightTranslation(G,P,g⁻¹)(A)
    using rtrans˙image by auto
  moreover from assms have g·g⁻¹ = 1 using group0˙2˙L6 by auto
  ultimately show 1 ∈ RightTranslation(G,P,g⁻¹)(A) by simp
```

qed

27.2 Odd functions

This section is about odd functions.

Odd functions are those that commute with the group inverse: $f(a^{-1}) = (f(a))^{-1}$.

```
definition
  IsOdd(G,P,f) ≡ (∀a∈G. f(GroupInv(G,P)(a)) = GroupInv(G,P)(f(a)) )
```

Let's see the definition of an odd function in a more readable notation.

```
lemma (in group0) group0˙6˙L1:
  shows IsOdd(G,P,p) ⟷ ( ∀a∈G. p(a⁻¹) = (p(a))⁻¹ )
  using IsOdd˙def by simp
```

We can express the definition of an odd function in two ways.

```
lemma (in group0) group0˙6˙L2:
  assumes A1: p : G→G
  shows
  (∀a∈G. p(a⁻¹) = (p(a))⁻¹) ⟷ (∀a∈G. (p(a⁻¹))⁻¹ = p(a))
proof
  assume ∀a∈G. p(a⁻¹) = (p(a))⁻¹
  with A1 show ∀a∈G. (p(a⁻¹))⁻¹ = p(a)
    using apply˙funtype group˙inv˙of˙inv by simp
next assume A2: ∀a∈G. (p(a⁻¹))⁻¹ = p(a)
  { fix a assume a∈G
    with A1 A2  have
      p(a⁻¹) ∈ G and ((p(a⁻¹))⁻¹)⁻¹ =  (p(a))⁻¹
    using apply˙funtype inverse˙in˙group by auto
  then have p(a⁻¹) = (p(a))⁻¹
    using group˙inv˙of˙inv by simp
  } then show ∀a∈G. p(a⁻¹) = (p(a))⁻¹ by simp
qed
```

27.3 Subgroups and interval arithmetic

The section Binary operations in the func˙ZF theory defines the notion of "lifting operation to subsets". In short, every binary operation $f : X \times X \longrightarrow X$ on a set X defines an operation on the subsets of X defined by $F(A, B) = \{f\langle x, y\rangle | x \in A, y \in B\}$. In the group context using multiplicative notation we can write this as $H \cdot K = \{x \cdot y | x \in A, y \in B\}$. Similarly we can define $H^{-1} = \{x^{-1} | x \in H\}$. In this section we study properties of these derived operation and how they relate to the concept of subgroups.

The next locale extends the groups0 locale with notation related to interval arithmetics.

```
locale group4 = group0 +
  fixes sdot (infixl · 70)
  defines sdot˙def [simp]: A·B  ≡ (P –lifted to subsets of″ G)⟨A,B⟩

  fixes sinv (˙⁻¹  [90] 91)
  defines sinv˙def[simp]: A⁻¹ ≡ GroupInv(G,P)(A)
```

The next lemma shows a somewhat more explicit way of defining the product of two subsets of a group.

```
lemma (in group4) interval˙prod: assumes A⊆G B⊆G
  shows A·B =  –x·y. ⟨x,y⟩ ∈ A×B″
  using assms group˙oper˙assocA lift˙subsets˙explained by auto
```

Product of elements of subsets of the group is in the set product of those subsets

```
lemma (in group4) interval˙prod˙el: assumes A⊆G B⊆G x∈A y∈B
  shows x·y ∈ A·B
  using assms interval˙prod by auto
```

An alternative definition of a group inverse of a set.

```
lemma (in group4) interval˙inv: assumes A⊆G
  shows A⁻¹ = –x⁻¹.x∈A″
proof -
  from groupAssum have GroupInv(G,P):G→G using group0˙2˙T2 by simp
  with assms show A⁻¹ = –x⁻¹.x∈A″ using func˙imagedef by simp
qed
```

Group inverse of a set is a subset of the group. Interestingly we don't need to assume the set is a subset of the group.

```
lemma (in group4) interval˙inv˙cl: shows A⁻¹ ⊆ G
proof -
  from groupAssum have GroupInv(G,P):G→G using group0˙2˙T2 by simp
  then show A⁻¹ ⊆ G using func1˙1˙L6(2) by simp
qed
```

The product of two subsets of a group is a subset of the group.

```
lemma (in group4) interval˙prod˙closed: assumes A⊆G B⊆G
  shows A·B ⊆ G
proof
  fix z assume z ∈ A·B
  with assms obtain x y where x∈A y∈B z=x·y using interval˙prod by auto
  with assms show z∈G using group˙op˙closed by auto
qed
```

The product of sets operation is associative.

```
lemma (in group4) interval˙prod˙assoc: assumes A⊆G B⊆G C⊆G
  shows A·B·C = A·(B·C)
```

```
proof -
  from groupAssum have (P –lifted to subsets of″ G) –is associative on″ Pow(G)
    unfolding IsAgroup˙def IsAmonoid˙def using lift˙subset˙assoc by simp
  with assms show thesis unfolding IsAssociative˙def by auto
qed
```

A simple rearrangement following from associativity of the product of sets operation.

```
lemma (in group4) interval˙prod˙rearr1: assumes A⊆G B⊆G C⊆G D⊆G
  shows A·B·(C·D) = A·(B·C)·D
proof -
  from assms(1,2) have A·B ⊆ G using interval˙prod˙closed by simp
  with assms(3,4) have A·B·(C·D) = A·B·C·D
    using interval˙prod˙assoc by simp
  also from assms(1,2,3) have A·B·C·D = A·(B·C)·D
    using interval˙prod˙assoc by simp
  finally show thesis by simp
qed
```

A subset A of the group is closed with respect to the group operation iff $A \cdot A \subseteq A$.

```
lemma (in group4) subset˙gr˙op˙cl: assumes A⊆G
  shows (A –is closed under″ P) ⟷ A·A ⊆ A
proof
  assume A –is closed under″ P
  { fix z assume z ∈ A·A
    with assms obtain x y where x∈A y∈A and z=x·y using interval˙prod by auto
    with ‹A –is closed under″ P› have z∈A unfolding IsOpClosed˙def by simp
  } thus A·A ⊆ A by auto
next
  assume A·A ⊆ A
  { fix x y assume x∈A y∈A
    with assms have x·y ∈ A·A using interval˙prod by auto
    with ‹A·A ⊆ A› have x·y ∈ A by auto
  } then show A –is closed under″ P unfolding IsOpClosed˙def by simp
qed
```

Inverse and square of a subgroup is this subgroup.

```
lemma (in group4) subgroup˙inv˙sq: assumes IsAsubgroup(H,P)
   shows H⁻¹ = H and H·H = H
proof
  from assms have H⊆G using group0˙3˙L2 by simp
  with assms show H⁻¹ ⊆ H using interval˙inv group0˙3˙T3A by auto
  { fix x assume x∈H
    with assms have (x⁻¹)⁻¹ ∈ –y⁻¹.y∈H″ using group0˙3˙T3A by auto
    moreover from ‹x∈H› ‹H⊆G› have (x⁻¹)⁻¹ = x using group˙inv˙of˙inv by auto
    ultimately have x ∈ –y⁻¹.y∈H″ by auto
    with ‹H⊆G› have x ∈ H⁻¹ using interval˙inv by simp
```

} thus $H \subseteq H^{-1}$ by auto
from assms have H –is closed under″ P using group0˙3˙L6 unfolding IsOpClosed˙def by simp
with assms have $H \cdot H \subseteq H$ using subset˙gr˙op˙cl group0˙3˙L2 by simp
moreover
{ fix x assume $x \in H$
with assms have $x \in G$ using group0˙3˙L2 by auto
from assms ‹$H \subseteq G$› ‹$x \in H$› have $x \cdot \mathbf{1} \in H \cdot H$ using group0˙3˙L5 interval˙prod by auto
with ‹$x \in G$› have $x \in H \cdot H$ using group0˙2˙L2 by simp
} hence $H \subseteq H \cdot H$ by auto
ultimately show $H \cdot H = H$ by auto
qed

Inverse of a product two sets is a product of inverses with the reversed order.

lemma (in group4) interval˙prod˙inv: assumes $A \subseteq G$ $B \subseteq G$
shows
$(A \cdot B)^{-1} = -(x \cdot y)^{-1}.\langle x,y \rangle \in A \times B''$
$(A \cdot B)^{-1} = -y^{-1} \cdot x^{-1}.\langle x,y \rangle \in A \times B''$
$(A \cdot B)^{-1} = (B^{-1}) \cdot (A^{-1})$
proof -
from assms have $(A \cdot B) \subseteq G$ using interval˙prod˙closed by simp
then have I: $(A \cdot B)^{-1} = -z^{-1}.z \in A \cdot B''$ using interval˙inv by simp
show II: $(A \cdot B)^{-1} = -(x \cdot y)^{-1}.\langle x,y \rangle \in A \times B''$
proof
{ fix p assume $p \in (A \cdot B)^{-1}$
with I obtain z where $p = z^{-1}$ and $z \in A \cdot B$ by auto
with assms obtain x y where $\langle x,y \rangle \in A \times B$ and $z = x \cdot y$ using interval˙prod by auto
with ‹$p = z^{-1}$› have $p \in -(x \cdot y)^{-1}.\langle x,y \rangle \in A \times B''$ by auto
} thus $(A \cdot B)^{-1} \subseteq -(x \cdot y)^{-1}.\langle x,y \rangle \in A \times B''$ by blast
{ fix p assume $p \in -(x \cdot y)^{-1}.\langle x,y \rangle \in A \times B''$
then obtain x y where $x \in A$ $y \in B$ and $p = (x \cdot y)^{-1}$ by auto
with assms ‹$(A \cdot B) \subseteq G$› have $p \in (A \cdot B)^{-1}$ using interval˙prod interval˙inv by auto
} thus $-(x \cdot y)^{-1}.\langle x,y \rangle \in A \times B'' \subseteq (A \cdot B)^{-1}$ by blast
qed
have $-(x \cdot y)^{-1}.\langle x,y \rangle \in A \times B'' = -y^{-1} \cdot x^{-1}.\langle x,y \rangle \in A \times B''$
proof
{ fix p assume $p \in -(x \cdot y)^{-1}.\langle x,y \rangle \in A \times B''$
then obtain x y where $x \in A$ $y \in B$ and $p = (x \cdot y)^{-1}$ by auto
with assms have $y^{-1} \cdot x^{-1} = (x \cdot y)^{-1}$ using group˙inv˙of˙two by auto
with ‹$p = (x \cdot y)^{-1}$› have $p = y^{-1} \cdot x^{-1}$ by simp
with ‹$x \in A$› ‹$y \in B$› have $p \in -y^{-1} \cdot x^{-1}.\langle x,y \rangle \in A \times B''$ by auto
} thus $-(x \cdot y)^{-1}.\langle x,y \rangle \in A \times B'' \subseteq -y^{-1} \cdot x^{-1}.\langle x,y \rangle \in A \times B''$ by blast
{ fix p assume $p \in -y^{-1} \cdot x^{-1}.\langle x,y \rangle \in A \times B''$
then obtain x y where $x \in A$ $y \in B$ and $p = y^{-1} \cdot x^{-1}$ by auto
with assms have $p = (x \cdot y)^{-1}$ using group˙inv˙of˙two by auto
with ‹$x \in A$› ‹$y \in B$› have $p \in -(x \cdot y)^{-1}.\langle x,y \rangle \in A \times B''$ by auto

} thus $-y^{-1}\cdot x^{-1}.\langle x,y\rangle \in A\times B'' \subseteq -(x\cdot y)^{-1}.\langle x,y\rangle \in A\times B''$ by blast
qed
with II show III: $(A\cdot B)^{-1} = -y^{-1}\cdot x^{-1}.\langle x,y\rangle \in A\times B''$ by simp
have $-y^{-1}\cdot x^{-1}.\langle x,y\rangle \in A\times B'' = (B^{-1})\cdot(A^{-1})$
proof
{ fix p assume $p\in -y^{-1}\cdot x^{-1}.\langle x,y\rangle \in A\times B''$
then obtain x y where $x\in A$ $y\in B$ and $p=y^{-1}\cdot x^{-1}$ by auto
with assms have $y^{-1} \in (B^{-1})$ and $x^{-1} \in (A^{-1})$
using interval˙inv by auto
with ‹$p=y^{-1}\cdot x^{-1}$› have $p \in (B^{-1})\cdot(A^{-1})$ using interval˙inv˙cl interval˙prod
by auto
} thus $-y^{-1}\cdot x^{-1}.\langle x,y\rangle \in A\times B'' \subseteq (B^{-1})\cdot(A^{-1})$ by blast
{ fix p assume $p \in (B^{-1})\cdot(A^{-1})$
then obtain y x where $y\in B^{-1}$ $x\in A^{-1}$ and $p=y\cdot x$
using interval˙inv˙cl interval˙prod by auto
with assms obtain x_1 y_1 where $x_1 \in A$ $y_1 \in B$ and $x={x_1}^{-1}$ $y={y_1}^{-1}$ using interval˙inv
by auto
with ‹$p=y\cdot x$› have $p \in -y^{-1}\cdot x^{-1}.\langle x,y\rangle \in A\times B''$ by auto
} thus $(B^{-1})\cdot(A^{-1}) \subseteq -y^{-1}\cdot x^{-1}.\langle x,y\rangle \in A\times B''$ by blast
qed
with III show $(A\cdot B)^{-1} = (B^{-1})\cdot(A^{-1})$ by simp
qed

If H, K are subgroups then $H \cdot K$ is a subgroup iff $H \cdot K = K \cdot H$.

theorem (in group4) prod˙subgr˙subgr:
assumes IsAsubgroup(H,P) and IsAsubgroup(K,P)
shows IsAsubgroup(H·K,P) $\longleftrightarrow$ H·K = K·H
proof
assume IsAsubgroup(H·K,P)
then have $(H\cdot K)^{-1} = H\cdot K$ using subgroup˙inv˙sq(1) by simp
with assms show H·K = K·H using group0˙3˙L2 interval˙prod˙inv subgroup˙inv˙sq(1)
by auto
next
from assms have $H\subseteq G$ and $K\subseteq G$ using group0˙3˙L2 by auto
have I: $H\cdot K \neq 0$
proof -
let x = **1** let y = **1**
from assms have $x\cdot y \in (H\cdot K)$ using group0˙3˙L5 group0˙3˙L2 interval˙prod
by auto
thus thesis by auto
qed
from ‹$H\subseteq G$› ‹$K\subseteq G$› have II: $H\cdot K \subseteq G$ using interval˙prod˙closed by simp
assume H·K = K·H
have III: (H·K)–is closed under″ P
proof -
have $(H\cdot K)\cdot(H\cdot K) = H\cdot K$
proof -
from ‹$H\subseteq G$› ‹$K\subseteq G$› have $(H\cdot K)\cdot(H\cdot K) = H\cdot(K\cdot H)\cdot K$

```
      using interval˙prod˙rearr1 by simp
    also from ‹H·K = K·H› have ... = H·(H·K)·K by simp
    also from ‹H⊆G› ‹K⊆G› have ... = (H·H)·(K·K)
      using interval˙prod˙rearr1 by simp
    also from assms have ... = H·K using subgroup˙inv˙sq(2) by simp
    finally show thesis by simp
  qed
  with ‹H·K ⊆ G› show thesis using subset˙gr˙op˙cl by simp
qed
have IV: ∀x ∈ H·K. x⁻¹ ∈ H·K
proof -
  { fix x assume x ∈ H·K
    with ‹H·K ⊆ G› have x⁻¹ ∈ (H·K)⁻¹ using interval˙inv by auto
    with assms ‹H⊆G› ‹K⊆G› ‹H·K = K·H› have x⁻¹ ∈ H·K
      using interval˙prod˙inv subgroup˙inv˙sq(1) by simp
  } thus thesis by auto
qed
from I II III IV show IsAsubgroup(H·K,P) using group0˙3˙T3 by simp
qed

end
```

28 Groups - and alternative definition

theory Group˙ZF˙1b imports Group˙ZF

begin

In a typical textbook a group is defined as a set G with an associative operation such that two conditions hold:

A: there is an element $e \in G$ such that for all $g \in G$ we have $e \cdot g = g$ and $g \cdot e = g$. We call this element a "unit" or a "neutral element" of the group.

B: for every $a \in G$ there exists a $b \in G$ such that $a \cdot b = e$, where e is the element of G whose existence is guaranteed by A.

The validity of this definition is rather dubious to me, as condition A does not define any specific element e that can be referred to in condition B - it merely states that a set of such units e is not empty. Of course it does work in the end as we can prove that the set of such neutral elements has exactly one element, but still the definition by itself is not valid. You just can't reference a variable bound by a quantifier outside of the scope of that quantifier.

One way around this is to first use condition A to define the notion of a monoid, then prove the uniqueness of e and then use the condition B to define groups.

Another way is to write conditions A and B together as follows:

$\exists_{e \in G}\ (\forall_{g \in G}\ e \cdot g = g \wedge g \cdot e = g) \wedge (\forall_{a \in G} \exists_{b \in G}\ a \cdot b = e)$.

This is rather ugly.

What I want to talk about is an amusing way to define groups directly without any reference to the neutral elements. Namely, we can define a group as a non-empty set G with an associative operation "$\cdot$" such that

C: for every $a, b \in G$ the equations $a \cdot x = b$ and $y \cdot a = b$ can be solved in G.

This theory file aims at proving the equivalence of this alternative definition with the usual definition of the group, as formulated in Group˙ZF.thy. The informal proofs come from an Aug. 14, 2005 post by buli on the matematyka.org forum.

28.1 An alternative definition of group

First we will define notation for writing about groups.

We will use the multiplicative notation for the group operation. To do this, we define a context (locale) that tells Isabelle to interpret $a \cdot b$ as the value of function P on the pair $\langle a, b\rangle$.

```
locale group2 =
  fixes P
  fixes dot (infixl · 70)
  defines dot˙def [simp]: a · b ≡ P⟨a,b⟩
```

The next theorem states that a set G with an associative operation that satisfies condition C is a group, as defined in IsarMathLib Group˙ZF theory.

```
theorem (in group2) altgroup˙is˙group:
  assumes A1: G≠0 and A2: P –is associative on″ G
  and A3: ∀a∈G.∀b∈G. ∃x∈G. a·x = b
  and A4: ∀a∈G.∀b∈G. ∃y∈G. y·a = b
  shows IsAgroup(G,P)
proof -
  from A1 obtain a where a∈G by auto
  with A3 obtain x where x∈G and a·x = a
    by auto
  from A4 ‹a∈G› obtain y where y∈G and y·a = a
    by auto
  have I: ∀b∈G. b = b·x ∧ b = y·b
  proof
    fix b assume b∈G
     with A4 ‹a∈G› obtain y_b where y_b∈G
      and y_b·a = b by auto
    from A3 ‹a∈G› ‹b∈G› obtain x_b where x_b∈G
      and a·x_b = b by auto
    from ‹a·x = a› ‹y·a = a› ‹y_b·a = b› ‹a·x_b = b›
    have b = y_b·(a·x) and b = (y·a)·x_b
      by auto
    moreover from A2 ‹a∈G› ‹x∈G› ‹y∈G› ‹x_b∈G› ‹y_b∈G› have
```

```
      (y·a)·x_b = y·(a·x_b)  y_b·(a·x) = (y_b·a)·x
      using IsAssociative˙def by auto
    moreover from ‹y_b·a = b› ‹a·x_b = b› have
      (y_b·a)·x = b·x  y·(a·x_b) = y·b
      by auto
    ultimately show b = b·x ∧ b = y·b by simp
  qed
  moreover have x = y
  proof -
    from ‹x∈G› I have x = y·x by simp
    also from ‹y∈G› I have y·x = y by simp
    finally show x = y by simp
  qed
  ultimately have ∀b∈G. b·x = b ∧ x·b = b by simp
  with A2 ‹x∈G› have IsAmonoid(G,P) using IsAmonoid˙def by auto
  with A3 show IsAgroup(G,P)
    using monoid0˙def monoid0.unit˙is˙neutral IsAgroup˙def
    by simp
qed
```

The converse of altgroup˙is˙group: in every (classically defined) group condition C holds. In informal mathematics we can say "Obviously condition C holds in any group." In formalized mathematics the word "obviously" is not in the language. The next theorem is proven in the context called group0 defined in the theory Group˙ZF.thy. Similarly to the group2 that context defines $a \cdot b$ as $P\langle a, b\rangle$ It also defines notation related to the group inverse and adds an assumption that the pair (G, P) is a group to all its theorems. This is why in the next theorem we don't explicitely assume that (G, P) is a group - this assumption is implicit in the context.

```
theorem (in group0) group˙is˙altgroup: shows
  ∀a∈G.∀b∈G. ∃x∈G. a·x = b and ∀a∈G.∀b∈G. ∃y∈G. y·a = b
proof -
  { fix a b assume a∈G  b∈G
    let x = a^-1· b
    let y = b·a^-1
    from ‹a∈G›  ‹b∈G›  have
      x ∈ G  y ∈ G  and  a·x = b  y·a = b
      using inverse˙in˙group group˙op˙closed inv˙cancel˙two
      by auto
    hence ∃x∈G. a·x = b and ∃y∈G. y·a = b by auto
  } thus
      ∀a∈G.∀b∈G. ∃x∈G. a·x = b and
      ∀a∈G.∀b∈G. ∃y∈G. y·a = b
    by auto
qed

end
```

29 Abelian Group

theory AbelianGroup˙ZF imports Group˙ZF

begin

A group is called "abelian" if its operation is commutative, i.e. $P\langle a, b\rangle = P\langle a, b\rangle$ for all group elements a, b, where P is the group operation. It is customary to use the additive notation for abelian groups, so this condition is typically written as $a+b = b+a$. We will be using multiplicative notation though (in which the commutativity condition of the operation is written as $a \cdot b = b \cdot a$), just to avoid the hassle of changing the notation we used for general groups.

29.1 Rearrangement formulae

This section is not interesting and should not be read. Here we will prove formulas is which right hand side uses the same factors as the left hand side, just in different order. These facts are obvious in informal math sense, but Isabelle prover is not able to derive them automatically, so we have to prove them by hand.

Proving the facts about associative and commutative operations is quite tedious in formalized mathematics. To a human the thing is simple: we can arrange the elements in any order and put parantheses wherever we want, it is all the same. However, formalizing this statement would be rather difficult (I think). The next lemma attempts a quasi-algorithmic approach to this type of problem. To prove that two expressions are equal, we first strip one from parantheses, then rearrange the elements in proper order, then put the parantheses where we want them to be. The algorithm for rearrangement is easy to describe: we keep putting the first element (from the right) that is in the wrong place at the left-most position until we get the proper arrangement. As far removing parantheses is concerned Isabelle does its job automatically.

```
lemma (in group0) group0˙4˙L2:
  assumes A1:P –is commutative on″ G
  and A2:a∈G b∈G c∈G d∈G E∈G F∈G
  shows (a·b)·(c·d)·(E·F) = (a·(d·F))·(b·(c·E))
proof -
  from A2 have (a·b)·(c·d)·(E·F) = a·b·c·d·E·F
    using group˙op˙closed group˙oper˙assoc
    by simp
  also have  a·b·c·d·E·F = a·d·F·b·c·E
  proof -
    from A1 A2 have a·b·c·d·E·F = F·(a·b·c·d·E)
      using IsCommutative˙def group˙op˙closed
```

```
      by simp
    also from A2 have F·(a·b·c·d·E) = F·a·b·c·d·E
      using group˙op˙closed group˙oper˙assoc
      by simp
    also from A1 A2 have F·a·b·c·d·E = d·(F·a·b·c)·E
      using IsCommutative˙def group˙op˙closed
      by simp
    also from A2 have d·(F·a·b·c)·E = d·F·a·b·c·E
      using group˙op˙closed group˙oper˙assoc
      by simp
    also from A1 A2 have  d·F·a·b·c·E = a·(d·F)·b·c·E
      using IsCommutative˙def group˙op˙closed
      by simp
    also from A2 have a·(d·F)·b·c·E = a·d·F·b·c·E
      using group˙op˙closed group˙oper˙assoc
      by simp
    finally show thesis by simp
  qed
  also from A2 have a·d·F·b·c·E = (a·(d·F))·(b·(c·E))
    using group˙op˙closed group˙oper˙assoc
    by simp
  finally show thesis by simp
qed
```

Another useful rearrangement.

```
lemma (in group0) group0˙4˙L3:
  assumes A1:P –is commutative on" G
  and A2: a∈G  b∈G and A3: c∈G  d∈G  E∈G  F∈G
  shows a·b·((c·d)^-1·(E·F)^-1) = (a·(E·c)^-1)·(b·(F·d)^-1)
proof -
  from A3 have T1:
    c^-1∈G d^-1∈G E^-1∈G F^-1∈G (c·d)^-1∈G (E·F)^-1∈G
    using inverse˙in˙group group˙op˙closed
    by auto
  from A2 T1 have
    a·b·((c·d)^-1·(E·F)^-1) = a·b·(c·d)^-1·(E·F)^-1
    using group˙op˙closed group˙oper˙assoc
    by simp
  also from A2 A3 have
    a·b·(c·d)^-1·(E·F)^-1 = (a·b)·(d^-1·c^-1)·(F^-1·E^-1)
    using group˙inv˙of˙two by simp
   also from A1 A2 T1 have
    (a·b)·(d^-1·c^-1)·(F^-1·E^-1) = (a·(c^-1·E^-1))·(b·(d^-1·F^-1))
    using group0˙4˙L2 by simp
  also from A2 A3 have
    (a·(c^-1·E^-1))·(b·(d^-1·F^-1)) = (a·(E·c)^-1)·(b·(F·d)^-1)
    using group˙inv˙of˙two by simp
  finally show thesis by simp
qed
```

Some useful rearrangements for two elements of a group.

```
lemma (in group0) group0˙4˙L4:
  assumes A1:P –is commutative on″ G
  and A2: a∈G b∈G
  shows
  b⁻¹·a⁻¹ = a⁻¹·b⁻¹
  (a·b)⁻¹ = a⁻¹·b⁻¹
  (a·b⁻¹)⁻¹ = a⁻¹·b
proof -
  from A2 have T1: b⁻¹∈G a⁻¹∈G using inverse˙in˙group by auto
  with A1 show b⁻¹·a⁻¹ = a⁻¹·b⁻¹ using IsCommutative˙def by simp
  with A2 show (a·b)⁻¹ = a⁻¹·b⁻¹ using group˙inv˙of˙two by simp
  from A2 T1 have (a·b⁻¹)⁻¹ = (b⁻¹)⁻¹·a⁻¹ using group˙inv˙of˙two by simp
  with A1 A2 T1 show (a·b⁻¹)⁻¹ = a⁻¹·b
    using group˙inv˙of˙inv IsCommutative˙def by simp
qed
```

Another bunch of useful rearrangements with three elements.

```
lemma (in group0) group0˙4˙L4A:
  assumes A1: P –is commutative on″ G
  and A2: a∈G  b∈G  c∈G
  shows
  a·b·c = c·a·b
  a⁻¹·(b⁻¹·c⁻¹)⁻¹ = (a·(b·c)⁻¹)⁻¹
  a·(b·c)⁻¹ = a·b⁻¹·c⁻¹
  a·(b·c⁻¹)⁻¹ = a·b⁻¹·c
  a·b⁻¹·c⁻¹ = a·c⁻¹·b⁻¹
proof -
  from A1 A2 have a·b·c = c·(a·b)
    using IsCommutative˙def group˙op˙closed
    by simp
  with A2 show a·b·c = c·a·b using
     group˙op˙closed group˙oper˙assoc
    by simp
  from A2 have T:
    b⁻¹∈G  c⁻¹∈G  b⁻¹·c⁻¹ ∈ G  a·b ∈ G
    using inverse˙in˙group group˙op˙closed
    by auto
  with A1 A2 show a⁻¹·(b⁻¹·c⁻¹)⁻¹ = (a·(b·c)⁻¹)⁻¹
    using group˙inv˙of˙two IsCommutative˙def
    by simp
  from A1 A2 T have a·(b·c)⁻¹ = a·(b⁻¹·c⁻¹)
    using group˙inv˙of˙two IsCommutative˙def by simp
  with A2 T show a·(b·c)⁻¹ = a·b⁻¹·c⁻¹
    using group˙oper˙assoc by simp
  from A1 A2 T have a·(b·c⁻¹)⁻¹ = a·(b⁻¹·(c⁻¹)⁻¹)
    using group˙inv˙of˙two IsCommutative˙def by simp
  with A2 T show a·(b·c⁻¹)⁻¹ = a·b⁻¹·c
    using group˙oper˙assoc group˙inv˙of˙inv by simp
```

```
  from A1 A2 T have a·b⁻¹·c⁻¹ = a·(c⁻¹·b⁻¹)
    using group˙oper˙assoc IsCommutative˙def by simp
  with A2 T show a·b⁻¹·c⁻¹ = a·c⁻¹·b⁻¹
    using group˙oper˙assoc by simp
qed
```

Another useful rearrangement.

```
lemma (in group0) group0˙4˙L4B:
  assumes P –is commutative on″ G
  and a∈G  b∈G  c∈G
  shows a·b⁻¹·(b·c⁻¹) = a·c⁻¹
  using assms inverse˙in˙group group˙op˙closed
    group0˙4˙L4 group˙oper˙assoc inv˙cancel˙two by simp
```

A couple of permutations of order for three alements.

```
lemma (in group0) group0˙4˙L4C:
  assumes A1: P –is commutative on″ G
  and A2: a∈G b∈G c∈G
  shows
  a·b·c = c·a·b
  a·b·c = a·(c·b)
  a·b·c = c·(a·b)
  a·b·c = c·b·a
proof -
  from A1 A2 show I: a·b·c = c·a·b
    using group0˙4˙L4A by simp
  also from A1 A2 have c·a·b = a·c·b
    using IsCommutative˙def by simp
  also from A2 have a·c·b = a·(c·b)
    using group˙oper˙assoc by simp
  finally show a·b·c = a·(c·b) by simp
  from A2 I show a·b·c = c·(a·b)
    using group˙oper˙assoc by simp
  also from A1 A2 have c·(a·b) = c·(b·a)
    using IsCommutative˙def by simp
  also from A2 have c·(b·a) = c·b·a
    using group˙oper˙assoc by simp
  finally show a·b·c = c·b·a by simp
qed
```

Some rearangement with three elements and inverse.

```
lemma (in group0) group0˙4˙L4D:
  assumes A1: P –is commutative on″ G
  and A2: a∈G  b∈G  c∈G
  shows
  a⁻¹·b⁻¹·c = c·a⁻¹·b⁻¹
  b⁻¹·a⁻¹·c = c·a⁻¹·b⁻¹
  (a⁻¹·b·c)⁻¹ = a·b⁻¹·c⁻¹
proof -
```

```
  from A2 have T:
    a⁻¹ ∈ G  b⁻¹ ∈ G  c⁻¹∈G
    using inverse˙in˙group by auto
  with A1 A2 show
    a⁻¹·b⁻¹·c = c·a⁻¹·b⁻¹
    b⁻¹·a⁻¹·c = c·a⁻¹·b⁻¹
    using  group0˙4˙L4A by auto
  from A1 A2 T show (a⁻¹·b·c)⁻¹ = a·b⁻¹·c⁻¹
    using group˙inv˙of˙three group˙inv˙of˙inv group0˙4˙L4C
    by simp
qed
```

Another rearrangement lemma with three elements and equation.

```
lemma (in group0) group0˙4˙L5: assumes A1:P –is commutative on″ G
  and A2: a∈G  b∈G  c∈G
  and A3: c = a·b⁻¹
  shows a = b·c
proof -
  from A2 A3 have c·(b⁻¹)⁻¹ = a
    using inverse˙in˙group group0˙2˙L18
    by simp
  with A1 A2 show thesis using
     group˙inv˙of˙inv IsCommutative˙def by simp
qed
```

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

```
lemma (in group0) group0˙4˙L6A: assumes A1: P –is commutative on″ G
  and A2: a∈G  b∈G
  shows
  a·b·a⁻¹ = b
  a⁻¹·b·a = b
  a⁻¹·(b·a) = b
  a·(b·a⁻¹) = b
proof -
  from A1 A2 have
    a·b·a⁻¹ = a⁻¹·a·b
    using inverse˙in˙group group0˙4˙L4A by blast
  also from A2 have ... = b
    using group0˙2˙L6 group0˙2˙L2 by simp
  finally show a·b·a⁻¹ = b by simp
  from A1 A2 have
    a⁻¹·b·a = a·a⁻¹·b
    using inverse˙in˙group group0˙4˙L4A by blast
  also from A2 have ... = b
    using group0˙2˙L6 group0˙2˙L2 by simp
  finally show a⁻¹·b·a = b by simp
  moreover from A2 have a⁻¹·b·a = a⁻¹·(b·a)
    using inverse˙in˙group group˙oper˙assoc by simp
```

ultimately show $a^{-1}\cdot(b\cdot a) = b$ by simp
from A1 A2 show $a\cdot(b\cdot a^{-1}) = b$
using inverse˙in˙group IsCommutative˙def inv˙cancel˙two
by simp
qed

Another lemma about cancelling with two elements.

lemma (in group0) group0˙4˙L6AA:
assumes A1: P –is commutative on″ G and A2: a∈G b∈G
shows $a\cdot b^{-1}\cdot a^{-1} = b^{-1}$
using assms inverse˙in˙group group0˙4˙L6A
by auto

Another lemma about cancelling with two elements.

lemma (in group0) group0˙4˙L6AB:
assumes A1: P –is commutative on″ G and A2: a∈G b∈G
shows
$a\cdot(a\cdot b)^{-1} = b^{-1}$
$a\cdot(b\cdot a^{-1}) = b$
proof -
from A2 have $a\cdot(a\cdot b)^{-1} = a\cdot(b^{-1}\cdot a^{-1})$
using group˙inv˙of˙two by simp
also from A2 have $\ldots = a\cdot b^{-1}\cdot a^{-1}$
using inverse˙in˙group group˙oper˙assoc by simp
also from A1 A2 have $\ldots = b^{-1}$
using group0˙4˙L6AA by simp
finally show $a\cdot(a\cdot b)^{-1} = b^{-1}$ by simp
from A1 A2 have $a\cdot(b\cdot a^{-1}) = a\cdot(a^{-1}\cdot b)$
using inverse˙in˙group IsCommutative˙def by simp
also from A2 have $\ldots = b$
using inverse˙in˙group group˙oper˙assoc group0˙2˙L6 group0˙2˙L2
by simp
finally show $a\cdot(b\cdot a^{-1}) = b$ by simp
qed

Another lemma about cancelling with two elements.

lemma (in group0) group0˙4˙L6AC:
assumes P –is commutative on″ G and a∈G b∈G
shows $a\cdot(a\cdot b^{-1})^{-1} = b$
using assms inverse˙in˙group group0˙4˙L6AB group˙inv˙of˙inv
by simp

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

lemma (in group0) group0˙4˙L6B: assumes A1: P –is commutative on″ G
and A2: a∈G b∈G c∈G
shows
$a\cdot b\cdot c\cdot a^{-1} = b\cdot c$

```
  a⁻¹·b·c·a = b·c
proof -
   from A2 have
     a·b·c·a⁻¹ = a·(b·c)·a⁻¹
     a⁻¹·b·c·a = a⁻¹·(b·c)·a
    using group˙op˙closed group˙oper˙assoc inverse˙in˙group
    by auto
  with A1 A2 show
    a·b·c·a⁻¹ = b·c
    a⁻¹·b·c·a = b·c
    using group˙op˙closed group0˙4˙L6A
    by auto
qed
```

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

```
lemma (in group0) group0˙4˙L6C: assumes A1: P –is commutative on″ G
  and A2: a∈G b∈G c∈G d∈G
  shows a·b·c·d·a⁻¹ = b·c·d
proof -
  from A2 have a·b·c·d·a⁻¹ = a·(b·c·d)·a⁻¹
    using group˙op˙closed group˙oper˙assoc
    by simp
  with A1 A2 show thesis
    using group˙op˙closed group0˙4˙L6A
    by simp
qed
```

Another couple of useful rearrangements of three elements and cancelling.

```
lemma (in group0) group0˙4˙L6D:
  assumes A1: P –is commutative on″ G
  and A2: a∈G  b∈G  c∈G
  shows
  a·b⁻¹·(a·c⁻¹)⁻¹ = c·b⁻¹
  (a·c)⁻¹·(b·c) = a⁻¹·b
  a·(b·(c·a⁻¹·b⁻¹)) = c
  a·b·c⁻¹·(c·a⁻¹) = b
proof -
  from A2 have T:
    a⁻¹ ∈ G  b⁻¹ ∈ G  c⁻¹ ∈ G
    a·b ∈ G  a·b⁻¹ ∈ G  c⁻¹·a⁻¹ ∈ G  c·a⁻¹ ∈ G
    using inverse˙in˙group group˙op˙closed by auto
  with A1 A2 show a·b⁻¹·(a·c⁻¹)⁻¹ = c·b⁻¹
    using group0˙2˙L12 group˙oper˙assoc group0˙4˙L6B
    IsCommutative˙def by simp
  from A2 T have (a·c)⁻¹·(b·c) = c⁻¹·a⁻¹·b·c
    using group˙inv˙of˙two group˙oper˙assoc by simp
  also from A1 A2 T have ... = a⁻¹·b
    using group0˙4˙L6B by simp
```

```
  finally show (a·c)^-1·(b·c) = a^-1·b
    by simp
  from A1 A2 T show a·(b·(c·a^-1·b^-1)) = c
    using group˙oper˙assoc group0˙4˙L6B group0˙4˙L6A
    by simp
  from T have a·b·c^-1·(c·a^-1) = a·b·(c^-1·(c·a^-1))
    using group˙oper˙assoc by simp
  also from A1 A2 T have ... = b
    using group˙oper˙assoc group0˙2˙L6 group0˙2˙L2 group0˙4˙L6A
    by simp
  finally show a·b·c^-1·(c·a^-1) = b by simp
qed
```

Another useful rearrangement of three elements and cancelling.

```
lemma (in group0) group0˙4˙L6E:
  assumes A1: P –is commutative on″ G
  and A2: a∈G  b∈G  c∈G
  shows
  a·b·(a·c)^-1 = b·c^-1
proof -
  from A2 have T: b^-1 ∈ G  c^-1 ∈ G
    using inverse˙in˙group by auto
  with A1 A2 have
    a·(b^-1)^-1·(a·(c^-1)^-1)^-1 = c^-1·(b^-1)^-1
    using group0˙4˙L6D by simp
  with A1 A2 T show a·b·(a·c)^-1 = b·c^-1
    using group˙inv˙of˙inv IsCommutative˙def
    by simp
qed
```

A rearrangement with two elements and canceelling, special case of group0˙4˙L6D when $c = b^{-1}$.

```
lemma (in group0) group0˙4˙L6F:
  assumes A1: P –is commutative on″ G
  and A2: a∈G  b∈G
  shows a·b^-1·(a·b)^-1 = b^-1·b^-1
proof -
  from A2 have b^-1 ∈ G
    using inverse˙in˙group by simp
  with A1 A2 have a·b^-1·(a·(b^-1)^-1)^-1 = b^-1·b^-1
    using group0˙4˙L6D by simp
  with A2 show a·b^-1·(a·b)^-1 = b^-1·b^-1
    using group˙inv˙of˙inv by simp
qed
```

Some other rearrangements with four elements. The algorithm for proof as in group0˙4˙L2 works very well here.

```
lemma (in group0) rearr˙ab˙gr˙4˙elemA:
```

```
  assumes A1: P –is commutative on″ G
  and A2: a∈G  b∈G  c∈G  d∈G
  shows
  a·b·c·d = a·d·b·c
  a·b·c·d = a·c·(b·d)
proof -
  from A1 A2 have a·b·c·d = d·(a·b·c)
    using  IsCommutative˙def group˙op˙closed
    by simp
  also from A2 have ... = d·a·b·c
    using group˙op˙closed group˙oper˙assoc
    by simp
  also from A1 A2 have ... = a·d·b·c
    using IsCommutative˙def group˙op˙closed
    by simp
  finally show a·b·c·d = a·d·b·c
    by simp
  from A1 A2 have a·b·c·d = c·(a·b)·d
    using IsCommutative˙def group˙op˙closed
    by simp
  also from A2 have ... = c·a·b·d
    using group˙op˙closed group˙oper˙assoc
    by simp
  also from A1 A2 have ... = a·c·b·d
    using IsCommutative˙def group˙op˙closed
    by simp
  also from A2 have ... = a·c·(b·d)
    using group˙op˙closed group˙oper˙assoc
    by simp
  finally show a·b·c·d = a·c·(b·d)
    by simp
qed
```

Some rearrangements with four elements and inverse that are applications of rearr˙ab˙gr˙4˙elem

```
lemma (in group0) rearr˙ab˙gr˙4˙elemB:
  assumes A1: P –is commutative on″ G
  and A2: a∈G  b∈G  c∈G  d∈G
  shows
  a·b⁻¹·c⁻¹·d⁻¹ = a·d⁻¹·b⁻¹·c⁻¹
  a·b·c·d⁻¹ = a·d⁻¹·b·c
  a·b·c⁻¹·d⁻¹ =  a·c⁻¹·(b·d⁻¹)
proof -
  from A2 have T: b⁻¹ ∈ G  c⁻¹ ∈ G  d⁻¹ ∈ G
    using inverse˙in˙group by auto
  with A1 A2 show
    a·b⁻¹·c⁻¹·d⁻¹ = a·d⁻¹·b⁻¹·c⁻¹
    a·b·c·d⁻¹ = a·d⁻¹·b·c
    a·b·c⁻¹·d⁻¹ =  a·c⁻¹·(b·d⁻¹)
```

using rearr˙ab˙gr˙4˙elemA by auto
qed

Some rearrangement lemmas with four elements.

lemma (in group0) group0˙4˙L7:
assumes A1: P –is commutative on″ G
and A2: a∈G b∈G c∈G d∈G
shows
$a\cdot b\cdot c\cdot d^{-1} = a\cdot d^{-1}\cdot b\cdot c$
$a\cdot d\cdot(b\cdot d\cdot(c\cdot d))^{-1} = a\cdot(b\cdot c)^{-1}\cdot d^{-1}$
$a\cdot(b\cdot c)\cdot d = a\cdot b\cdot d\cdot c$
proof -
from A2 have T:
$b\cdot c \in G$ $d^{-1} \in G$ $b^{-1}\in G$ $c^{-1}\in G$
$d^{-1}\cdot b \in G$ $c^{-1}\cdot d \in G$ $(b\cdot c)^{-1} \in G$
$b\cdot d \in G$ $b\cdot d\cdot c \in G$ $(b\cdot d\cdot c)^{-1} \in G$
$a\cdot d \in G$ $b\cdot c \in G$
using group˙op˙closed inverse˙in˙group
by auto
with A1 A2 have $a\cdot b\cdot c\cdot d^{-1} = a\cdot(d^{-1}\cdot b\cdot c)$
using group˙oper˙assoc group0˙4˙L4A by simp
also from A2 T have $a\cdot(d^{-1}\cdot b\cdot c) = a\cdot d^{-1}\cdot b\cdot c$
using group˙oper˙assoc by simp
finally show $a\cdot b\cdot c\cdot d^{-1} = a\cdot d^{-1}\cdot b\cdot c$ by simp
from A2 T have $a\cdot d\cdot(b\cdot d\cdot(c\cdot d))^{-1} = a\cdot d\cdot(d^{-1}\cdot(b\cdot d\cdot c)^{-1})$
using group˙oper˙assoc group˙inv˙of˙two by simp
also from A2 T have $\ldots = a\cdot(b\cdot d\cdot c)^{-1}$
using group˙oper˙assoc inv˙cancel˙two by simp
also from A1 A2 have $\ldots = a\cdot(d\cdot(b\cdot c))^{-1}$
using IsCommutative˙def group˙oper˙assoc by simp
also from A2 T have $\ldots = a\cdot((b\cdot c)^{-1}\cdot d^{-1})$
using group˙inv˙of˙two by simp
also from A2 T have $\ldots = a\cdot(b\cdot c)^{-1}\cdot d^{-1}$
using group˙oper˙assoc by simp
finally show $a\cdot d\cdot(b\cdot d\cdot(c\cdot d))^{-1} = a\cdot(b\cdot c)^{-1}\cdot d^{-1}$
by simp
from A2 have $a\cdot(b\cdot c)\cdot d = a\cdot(b\cdot(c\cdot d))$
using group˙op˙closed group˙oper˙assoc by simp
also from A1 A2 have $\ldots = a\cdot(b\cdot(d\cdot c))$
using IsCommutative˙def group˙op˙closed by simp
also from A2 have $\ldots = a\cdot b\cdot d\cdot c$
using group˙op˙closed group˙oper˙assoc by simp
finally show $a\cdot(b\cdot c)\cdot d = a\cdot b\cdot d\cdot c$ by simp
qed

Some other rearrangements with four elements.

lemma (in group0) group0˙4˙L8:
assumes A1: P –is commutative on″ G
and A2: a∈G b∈G c∈G d∈G

```
  shows
  a·(b·c)^-1 = (a·d^-1·c^-1)·(d·b^-1)
  a·b·(c·d) = c·a·(b·d)
  a·b·(c·d) = a·c·(b·d)
  a·(b·c^-1)·d = a·b·d·c^-1
  (a·b)·(c·d)^-1·(b·d^-1)^-1 = a·c^-1
proof -
  from A2 have T:
    b·c ∈ G a·b ∈ G d^-1 ∈ G b^-1∈G c^-1∈G
    d^-1·b ∈ G c^-1·d ∈ G (b·c)^-1 ∈ G
    a·b ∈ G  (c·d)^-1 ∈ G  (b·d^-1)^-1 ∈ G  d·b^-1 ∈ G
    using group˙op˙closed inverse˙in˙group
    by auto
  from A2 have a·(b·c)^-1 = a·c^-1·b^-1 using group0˙2˙L14A by blast
  moreover from A2 have a·c^-1 = (a·d^-1)·(d·c^-1) using group0˙2˙L14A
    by blast
  ultimately have a·(b·c)^-1 = (a·d^-1)·(d·c^-1)·b^-1 by simp
  with A1 A2 T have a·(b·c)^-1= a·d^-1·(c^-1·d)·b^-1
    using IsCommutative˙def by simp
  with A2 T show a·(b·c)^-1 = (a·d^-1·c^-1)·(d·b^-1)
    using group˙op˙closed group˙oper˙assoc by simp
  from A2 T have a·b·(c·d) = a·b·c·d
    using group˙oper˙assoc by simp
  also have a·b·c·d = c·a·b·d
  proof -
    from A1 A2 have a·b·c·d = c·(a·b)·d
      using IsCommutative˙def group˙op˙closed
      by simp
    also from A2 have ... = c·a·b·d
      using group˙op˙closed group˙oper˙assoc
      by simp
    finally show thesis by simp
  qed
  also from A2 have c·a·b·d =  c·a·(b·d)
    using group˙op˙closed group˙oper˙assoc
    by simp
  finally show a·b·(c·d) = c·a·(b·d) by simp
  with A1 A2 show a·b·(c·d) = a·c·(b·d)
    using IsCommutative˙def by simp
  from A1 A2 T show a·(b·c^-1)·d = a·b·d·c^-1
    using group0˙4˙L7 by simp
  from T have (a·b)·(c·d)^-1·(b·d^-1)^-1 = (a·b)·((c·d)^-1·(b·d^-1)^-1)
    using group˙oper˙assoc by simp
  also from A1 A2 T have ... = (a·b)·(c^-1·d^-1·(d·b^-1))
    using group˙inv˙of˙two group0˙2˙L12 IsCommutative˙def
    by simp
  also from T have ... = (a·b)·(c^-1·(d^-1·(d·b^-1)))
    using group˙oper˙assoc by simp
  also from A1 A2 T have ... = a·c^-1
```

using group˙oper˙assoc group0˙2˙L6 group0˙2˙L2 IsCommutative˙def
inv˙cancel˙two by simp
finally show $(a\cdot b)\cdot(c\cdot d)^{-1}\cdot(b\cdot d^{-1})^{-1} = a\cdot c^{-1}$
by simp
qed

Some other rearrangements with four elements.

lemma (in group0) group0˙4˙L8A:
assumes A1: P –is commutative on″ G
and A2: $a\in G$ $b\in G$ $c\in G$ $d\in G$
shows
$a\cdot b^{-1}\cdot(c\cdot d^{-1}) = a\cdot c\cdot(b^{-1}\cdot d^{-1})$
$a\cdot b^{-1}\cdot(c\cdot d^{-1}) = a\cdot c\cdot b^{-1}\cdot d^{-1}$
proof -
from A2 have
T: $a\in G$ $b^{-1} \in G$ $c\in G$ $d^{-1} \in G$
using inverse˙in˙group by auto
with A1 show $a\cdot b^{-1}\cdot(c\cdot d^{-1}) = a\cdot c\cdot(b^{-1}\cdot d^{-1})$
by (rule group0˙4˙L8)
with A2 T show $a\cdot b^{-1}\cdot(c\cdot d^{-1}) = a\cdot c\cdot b^{-1}\cdot d^{-1}$
using group˙op˙closed group˙oper˙assoc
by simp
qed

Some rearrangements with an equation.

lemma (in group0) group0˙4˙L9:
assumes A1: P –is commutative on″ G
and A2: $a\in G$ $b\in G$ $c\in G$ $d\in G$
and A3: $a = b\cdot c^{-1}\cdot d^{-1}$
shows
$d = b\cdot a^{-1}\cdot c^{-1}$
$d = a^{-1}\cdot b\cdot c^{-1}$
$b = a\cdot d\cdot c$
proof -
from A2 have T:
$a^{-1} \in G$ $c^{-1} \in G$ $d^{-1} \in G$ $b\cdot c^{-1} \in G$
using group˙op˙closed inverse˙in˙group
by auto
with A2 A3 have $a\cdot(d^{-1})^{-1} = b\cdot c^{-1}$
using group0˙2˙L18 by simp
with A2 have $b\cdot c^{-1} = a\cdot d$
using group˙inv˙of˙inv by simp
with A2 T have I: $a^{-1}\cdot(b\cdot c^{-1}) = d$
using group0˙2˙L18 by simp
with A1 A2 T show
$d = b\cdot a^{-1}\cdot c^{-1}$
$d = a^{-1}\cdot b\cdot c^{-1}$
using group˙oper˙assoc IsCommutative˙def by auto
from A3 have $a\cdot d\cdot c = (b\cdot c^{-1}\cdot d^{-1})\cdot d\cdot c$ by simp

also from A2 T have ... = b·c^{-1}·(d^{-1}·d)·c
using group˙oper˙assoc by simp
also from A2 T have ... = b·c^{-1}·c
using group0˙2˙L6 group0˙2˙L2 by simp
also from A2 T have ... = b·(c^{-1}·c)
using group˙oper˙assoc by simp
also from A2 have ... = b
using group0˙2˙L6 group0˙2˙L2 by simp
finally have a·d·c = b by simp
thus b = a·d·c by simp
qed

end

30 Groups 2

theory Group˙ZF˙2 imports AbelianGroup˙ZF func˙ZF EquivClass1

begin

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

30.1 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 unit˙is˙neutral func1˙3˙L1 by simp
  show ∀s∈X→G. F⟨ E,s⟩ = s ∧ F⟨ s,E⟩ = s
  proof
    fix s assume A3:s:X→G
    from 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 unit˙is˙neutral by auto
    ultimately show
      F⟨ E,s⟩ = s ∧ F⟨ s,E⟩ = s
      using fun˙extension˙iff by auto
  qed
qed
```

Monoids can be lifted to a function space.

```
lemma (in monoid0) Group˙ZF˙2˙1˙T1:
  assumes A1: F = f –lifted to function space over″ X
  shows IsAmonoid(X→G,F)
proof -
  from 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 monoid operation.

```
lemma (in monoid0) lifted˙val:
  assumes F = f –lifted to function space over″ X
  and s:X→G  r:X→G
  and x∈X
  shows (F⟨s,r⟩)(x) = s(x) ⊕ r(x)
  using monoidAsssum assms IsAmonoid˙def IsAssociative˙def
      group0˙1˙L3B func˙ZF˙1˙L4
  by auto
```

The lifted operation acts on the functions in a natural way defined by the group operation. This is the same as lifted˙val, but in the group0 context.

```
lemma (in group0) Group˙ZF˙2˙1˙L3:
  assumes F = P –lifted to function space over″ X
  and s:X→G r:X→G
  and x∈X
  shows (F⟨s,r⟩)(x) = s(x)·r(x)
  using assms group0˙2˙L1 monoid0.lifted˙val by simp
```

In the group0 context we can apply theorems proven in monoid0 context to the lifted monoid.

```
lemma (in group0) Group˙ZF˙2˙1˙L4:
  assumes A1: F = P –lifted to function space over″ X
  shows monoid0(X→G,F)
proof -
```

```
  from A1 show thesis
    using group0˙2˙L1 monoid0.Group˙ZF˙2˙1˙T1 monoid0˙def
    by simp
qed
```

The compostion of a function $f : X \to G$ with the group inverse is a right inverse for the lifted group.

```
lemma (in group0) Group˙ZF˙2˙1˙L5:
  assumes A1: F = P –lifted to function space over″ X
  and A2: s : X→G
  and A3: i = GroupInv(G,P) O s
  shows i: X→G and F⟨ s,i⟩ = TheNeutralElement(X→G,F)
proof -
  let E = ConstantFunction(X,1)
  have E : X→G
    using group0˙2˙L2 func1˙3˙L1 by simp
  moreover from groupAssum A2 A3 A1 have
    F⟨ s,i⟩ :  X→G using group0˙2˙T2 comp˙fun
      Group˙ZF˙2˙1˙L4 monoid0.group0˙1˙L1
    by simp
  moreover from groupAssum A2 A3 A1 have
    ∀x∈X. (F⟨ s,i⟩)(x) = E(x)
    using group0˙2˙T2 comp˙fun Group˙ZF˙2˙1˙L3
      comp˙fun˙apply apply˙funtype group0˙2˙L6 func1˙3˙L2
    by simp
  moreover from groupAssum A1 have
    E = TheNeutralElement(X→G,F)
    using IsAgroup˙def Group˙ZF˙2˙1˙L2 by simp
  ultimately show F⟨ s,i⟩ = TheNeutralElement(X→G,F)
    using fun˙extension˙iff IsAgroup˙def Group˙ZF˙2˙1˙L2
    by simp
  from groupAssum A2 A3 show i: X→G
    using group0˙2˙T2 comp˙fun by simp
qed
```

Groups can be lifted to the function space.

```
theorem (in group0) Group˙ZF˙2˙1˙T2:
  assumes A1: F = P –lifted to function space over″ X
  shows IsAgroup(X→G,F)
proof -
  from A1 have IsAmonoid(X→G,F)
    using group0˙2˙L1 monoid0.Group˙ZF˙2˙1˙T1
    by simp
  moreover have
    ∀s∈X→G. ∃i∈X→G. F⟨ s,i⟩ = TheNeutralElement(X→G,F)
  proof
    fix s assume A2: s : X→G
    let i = GroupInv(G,P) O s
    from groupAssum A2 have i:X→G
```

```
    using group0˙2˙T2 comp˙fun by simp
  moreover from A1 A2 have
    F⟨ s,i⟩ = TheNeutralElement(X→G,F)
    using Group˙ZF˙2˙1˙L5 by fast
 ultimately show ∃i∈X→G. F⟨ s,i⟩ = TheNeutralElement(X→G,F)
    by auto
qed
ultimately show thesis using IsAgroup˙def
  by simp
qed
```

What is the group inverse for the lifted group?

```
lemma (in group0) Group˙ZF˙2˙1˙L6:
  assumes A1: F = P –lifted to function space over″ X
  shows ∀s∈(X→G). GroupInv(X→G,F)(s) = GroupInv(G,P) O s
proof -
  from A1 have  group0(X→G,F)
    using group0˙def Group˙ZF˙2˙1˙T2
    by simp
  moreover from A1 have ∀s∈X→G. GroupInv(G,P) O s : X→G ∧
    F⟨ s,GroupInv(G,P) O s⟩ = TheNeutralElement(X→G,F)
    using Group˙ZF˙2˙1˙L5 by simp
  ultimately have
    ∀s∈(X→G).  GroupInv(G,P) O s = GroupInv(X→G,F)(s)
    by (rule group0.group0˙2˙L9A)
  thus thesis by simp
qed
```

What is the value of the group inverse for the lifted group?

```
corollary (in group0) lift˙gr˙inv˙val:
  assumes F = P –lifted to function space over″ X and
  s : X→G and x∈X
  shows  (GroupInv(X→G,F)(s))(x) = (s(x))⁻¹
  using groupAssum assms Group˙ZF˙2˙1˙L6 group0˙2˙T2 comp˙fun˙apply
  by simp
```

What is the group inverse in a subgroup of the lifted group?

```
lemma (in group0) Group˙ZF˙2˙1˙L6A:
  assumes A1: F = P –lifted to function space over″ X
  and A2: IsAsubgroup(H,F)
  and A3: g = restrict(F,H×H)
  and A4: s∈H
  shows GroupInv(H,g)(s) = GroupInv(G,P) O s
proof -
  from A1 have T1: group0(X→G,F)
    using group0˙def Group˙ZF˙2˙1˙T2
    by simp
  with A2 A3 A4 have GroupInv(H,g)(s) = GroupInv(X→G,F)(s)
    using group0.group0˙3˙T1 restrict by simp
```

```
  moreover from T1 A1 A2 A4 have
    GroupInv(X→G,F)(s) = GroupInv(G,P) O s
    using group0.group0˙3˙L2 Group˙ZF˙2˙1˙L6 by blast
  ultimately show thesis by simp
qed
```

If a group is abelian, then its lift to a function space is also abelian.

```
lemma (in group0) Group˙ZF˙2˙1˙L7:
  assumes A1: F = P –lifted to function space over″ X
  and A2: P –is commutative on″ G
  shows F –is commutative on″ (X→G)
proof-
  from A1 A2  have
    F –is commutative on″ (X→range(P))
    using group˙oper˙assocA func˙ZF˙2˙L2
    by simp
  moreover from groupAssum have range(P) = G
    using group0˙2˙L1 monoid0.group0˙1˙L3B
    by simp
  ultimately show thesis by simp
qed
```

30.2 Equivalence relations on groups

The goal of this section is to establish that (under some conditions) given an equivalence relation on a group or (monoid)we can project the group (monoid) structure on the quotient and obtain another group.

The neutral element class is neutral in the projection.

```
lemma (in monoid0) Group˙ZF˙2˙2˙L1:
  assumes A1: equiv(G,r) and A2:Congruent2(r,f)
  and A3: F = ProjFun2(G,r,f)
  and A4: e = TheNeutralElement(G,f)
  shows r–e″ ∈ G//r ∧
  (∀c ∈ G//r. F⟨ r–e″,c⟩ = c ∧  F⟨ c,r–e″⟩ = c)
proof
  from A4 show T1:r–e″ ∈ G//r
    using unit˙is˙neutral quotientI
    by simp
  show
    ∀c ∈ G//r. F⟨ r–e″,c⟩ = c ∧  F⟨ c,r–e″⟩ = c
  proof
    fix c assume A5:c ∈ G//r
    then obtain g where D1:g∈G c = r–g″
      using quotient˙def by auto
    with A1 A2 A3 A4 D1 show
      F⟨ r–e″,c⟩ = c ∧  F⟨ c,r–e″⟩ = c
      using unit˙is˙neutral EquivClass˙1˙L10
      by simp
```

```
  qed
qed
```

The projected structure is a monoid.

```
theorem (in monoid0) Group˙ZF˙2˙2˙T1:
  assumes A1: equiv(G,r) and A2: Congruent2(r,f)
  and A3: F = ProjFun2(G,r,f)
  shows IsAmonoid(G//r,F)
proof -
  let E = r–TheNeutralElement(G,f)″
  from A1 A2 A3 have
    E ∈ G//r ∧ (∀c∈G//r. F⟨ E,c⟩ = c ∧ F⟨ c,E⟩ = c)
    using Group˙ZF˙2˙2˙L1 by simp
  hence
    ∃E∈G//r. ∀ c∈G//r. F⟨ E,c⟩ = c ∧ F⟨ c,E⟩ = c
    by auto
  with 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,P)
  and A3: F = ProjFun2(G,r,P)
  and A4: a∈G b∈G
  shows F⟨ r–a″,r–b″⟩ = r–a·b″
proof -
```

```
  from A1 A2 A3 A4 show thesis
    using EquivClass˙1˙L10 by simp
qed
```

The class of the inverse is a right inverse of the class.

```
lemma (in group0) Group˙ZF˙2˙2˙L3:
  assumes A1: equiv(G,r) and A2: Congruent2(r,P)
  and A3: F = ProjFun2(G,r,P)
  and A4: a∈G
  shows F⟨r–a″,r–a⁻¹″⟩ = TheNeutralElement(G//r,F)
proof -
  from A1 A2 A3 A4 have
    F⟨r–a″,r–a⁻¹″⟩ = r–1″
    using inverse˙in˙group Group˙ZF˙2˙2˙L2 group0˙2˙L6
    by simp
  with groupAssum A1 A2 A3 show thesis
    using IsAgroup˙def Group˙ZF˙2˙2˙L1 by simp
qed
```

The group structure can be projected to the quotient space.

```
theorem (in group0) Group˙ZF˙3˙T2:
  assumes A1: equiv(G,r) and A2: Congruent2(r,P)
  shows IsAgroup(G//r,ProjFun2(G,r,P))
proof -
  let F = ProjFun2(G,r,P)
  let E = TheNeutralElement(G//r,F)
  from groupAssum A1 A2 have IsAmonoid(G//r,F)
    using IsAgroup˙def monoid0˙def monoid0.Group˙ZF˙2˙2˙T1
    by simp
  moreover have
    ∀c∈G//r. ∃b∈G//r. F⟨ c,b⟩ = E
  proof
    fix c assume A3: c ∈ G//r
    then obtain g where D1: g∈G  c = r–g″
      using quotient˙def by auto
    let b = r–g⁻¹″
    from D1 have b ∈ G//r
      using inverse˙in˙group quotientI
      by simp
    moreover from A1 A2 D1 have
      F⟨ c,b⟩ = E
      using Group˙ZF˙2˙2˙L3 by simp
    ultimately show ∃b∈G//r. F⟨ c,b⟩ = E
      by auto
  qed
  ultimately show thesis
    using IsAgroup˙def by simp
qed
```

The group inverse (in the projected group) of a class is the class of the

inverse.

```
lemma (in group0) Group˙ZF˙2˙2˙L4:
  assumes A1: equiv(G,r) and
  A2: Congruent2(r,P) and
  A3: F = ProjFun2(G,r,P) and
  A4: a∈G
  shows r–a^{-1}″ = GroupInv(G//r,F)(r–a″)
proof -
  from A1 A2 A3 have group0(G//r,F)
    using Group˙ZF˙3˙T2 group0˙def by simp
  moreover from A4 have
    r–a″ ∈ G//r  r–a^{-1}″ ∈ G//r
    using inverse˙in˙group quotientI by auto
  moreover from A1 A2 A3 A4 have
     F⟨r–a″,r–a^{-1}″⟩ = TheNeutralElement(G//r,F)
    using Group˙ZF˙2˙2˙L3 by simp
  ultimately show thesis
    by (rule group0.group0˙2˙L9)
qed
```

30.3 Normal subgroups and quotient groups

If H is a subgroup of G, then for every $a \in G$ we can cosider the sets $\{a \cdot h.h \in H\}$ and $\{h \cdot a.h \in H\}$ (called a left and right "coset of H", resp.) These sets sometimes form a group, called the "quotient group". This section discusses the notion of quotient groups.

A normal subgorup N of a group G is such that aba^{-1} belongs to N if $a \in G, b \in N$.

```
definition
  IsAnormalSubgroup(G,P,N) ≡ IsAsubgroup(N,P) ∧
  (∀n∈N.∀g∈G. P⟨  P⟨  g,n ⟩,GroupInv(G,P)(g) ⟩ ∈ N)
```

Having a group and a normal subgroup N we can create another group consisting of eqivalence classes of the relation $a \sim b \equiv a \cdot b^{-1} \in N$. We will refer to this relation as the quotient group relation. The classes of this relation are in fact cosets of subgroup H.

```
definition
  QuotientGroupRel(G,P,H) ≡
  –⟨ a,b⟩ ∈ G×G. P⟨ a, GroupInv(G,P)(b)⟩ ∈ H″
```

Next we define the operation in the quotient group as the projection of the group operation on the classses of the quotient group relation.

```
definition
  QuotientGroupOp(G,P,H) ≡ ProjFun2(G,QuotientGroupRel(G,P,H ),P)
```

Definition of a normal subgroup in a more readable notation.

```
lemma (in group0) Group˙ZF˙2˙4˙L0:
  assumes IsAnormalSubgroup(G,P,H)
  and g∈G n∈H
  shows g·n·g⁻¹ ∈ H
  using assms IsAnormalSubgroup˙def by simp
```

The quotient group relation is reflexive.

```
lemma (in group0) Group˙ZF˙2˙4˙L1:
  assumes IsAsubgroup(H,P)
  shows refl(G,QuotientGroupRel(G,P,H))
  using assms  group0˙2˙L6 group0˙3˙L5
    QuotientGroupRel˙def refl˙def by simp
```

The quotient group relation is symmetric.

```
lemma (in group0) Group˙ZF˙2˙4˙L2:
  assumes A1:IsAsubgroup(H,P)
  shows sym(QuotientGroupRel(G,P,H))
proof -
  {
    fix a b assume A2: ⟨ a,b⟩ ∈ QuotientGroupRel(G,P,H)
    with A1 have (a·b⁻¹)⁻¹ ∈ H
      using QuotientGroupRel˙def group0˙3˙T3A
      by simp
    moreover from A2 have (a·b⁻¹)⁻¹ =  b·a⁻¹
      using QuotientGroupRel˙def group0˙2˙L12
      by simp
    ultimately have b·a⁻¹ ∈ H by simp
    with A2 have ⟨ b,a⟩ ∈ QuotientGroupRel(G,P,H)
      using QuotientGroupRel˙def by simp
  }
  then show thesis using symI by simp
qed
```

The quotient group relation is transistive.

```
lemma (in group0) Group˙ZF˙2˙4˙L3A:
  assumes A1: IsAsubgroup(H,P) and
  A2: ⟨ a,b⟩ ∈ QuotientGroupRel(G,P,H) and
  A3: ⟨ b,c⟩ ∈ QuotientGroupRel(G,P,H)
  shows ⟨ a,c⟩ ∈ QuotientGroupRel(G,P,H)
proof -
  let r = QuotientGroupRel(G,P,H)
  from A2 A3 have T1:a∈G b∈G c∈G
    using QuotientGroupRel˙def by auto
  from A1 A2 A3 have (a·b⁻¹)·(b·c⁻¹) ∈ H
    using  QuotientGroupRel˙def group0˙3˙L6
    by simp
  moreover from T1 have
    a·c⁻¹ = (a·b⁻¹)·(b·c⁻¹)
    using group0˙2˙L14A by blast
```

```
  ultimately have a·c^-1 ∈ H
    by simp
  with T1 show thesis using QuotientGroupRel˙def
    by simp
qed
```

The quotient group relation is an equivalence relation. Note we do not need the subgroup to be normal for this to be true.

```
lemma (in group0) Group˙ZF˙2˙4˙L3: assumes A1:IsAsubgroup(H,P)
  shows equiv(G,QuotientGroupRel(G,P,H))
proof -
  let r = QuotientGroupRel(G,P,H)
  from A1 have
     ∀a b c. (⟨a, b⟩ ∈ r  ∧  ⟨b, c⟩ ∈ r ⟶ ⟨a, c⟩ ∈ r)
    using Group˙ZF˙2˙4˙L3A by blast
  then have trans(r)
    using Fol1˙L2 by blast
  with A1 show thesis
    using Group˙ZF˙2˙4˙L1 Group˙ZF˙2˙4˙L2
      QuotientGroupRel˙def equiv˙def
    by auto
qed
```

The next lemma states the essential condition for congruency of the group operation with respect to the quotient group relation.

```
lemma (in group0) Group˙ZF˙2˙4˙L4:
  assumes A1: IsAnormalSubgroup(G,P,H)
  and A2: ⟨a1,a2⟩ ∈ QuotientGroupRel(G,P,H)
  and A3: ⟨b1,b2⟩ ∈ QuotientGroupRel(G,P,H)
  shows ⟨a1·b1, a2·b2⟩ ∈ QuotientGroupRel(G,P,H)
proof -
  from A2 A3 have T1:
    a1∈G  a2∈G  b1∈G  b2∈G
    a1·b1 ∈ G  a2·b2 ∈ G
    b1·b2^-1 ∈ H  a1·a2^-1 ∈ H
    using QuotientGroupRel˙def group0˙2˙L1 monoid0.group0˙1˙L1
    by auto
  with A1 show thesis using
    IsAnormalSubgroup˙def group0˙3˙L6 group0˙2˙L15
    QuotientGroupRel˙def by simp
qed
```

If the subgroup is normal, the group operation is congruent with respect to the quotient group relation.

```
lemma Group˙ZF˙2˙4˙L5A:
  assumes IsAgroup(G,P)
  and IsAnormalSubgroup(G,P,H)
  shows Congruent2(QuotientGroupRel(G,P,H),P)
```

```
  using assms group0˙def group0.Group˙ZF˙2˙4˙L4 Congruent2˙def
  by simp
```

The quotient group is indeed a group.

```
theorem Group˙ZF˙2˙4˙T1:
  assumes IsAgroup(G,P) and IsAnormalSubgroup(G,P,H)
  shows
  IsAgroup(G//QuotientGroupRel(G,P,H),QuotientGroupOp(G,P,H))
  using assms group0˙def group0.Group˙ZF˙2˙4˙L3 IsAnormalSubgroup˙def
    Group˙ZF˙2˙4˙L5A group0.Group˙ZF˙3˙T2 QuotientGroupOp˙def
  by simp
```

The class (coset) of the neutral element is the neutral element of the quotient group.

```
lemma Group˙ZF˙2˙4˙L5B:
  assumes IsAgroup(G,P) and IsAnormalSubgroup(G,P,H)
  and r = QuotientGroupRel(G,P,H)
  and e = TheNeutralElement(G,P)
  shows  r–e″ = TheNeutralElement(G//r,QuotientGroupOp(G,P,H))
  using assms IsAnormalSubgroup˙def group0˙def
    IsAgroup˙def group0.Group˙ZF˙2˙4˙L3 Group˙ZF˙2˙4˙L5A
    QuotientGroupOp˙def Group˙ZF˙2˙2˙L1
  by simp
```

A group element is equivalent to the neutral element iff it is in the subgroup we divide the group by.

```
lemma (in group0) Group˙ZF˙2˙4˙L5C: assumes a∈G
  shows ⟨a,1⟩ ∈ QuotientGroupRel(G,P,H) ⟷ a∈H
  using assms QuotientGroupRel˙def group˙inv˙of˙one group0˙2˙L2
  by auto
```

A group element is in H iff its class is the neutral element of G/H.

```
lemma (in group0) Group˙ZF˙2˙4˙L5D:
  assumes A1: IsAnormalSubgroup(G,P,H) and
  A2: a∈G and
  A3: r = QuotientGroupRel(G,P,H) and
  A4: TheNeutralElement(G//r,QuotientGroupOp(G,P,H)) = e
  shows r–a″ = e ⟷ ⟨a,1⟩ ∈ r
proof
  assume r–a″ = e
  with groupAssum assms have
    r–1″ = r–a″ and I: equiv(G,r)
    using Group˙ZF˙2˙4˙L5B IsAnormalSubgroup˙def Group˙ZF˙2˙4˙L3
    by auto
  with A2 have ⟨1,a⟩ ∈ r using eq˙equiv˙class
    by simp
  with I show ⟨a,1⟩ ∈ r by (rule equiv˙is˙sym)
next assume ⟨a,1⟩ ∈ r
```

```
  moreover from A1 A3 have equiv(G,r)
    using IsAnormalSubgroup˙def Group˙ZF˙2˙4˙L3
    by simp
  ultimately have r–a″ = r–1″
    using equiv˙class˙eq by simp
  with groupAssum A1 A3 A4 show r–a″ = e
    using Group˙ZF˙2˙4˙L5B by simp
qed
```

The class of $a \in G$ is the neutral element of the quotient G/H iff $a \in H$.

```
lemma (in group0) Group˙ZF˙2˙4˙L5E:
  assumes IsAnormalSubgroup(G,P,H) and
  a∈G and r = QuotientGroupRel(G,P,H) and
  TheNeutralElement(G//r,QuotientGroupOp(G,P,H)) = e
  shows r–a″ = e ⟷ a∈H
  using assms Group˙ZF˙2˙4˙L5C  Group˙ZF˙2˙4˙L5D
  by simp
```

Essential condition to show that every subgroup of an abelian group is normal.

```
lemma (in group0) Group˙ZF˙2˙4˙L5:
  assumes A1: P –is commutative on″ G
  and A2: IsAsubgroup(H,P)
  and A3: g∈G  h∈H
  shows g·h·g⁻¹ ∈ H
proof -
  from A2 A3 have T1:h∈G g⁻¹ ∈ G
    using group0˙3˙L2 inverse˙in˙group by auto
  with A3 A1 have g·h·g⁻¹ = g⁻¹·g·h
    using group0˙4˙L4A by simp
  with A3 T1 show thesis using
    group0˙2˙L6 group0˙2˙L2
    by simp
qed
```

Every subgroup of an abelian group is normal. Moreover, the quotient group is also abelian.

```
lemma Group˙ZF˙2˙4˙L6:
  assumes A1: IsAgroup(G,P)
  and A2: P –is commutative on″ G
  and A3: IsAsubgroup(H,P)
  shows  IsAnormalSubgroup(G,P,H)
  QuotientGroupOp(G,P,H) –is commutative on″ (G//QuotientGroupRel(G,P,H))
proof -
  from A1 A2 A3 show T1: IsAnormalSubgroup(G,P,H) using
    group0˙def IsAnormalSubgroup˙def group0.Group˙ZF˙2˙4˙L5
    by simp
  let r = QuotientGroupRel(G,P,H)
```

```
  from A1 A3 T1 have equiv(G,r) Congruent2(r,P)
    using group0˙def group0.Group˙ZF˙2˙4˙L3 Group˙ZF˙2˙4˙L5A
    by auto
  with A2 show
   QuotientGroupOp(G,P,H) –is commutative on″ (G//QuotientGroupRel(G,P,H))
    using EquivClass˙2˙T1 QuotientGroupOp˙def
    by simp
qed
```

The group inverse (in the quotient group) of a class (coset) is the class of the inverse.

```
lemma (in group0) Group˙ZF˙2˙4˙L7:
  assumes IsAnormalSubgroup(G,P,H)
  and a∈G and r = QuotientGroupRel(G,P,H)
  and F = QuotientGroupOp(G,P,H)
  shows r–a⁻¹″ = GroupInv(G//r,F)(r–a″)
  using groupAssum assms IsAnormalSubgroup˙def Group˙ZF˙2˙4˙L3
    Group˙ZF˙2˙4˙L5A QuotientGroupOp˙def Group˙ZF˙2˙2˙L4
  by simp
```

30.4 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

end
```

31 Groups 3

theory Group˙ZF˙3 imports Group˙ZF˙2 Finite1

begin

In this theory we consider notions in group theory that are useful for the construction of real numbers in the Real˙ZF˙x series of theories.

31.1 Group valued finite range functions

In this section show that the group valued functions $f : X \to G$, with the property that $f(X)$ is a finite subset of G, is a group. Such functions play an important role in the construction of real numbers in the Real˙ZF series.

The following proves the essential condition to show that the set of finite range functions is closed with respect to the lifted group operation.

```
lemma (in group0) Group˙ZF˙3˙1˙L1:
  assumes A1: F = P –lifted to function space over″ X
  and
  A2: s ∈ FinRangeFunctions(X,G)  r ∈  FinRangeFunctions(X,G)
  shows F⟨ s,r⟩ ∈ FinRangeFunctions(X,G)
proof -
  let q = F⟨ s,r⟩
  from A2 have T1:s:X→G r:X→G
    using FinRangeFunctions˙def by auto
  with A1 have T2:q : X→G
    using group0˙2˙L1 monoid0.Group˙ZF˙2˙1˙L0
    by simp
  moreover have q(X) ∈ Fin(G)
  proof -
    from A2 have
      –s(x). x∈X″ ∈ Fin(G)
      –r(x). x∈X″ ∈ Fin(G)
      using Finite1˙L18 by auto
    with A1 T1 T2 show thesis using
      group˙oper˙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 = P –lifted to function space over″ X
  shows FinRangeFunctions(X,G) –is closed under″ F
proof -
  let A = FinRangeFunctions(X,G)
  from A1 have ∀x∈A. ∀y∈A. F⟨ x,y⟩ ∈ A
    using Group˙ZF˙3˙1˙L1 by simp
  then show thesis using IsOpClosed˙def by simp
qed
```

A composition of a finite range function with the group inverse is a finite range function.

```
lemma (in group0) Group˙ZF˙3˙1˙L3:
  assumes A1: s ∈ FinRangeFunctions(X,G)
  shows GroupInv(G,P) O s ∈ FinRangeFunctions(X,G)
  using groupAssum assms group0˙2˙T2 Finite1˙L20 by simp
```

The set of finite range functions is s subgroup of the lifted group.

```
theorem Group˙ZF˙3˙1˙T1:
  assumes A1: IsAgroup(G,P)
  and A2: F = P –lifted to function space over″ X
  and A3: X≠0
  shows IsAsubgroup(FinRangeFunctions(X,G),F)
proof -
  let e = TheNeutralElement(G,P)
  let S = FinRangeFunctions(X,G)
  from A1 have T1: group0(G,P) using group0˙def
    by simp
  with A1 A2 have T2:group0(X→G,F)
    using group0.Group˙ZF˙2˙1˙T2 group0˙def
    by simp
  moreover have S ≠ 0
  proof -
    from T1 A3 have
      ConstantFunction(X,e) ∈ S
      using group0.group0˙2˙L1 monoid0.unit˙is˙neutral
 Finite1˙L17 by simp
    thus thesis by auto
  qed
  moreover have S ⊆ X→G
```

```
    using FinRangeFunctions˙def by auto
  moreover from A2 T1 have
    S –is closed under″ F
    using group0.Group˙ZF˙3˙1˙L2
    by simp
  moreover from A1 A2 T1 have
    ∀s ∈ S. GroupInv(X→G,F)(s) ∈ S
    using FinRangeFunctions˙def group0.Group˙ZF˙2˙1˙L6
      group0.Group˙ZF˙3˙1˙L3 by simp
  ultimately show thesis
    using group0.group0˙3˙T3 by simp
qed
```

31.2 Almost homomorphisms

An almost homomorphism is a group valued function defined on a monoid M with the property that the set $\{f(m+n)-f(m)-f(n)\}_{m,n\in M}$ is finite. This term is used by R. D. Arthan in "The Eudoxus Real Numbers". We use this term in the general group context and use the A'Campo's term "slopes" (see his "A natural construction for the real numbers") to mean an almost homomorphism mapping interegers into themselves. We consider almost homomorphisms because we use slopes to define real numbers in the Real˙ZF˙x series.

HomDiff is an acronym for "homomorphism difference". This is the expression $s(mn)(s(m)s(n))^{-1}$, or $s(m+n)-s(m)-s(n)$ in the additive notation. It is equal to the neutral element of the group if s is a homomorphism.

```
definition
  HomDiff(G,f,s,x) ≡
  f⟨s(f⟨ fst(x),snd(x)⟩) ,
  (GroupInv(G,f)(f⟨ s(fst(x)),s(snd(x))⟩))⟩
```

Almost homomorphisms are defined as those maps $s : G \to G$ such that the homomorphism difference takes only finite number of values on $G \times G$.

```
definition
  AlmostHoms(G,f) ≡
  –s ∈ G→G.–HomDiff(G,f,s,x). x ∈ G×G ″ ∈ Fin(G)″
```

AlHomOp1(G, f) is the group operation on almost homomorphisms defined in a natural way by $(s \cdot r)(n) = s(n) \cdot r(n)$. In the terminology defined in func1.thy this is the group operation f (on G) lifted to the function space $G \to G$ and restricted to the set AlmostHoms(G, f).

```
definition
  AlHomOp1(G,f) ≡
  restrict(f –lifted to function space over″ G,
  AlmostHoms(G,f)×AlmostHoms(G,f))
```

We also define a composition (binary) operator on almost homomorphisms in a natural way. We call that operator AlHomOp2 - the second operation on almost homomorphisms. Composition of almost homomorphisms is used to define multiplication of real numbers in Real˙ZF series.

```
definition
  AlHomOp2(G,f) ≡
  restrict(Composition(G),AlmostHoms(G,f)×AlmostHoms(G,f))
```

This lemma provides more readable notation for the HomDiff definition. Not really intended to be used in proofs, but just to see the definition in the notation defined in the group0 locale.

```
lemma (in group0) HomDiff˙notation:
  shows HomDiff(G,P,s,⟨ m,n⟩) = s(m·n)·(s(m)·s(n))^-1
  using HomDiff˙def by simp
```

The next lemma shows the set from the definition of almost homomorphism in a different form.

```
lemma (in group0) Group˙ZF˙3˙2˙L1A: shows
  –HomDiff(G,P,s,x). x ∈ G×G ″ = –s(m·n)·(s(m)·s(n))^-1. ⟨ m,n⟩ ∈ G×G″
proof -
  have ∀m∈G.∀n∈G. HomDiff(G,P,s,⟨ m,n⟩) = s(m·n)·(s(m)·s(n))^-1
    using HomDiff˙notation by simp
  then show thesis by (rule ZF1˙1˙L4A)
qed
```

Let's define some notation. We inherit the notation and assumptions from the group0 context (locale) and add some. We will use AH to denote the set of almost homomorphisms. $\sim$ is the inverse (negative if the group is the group of integers) of almost homomorphisms, $(\sim p)(n) = p(n)^{-1}$. δ will denote the homomorphism difference specific for the group (HomDiff(G, f)). The notation $s \approx r$ will mean that s, r are almost equal, that is they are in the equivalence relation defined by the group of finite range functions (that is a normal subgroup of almost homomorphisms, if the group is abelian). We show that this is equivalent to the set $\{s(n) \cdot r(n)^{-1} : n \in G\}$ being finite. We also add an assumption that the G is abelian as many needed properties do not hold without that.

```
locale group1 = group0 +
  assumes isAbelian: P –is commutative on″ G

  fixes AH
  defines AH˙def [simp]: AH ≡ AlmostHoms(G,P)

  fixes Op1
  defines Op1˙def [simp]: Op1 ≡ AlHomOp1(G,P)

  fixes Op2
```

```
  defines Op2˙def [simp]: Op2 ≡ AlHomOp2(G,P)

  fixes FR
  defines FR˙def [simp]: FR ≡ FinRangeFunctions(G,G)

  fixes neg (∼˙ [90] 91)
  defines neg˙def [simp]: ∼s ≡ GroupInv(G,P) O s

  fixes δ
  defines δ˙def [simp]: δ(s,x) ≡ HomDiff(G,P,s,x)

  fixes AHprod (infix · 69)
  defines AHprod˙def [simp]: s · r ≡  AlHomOp1(G,P)⟨s,r⟩

  fixes AHcomp (infix ∘ 70)
  defines AHcomp˙def [simp]: s ∘ r ≡  AlHomOp2(G,P)⟨s,r⟩

  fixes AlEq (infix ≈ 68)
  defines AlEq˙def [simp]:
  s ≈ r ≡ ⟨s,r⟩ ∈ QuotientGroupRel(AH,Op1,FR)
```

HomDiff is a homomorphism on the lifted group structure.

```
lemma (in group1) Group˙ZF˙3˙2˙L1:
  assumes A1: s:G→G  r:G→G
  and A2: x ∈ G×G
  and A3: F = P –lifted to function space over˝ G
  shows δ(F⟨ s,r⟩,x) = δ(s,x)·δ(r,x)
proof -
  let p = F⟨ s,r⟩
  from A2 obtain m n where
    D1: x = ⟨ m,n⟩ m∈G n∈G
    by auto
  then have T1:m·n ∈ G
    using group0˙2˙L1 monoid0.group0˙1˙L1 by simp
  with A1 D1 have T2:
    s(m)∈G s(n)∈G r(m)∈G
    r(n)∈G s(m·n)∈G r(m·n)∈G
    using apply˙funtype by auto
  from A3 A1 have T3:p : G→G
    using group0˙2˙L1 monoid0.Group˙ZF˙2˙1˙L0
    by simp
  from D1 T3 have
    δ(p,x) = p(m·n)·((p(n))^-1·(p(m))^-1)
    using HomDiff˙notation apply˙funtype group˙inv˙of˙two
    by simp
  also from A3 A1 D1 T1 isAbelian T2 have
    ... = δ(s,x)· δ(r,x)
    using Group˙ZF˙2˙1˙L3 group0˙4˙L3 HomDiff˙notation
    by simp
```

```
  finally show thesis by simp
qed
```

The group operation lifted to the function space over G preserves almost homomorphisms.

```
lemma (in group1) Group˙ZF˙3˙2˙L2: assumes A1: s ∈ AH r ∈ AH
  and A2: F = P –lifted to function space over″ G
  shows F⟨ s,r⟩ ∈ AH
proof -
  let p = F⟨ s,r⟩
  from A1 A2 have p : G→G
    using AlmostHoms˙def group0˙2˙L1 monoid0.Group˙ZF˙2˙1˙L0
    by simp
  moreover have
    –δ(p,x). x ∈ G×G″ ∈ Fin(G)
  proof -
    from A1 have
      –δ(s,x). x ∈ G×G ″ ∈ Fin(G)
      –δ(r,x). x ∈ G×G ″ ∈ Fin(G)
      using AlmostHoms˙def by auto
    with groupAssum A1 A2 show thesis
      using IsAgroup˙def IsAmonoid˙def IsAssociative˙def
      Finite1˙L15 AlmostHoms˙def Group˙ZF˙3˙2˙L1
      by auto
  qed
  ultimately show thesis using AlmostHoms˙def
    by simp
qed
```

The set of almost homomorphisms is closed under the lifted group operation.

```
lemma (in group1) Group˙ZF˙3˙2˙L3:
  assumes F = P –lifted to function space over″ G
  shows AH –is closed under″ F
  using assms IsOpClosed˙def Group˙ZF˙3˙2˙L2 by simp
```

The terms in the homomorphism difference for a function are in the group.

```
lemma (in group1) Group˙ZF˙3˙2˙L4:
  assumes s:G→G and m∈G  n∈G
  shows
  m·n ∈ G
  s(m·n) ∈ G
  s(m) ∈ G s(n) ∈ G
  δ(s,⟨ m,n⟩) ∈ G
  s(m)·s(n) ∈ G
  using assms group˙op˙closed inverse˙in˙group
    apply˙funtype HomDiff˙def by auto
```

It is handy to have a version of Group˙ZF˙3˙2˙L4 specifically for almost homomorphisms.

```
corollary (in group1) Group˙ZF˙3˙2˙L4A:
  assumes s ∈ AH and m∈G  n∈G
  shows m·n ∈ G
  s(m·n) ∈ G
  s(m) ∈ G s(n) ∈ G
  δ(s,⟨ m,n⟩) ∈ G
  s(m)·s(n) ∈ G
  using assms AlmostHoms˙def Group˙ZF˙3˙2˙L4
  by auto
```

The terms in the homomorphism difference are in the group, a different form.

```
lemma (in group1) Group˙ZF˙3˙2˙L4B:
  assumes A1:s ∈ AH and A2:x∈G×G
  shows fst(x)·snd(x) ∈ G
  s(fst(x)·snd(x)) ∈ G
  s(fst(x)) ∈ G s(snd(x)) ∈ G
  δ(s,x) ∈ G
  s(fst(x))·s(snd(x)) ∈ G
proof -
  let m = fst(x)
  let n = snd(x)
  from A1 A2 show
    m·n ∈ G  s(m·n) ∈ G
    s(m) ∈ G s(n) ∈ G
    s(m)·s(n) ∈ G
    using Group˙ZF˙3˙2˙L4A
    by auto
  from A1 A2 have δ(s,⟨ m,n⟩) ∈ G using Group˙ZF˙3˙2˙L4A
    by simp
  moreover from A2 have ⟨ m,n⟩ = x by auto
  ultimately show δ(s,x) ∈ G by simp
qed
```

What are the values of the inverse of an almost homomorphism?

```
lemma (in group1) Group˙ZF˙3˙2˙L5:
  assumes s ∈ AH and n∈G
  shows (∼s)(n) = (s(n))^-1
  using assms AlmostHoms˙def comp˙fun˙apply by auto
```

Homomorphism difference commutes with the inverse for almost homomorphisms.

```
lemma (in group1) Group˙ZF˙3˙2˙L6:
  assumes A1:s ∈ AH and A2:x∈G×G
  shows δ(∼s,x) = (δ(s,x))^-1
proof -
  let m = fst(x)
  let n = snd(x)
```

```
  have δ(∼s,x) = (∼s)(m·n)·((∼s)(m)·(∼s)(n))^-1
    using HomDiff˙def by simp
  from A1 A2 isAbelian show thesis
    using Group˙ZF˙3˙2˙L4B HomDiff˙def
      Group˙ZF˙3˙2˙L5 group0˙4˙L4A
    by simp
qed
```

The inverse of an almost homomorphism maps the group into itself.

```
lemma (in group1) Group˙ZF˙3˙2˙L7:
  assumes s ∈ AH
  shows ∼s : G→G
  using groupAssum assms AlmostHoms˙def group0˙2˙T2 comp˙fun by auto
```

The inverse of an almost homomorphism is an almost homomorphism.

```
lemma (in group1) Group˙ZF˙3˙2˙L8:
  assumes A1: F = P –lifted to function space over″ G
  and A2: s ∈ AH
  shows GroupInv(G→G,F)(s) ∈ AH
proof -
  from A2 have –δ(s,x). x ∈ G×G″ ∈ Fin(G)
    using AlmostHoms˙def by simp
  with groupAssum  have
    GroupInv(G,P)–δ(s,x). x ∈ G×G″ ∈ Fin(G)
    using group0˙2˙T2 Finite1˙L6A by blast
  moreover have
     GroupInv(G,P)–δ(s,x). x ∈ G×G″ =
    –(δ(s,x))^-1. x ∈ G×G″
  proof -
    from groupAssum have
      GroupInv(G,P) : G→G
      using group0˙2˙T2 by simp
    moreover from A2 have
      ∀x∈G×G. δ(s,x)∈G
      using Group˙ZF˙3˙2˙L4B by simp
    ultimately show thesis
      using func1˙1˙L17 by simp
  qed
  ultimately have –(δ(s,x))^-1. x ∈ G×G″ ∈ Fin(G)
    by simp
  moreover from A2 have
    –(δ(s,x))^-1. x ∈ G×G″ = –δ(∼s,x). x ∈ G×G″
    using Group˙ZF˙3˙2˙L6 by simp
  ultimately have –δ(∼s,x). x ∈ G×G″ ∈ Fin(G)
    by simp
  with A2 groupAssum A1 show thesis
    using Group˙ZF˙3˙2˙L7 AlmostHoms˙def Group˙ZF˙2˙1˙L6
    by simp
qed
```

The function that assigns the neutral element everywhere is an almost homomorphism.

```
lemma (in group1) Group˙ZF˙3˙2˙L9: shows
  ConstantFunction(G,1) ∈ AH and AH≠0
proof -
  let z = ConstantFunction(G,1)
  have G×G≠0 using group0˙2˙L1 monoid0.group0˙1˙L3A
    by blast
  moreover have ∀x∈G×G. δ(z,x) = 1
  proof
    fix x assume A1:x ∈ G × G
    then obtain m n where x = ⟨ m,n⟩ m∈G n∈G
      by auto
    then show δ(z,x) = 1
      using group0˙2˙L1 monoid0.group0˙1˙L1
 func1˙3˙L2 HomDiff˙def group0˙2˙L2
 group˙inv˙of˙one by simp
  qed
  ultimately have −δ(z,x). x∈G×G″ = −1″ by (rule ZF1˙1˙L5)
  then show z ∈ AH using group0˙2˙L2 Finite1˙L16
    func1˙3˙L1 group0˙2˙L2 AlmostHoms˙def by simp
  then show AH≠0 by auto
qed
```

If the group is abelian, then almost homomorphisms form a subgroup of the lifted group.

```
lemma Group˙ZF˙3˙2˙L10:
  assumes A1: IsAgroup(G,P)
  and A2: P −is commutative on″ G
  and A3: F = P −lifted to function space over″ G
  shows IsAsubgroup(AlmostHoms(G,P),F)
proof -
  let AH = AlmostHoms(G,P)
  from A2 A1 have T1: group1(G,P)
    using group1˙axioms.intro group0˙def group1˙def
    by simp
  from A1 A3 have group0(G→G,F)
    using group0˙def group0.Group˙ZF˙2˙1˙T2 by simp
  moreover from T1 have AH≠0
    using group1.Group˙ZF˙3˙2˙L9 by simp
  moreover have T2:AH ⊆ G→G
    using AlmostHoms˙def by auto
  moreover from T1 A3 have
    AH −is closed under″ F
    using group1.Group˙ZF˙3˙2˙L3 by simp
  moreover from T1 A3 have
    ∀s∈AH. GroupInv(G→G,F)(s) ∈ AH
    using group1.Group˙ZF˙3˙2˙L8 by simp
  ultimately show IsAsubgroup(AlmostHoms(G,P),F)
```

```
  using group0.group0˙3˙T3 by simp
qed
```

If the group is abelian, then almost homomorphisms form a group with the first operation, hence we can use theorems proven in group0 context aplied to this group.

```
lemma (in group1) Group˙ZF˙3˙2˙L10A:
 shows IsAgroup(AH,Op1) group0(AH,Op1)
  using groupAssum isAbelian Group˙ZF˙3˙2˙L10 IsAsubgroup˙def
    AlHomOp1˙def group0˙def by auto
```

The group of almost homomorphisms is abelian

```
lemma Group˙ZF˙3˙2˙L11: assumes A1: IsAgroup(G,f)
 and A2: f –is commutative on″ G
 shows
 IsAgroup(AlmostHoms(G,f),AlHomOp1(G,f))
 AlHomOp1(G,f) –is commutative on″ AlmostHoms(G,f)
proof-
 let AH = AlmostHoms(G,f)
 let F = f –lifted to function space over″ G
 from A1 A2 have IsAsubgroup(AH,F)
  using Group˙ZF˙3˙2˙L10 by simp
 then show IsAgroup(AH,AlHomOp1(G,f))
  using IsAsubgroup˙def AlHomOp1˙def by simp
 from A1 have F : (G→G)×(G→G)→(G→G)
  using IsAgroup˙def monoid0˙def monoid0.Group˙ZF˙2˙1˙L0A
  by simp
 moreover have AH ⊆ G→G
  using AlmostHoms˙def by auto
 moreover from A1 A2 have
  F –is commutative on″ (G→G)
  using group0˙def group0.Group˙ZF˙2˙1˙L7
  by simp
 ultimately show
  AlHomOp1(G,f)–is commutative on″ AH
  using func˙ZF˙4˙L1 AlHomOp1˙def by simp
qed
```

The first operation on homomorphisms acts in a natural way on its operands.

```
lemma (in group1) Group˙ZF˙3˙2˙L12:
 assumes s∈AH  r∈AH and n∈G
 shows (s•r)(n) = s(n)·r(n)
 using assms AlHomOp1˙def restrict AlmostHoms˙def Group˙ZF˙2˙1˙L3
 by simp
```

What is the group inverse in the group of almost homomorphisms?

```
lemma (in group1) Group˙ZF˙3˙2˙L13:
 assumes A1: s∈AH
```

```
  shows
  GroupInv(AH,Op1)(s) = GroupInv(G,P) O s
  GroupInv(AH,Op1)(s) ∈ AH
  GroupInv(G,P) O s ∈ AH
proof -
  let F = P –lifted to function space over″ G
  from groupAssum isAbelian have IsAsubgroup(AH,F)
    using Group˙ZF˙3˙2˙L10 by simp
  with A1 show I: GroupInv(AH,Op1)(s) = GroupInv(G,P) O s
    using AlHomOp1˙def Group˙ZF˙2˙1˙L6A by simp
  from A1 show GroupInv(AH,Op1)(s) ∈ AH
    using Group˙ZF˙3˙2˙L10A group0.inverse˙in˙group by simp
  with I show GroupInv(G,P) O s ∈ AH by simp
qed
```

The group inverse in the group of almost homomorphisms acts in a natural way on its operand.

```
lemma (in group1) Group˙ZF˙3˙2˙L14:
  assumes s∈AH and n∈G
  shows (GroupInv(AH,Op1)(s))(n) = (s(n))⁻¹
  using isAbelian assms Group˙ZF˙3˙2˙L13 AlmostHoms˙def comp˙fun˙apply
  by auto
```

The next lemma states that if s, r are almost homomorphisms, then $s \cdot r^{-1}$ is also an almost homomorphism.

```
lemma Group˙ZF˙3˙2˙L15: assumes IsAgroup(G,f)
  and f –is commutative on″ G
  and AH = AlmostHoms(G,f) Op1 = AlHomOp1(G,f)
  and s ∈ AH  r ∈ AH
  shows
  Op1⟨ s,r⟩ ∈ AH
  GroupInv(AH,Op1)(r) ∈ AH
  Op1⟨ s,GroupInv(AH,Op1)(r)⟩ ∈ AH
  using assms group0˙def group1˙axioms.intro group1˙def
      group1.Group˙ZF˙3˙2˙L10A group0.group0˙2˙L1
      monoid0.group0˙1˙L1 group0.inverse˙in˙group by auto
```

A version of Group˙ZF˙3˙2˙L15 formulated in notation used in group1 context. States that the product of almost homomorphisms is an almost homomorphism and the the product of an almost homomorphism with a (pointwise) inverse of an almost homomorphism is an almost homomorphism.

```
corollary (in group1) Group˙ZF˙3˙2˙L16: assumes s ∈ AH  r ∈ AH
  shows s•r ∈ AH  s•(∼r) ∈ AH
  using assms isAbelian group0˙def group1˙axioms group1˙def
  Group˙ZF˙3˙2˙L15 Group˙ZF˙3˙2˙L13 by auto
```

31.3 The classes of almost homomorphisms

In the Real˙ZF series we define real numbers as a quotient of the group of integer almost homomorphisms by the integer finite range functions. In this section we setup the background for that in the general group context.

Finite range functions are almost homomorphisms.

```
lemma (in group1) Group˙ZF˙3˙3˙L1: shows FR ⊆ AH
proof
  fix s assume A1:s ∈ FR
  then have T1:–s(n). n ∈ G″ ∈ Fin(G)
    –s(fst(x)). x∈G×G″ ∈ Fin(G)
    –s(snd(x)). x∈G×G″ ∈ Fin(G)
    using Finite1˙L18 Finite1˙L6B by auto
  have –s(fst(x)·snd(x)). x ∈ G×G″ ∈ Fin(G)
  proof -
    have ∀x∈G×G. fst(x)·snd(x) ∈ G
      using group0˙2˙L1 monoid0.group0˙1˙L1 by simp
    moreover from T1 have –s(n). n ∈ G″ ∈ Fin(G) by simp
    ultimately show thesis by (rule Finite1˙L6B)
  qed
  moreover have
    –(s(fst(x))·s(snd(x)))^-1. x∈G×G″ ∈ Fin(G)
  proof -
    have ∀g∈G. g^-1 ∈ G using inverse˙in˙group
      by simp
    moreover from T1 have
      –s(fst(x))·s(snd(x)). x∈G×G″ ∈ Fin(G)
      using group˙oper˙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: shows
  QuotientGroupRel(AH,Op1,FR) ⊆ AH × AH and
  equiv(AH,QuotientGroupRel(AH,Op1,FR))
  using groupAssum isAbelian QuotientGroupRel˙def
    Group˙ZF˙3˙3˙L2 Group˙ZF˙3˙2˙L10A group0.Group˙ZF˙2˙4˙L3
  by auto
```

The "almost equal" relation is symmetric.

```
lemma (in group1) Group˙ZF˙3˙3˙L3A: assumes A1: s≈r
```

```
  shows r≈s
proof -
  let R = QuotientGroupRel(AH,Op1,FR)
  from A1 have equiv(AH,R) and ⟨s,r⟩ ∈ R
    using Group˙ZF˙3˙3˙L3 by 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 assms Group˙ZF˙3˙2˙L11
    group0˙def Group˙ZF˙3˙3˙L2 group0.Group˙ZF˙2˙4˙L5E
  by simp
```

The group inverse of a class of an almost homomorphism f is the class of the inverse of f.

```
lemma (in group1) Group˙ZF˙3˙3˙L6:
  assumes A1: s ∈ AH  and
  r = QuotientGroupRel(AH,Op1,FR) and
  F = ProjFun2(AH,r,Op1)
  shows r–∼s″ = GroupInv(AH//r,F)(r–s″)
proof -
  from groupAssum isAbelian assms have
    r–GroupInv(AH, Op1)(s)″ = GroupInv(AH//r,F)(r  –s″)
    using Group˙ZF˙3˙2˙L10A Group˙ZF˙3˙3˙L2 QuotientGroupOp˙def
      group0.Group˙ZF˙2˙4˙L7 by simp
  with A1 show thesis using Group˙ZF˙3˙2˙L13
    by simp
qed
```

31.4 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

series. In particular we show that the set of almost homomorphisms is closed under composition and that composition is congruent with respect to the equivalence relation defined by the group of finite range functions (a normal subgroup of almost homomorphisms).

The next formula restates the definition of the homomorphism difference to express the value an almost homomorphism on a product.

```
lemma (in group1) Group˙ZF˙3˙4˙L1:
  assumes s∈AH and  m∈G  n∈G
  shows s(m·n) = s(m)·s(n)·δ(s,⟨ m,n⟩)
  using isAbelian assms Group˙ZF˙3˙2˙L4A HomDiff˙def group0˙4˙L5
  by simp
```

What is the value of a composition of almost homomorhisms?

```
lemma (in group1) Group˙ZF˙3˙4˙L2:
  assumes s∈AH  r∈AH and m∈G
  shows (s∘r)(m) = s(r(m))  s(r(m)) ∈ G
  using assms AlmostHoms˙def func˙ZF˙5˙L3 restrict AlHomOp2˙def
    apply˙funtype by auto
```

What is the homomorphism difference of a composition?

```
lemma (in group1) Group˙ZF˙3˙4˙L3:
  assumes A1: s∈AH  r∈AH and A2: m∈G  n∈G
  shows δ(s∘r,⟨ m,n⟩) =
  δ(s,⟨ r(m),r(n)⟩)·s(δ(r,⟨ m,n⟩))·δ(s,⟨ r(m)·r(n),δ(r,⟨ m,n⟩)⟩)
proof -
  from A1 A2 have T1:
    s(r(m))· s(r(n)) ∈ G
    δ(s,⟨ r(m),r(n)⟩)∈ G s(δ(r,⟨ m,n⟩)) ∈G
    δ(s,⟨ (r(m)·r(n)),δ(r,⟨ m,n⟩)⟩) ∈ G
    using Group˙ZF˙3˙4˙L2 AlmostHoms˙def apply˙funtype
      Group˙ZF˙3˙2˙L4A group0˙2˙L1 monoid0.group0˙1˙L1
    by auto
  from A1 A2 have δ(s∘r,⟨ m,n⟩) =
    s(r(m)·r(n)·δ(r,⟨ m,n⟩))·(s((r(m)))·s(r(n)))^-1
    using HomDiff˙def group0˙2˙L1 monoid0.group0˙1˙L1 Group˙ZF˙3˙4˙L2
      Group˙ZF˙3˙4˙L1 by simp
  moreover from A1 A2 have
    s(r(m)·r(n)·δ(r,⟨ m,n⟩)) =
    s(r(m)·r(n))·s(δ(r,⟨ m,n⟩))·δ(s,⟨ (r(m)·r(n)),δ(r,⟨ m,n⟩)⟩)
    s(r(m)·r(n)) = s(r(m))·s(r(n))·δ(s,⟨ r(m),r(n)⟩)
    using Group˙ZF˙3˙2˙L4A Group˙ZF˙3˙4˙L1 by auto
  moreover from T1 isAbelian have
    s(r(m))·s(r(n))·δ(s,⟨ r(m),r(n)⟩)·
    s(δ(r,⟨ m,n⟩))·δ(s,⟨ (r(m)·r(n)),δ(r,⟨ m,n⟩)⟩)·
    (s((r(m)))·s(r(n)))^-1 =
    δ(s,⟨ r(m),r(n)⟩)·s(δ(r,⟨ m,n⟩))·δ(s,⟨ (r(m)·r(n)),δ(r,⟨ m,n⟩)⟩)
    using group0˙4˙L6C by simp
```

```
  ultimately show thesis by simp
qed
```

What is the homomorphism difference of a composition (another form)? Here we split the homomorphism difference of a composition into a product of three factors. This will help us in proving that the range of homomorphism difference for the composition is finite, as each factor has finite range.

```
lemma (in group1) Group˙ZF˙3˙4˙L4:
  assumes A1: s∈AH  r∈AH and A2: x ∈ G×G
  and A3:
  A = δ(s,⟨ r(fst(x)),r(snd(x))⟩)
  B = s(δ(r,x))
  C = δ(s,⟨ (r(fst(x))·r(snd(x))),δ(r,x)⟩)
  shows δ(s∘r,x) = A·B·C
proof -
  let m = fst(x)
  let n = snd(x)
  note A1
  moreover from A2 have m∈G n∈G
    by auto
  ultimately have
    δ(s∘r,⟨ m,n⟩) =
    δ(s,⟨ r(m),r(n)⟩)·s(δ(r,⟨ m,n⟩))·
    δ(s,⟨ (r(m)·r(n)),δ(r,⟨ m,n⟩)⟩)
    by (rule Group˙ZF˙3˙4˙L3)
  with A1 A2 A3 show thesis
    by auto
qed
```

The range of the homomorphism difference of a composition of two almost homomorphisms is finite. This is the essential condition to show that a composition of almost homomorphisms is an almost homomorphism.

```
lemma (in group1) Group˙ZF˙3˙4˙L5:
  assumes A1: s∈AH  r∈AH
  shows −δ(Composition(G)⟨ s,r⟩,x). x ∈ G×G″ ∈ Fin(G)
proof -
  from A1 have
    ∀x∈G×G. ⟨ r(fst(x)),r(snd(x))⟩ ∈ G×G
    using Group˙ZF˙3˙2˙L4B by simp
  moreover from A1 have
    −δ(s,x). x∈G×G″ ∈ Fin(G)
    using AlmostHoms˙def by simp
  ultimately have
    −δ(s,⟨ r(fst(x)),r(snd(x))⟩). x∈G×G″ ∈ Fin(G)
    by (rule Finite1˙L6B)
  moreover have −s(δ(r,x)). x∈G×G″ ∈ Fin(G)
  proof -
    from A1 have ∀m∈G. s(m) ∈ G
```

```
    using AlmostHoms˙def apply˙funtype by auto
  moreover from A1 have −δ(r,x). x∈G×G″ ∈ Fin(G)
    using AlmostHoms˙def by simp
  ultimately show thesis
    by (rule Finite1˙L6C)
qed
ultimately have
  −δ(s,⟨ r(fst(x)),r(snd(x))⟩)·s(δ(r,x)). x∈G×G″ ∈ Fin(G)
  using group˙oper˙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,P) –is associative on″ AH
proof -
  show AH –is closed under″ Composition(G)
    using Group˙ZF˙3˙4˙T1 IsOpClosed˙def by simp
  moreover have AH ⊆ G→G using AlmostHoms˙def
    by auto
  moreover have
    Composition(G) –is associative on″ (G→G)
    using func˙ZF˙5˙L5 by simp
  ultimately show AlHomOp2(G,P) –is associative on″ AH
    using func˙ZF˙4˙L3 AlHomOp2˙def by simp
qed
```

Type information related to the situation of two almost homomorphisms.

```
lemma (in group1) Group˙ZF˙3˙4˙L7:
  assumes A1: s∈AH  r∈AH and A2: n∈G
  shows
  s(n) ∈ G (r(n))^-1 ∈ G
  s(n)·(r(n))^-1 ∈ G   s(r(n)) ∈ G
proof -
  from A1 A2 show
    s(n) ∈ G
    (r(n))^-1 ∈ G
    s(r(n)) ∈ G
    s(n)·(r(n))^-1 ∈ G
    using AlmostHoms˙def apply˙type
      group0˙2˙L1 monoid0.group0˙1˙L1 inverse˙in˙group
    by auto
qed
```

Type information related to the situation of three almost homomorphisms.

```
lemma (in group1) Group˙ZF˙3˙4˙L8:
  assumes A1: s∈AH  r∈AH  q∈AH and A2: n∈G
  shows
  q(n)∈G
  s(r(n)) ∈ G
  r(n)·(q(n))^-1 ∈ G
  s(r(n)·(q(n))^-1) ∈ G
  δ(s,⟨ q(n),r(n)·(q(n))^-1⟩) ∈ G
proof -
  from A1 A2 show
    q(n)∈ G  s(r(n)) ∈ G r(n)·(q(n))^-1 ∈ G
    using AlmostHoms˙def apply˙type
      group0˙2˙L1 monoid0.group0˙1˙L1 inverse˙in˙group
    by auto
  with A1 A2 show s(r(n)·(q(n))^-1) ∈ G
```

δ(s,⟨ q(n),r(n)·(q(n))$^{-1}$⟩) ∈ G
using AlmostHoms˙def apply˙type Group˙ZF˙3˙2˙L4A
by auto
qed

A formula useful in showing that the composition of almost homomorphisms is congruent with respect to the quotient group relation.

lemma (in group1) Group˙ZF˙3˙4˙L9:
assumes A1: s1 ∈ AH r1 ∈ AH s2 ∈ AH r2 ∈ AH
and A2: n∈G
shows (s1∘r1)(n)·((s2∘r2)(n))$^{-1}$ =
s1(r2(n))· (s2(r2(n)))$^{-1}$·s1(r1(n)·(r2(n))$^{-1}$)·
δ(s1,⟨ r2(n),r1(n)·(r2(n))$^{-1}$⟩)
proof -
from A1 A2 isAbelian have
(s1∘r1)(n)·((s2∘r2)(n))$^{-1}$ =
s1(r2(n)·(r1(n)·(r2(n))$^{-1}$))·(s2(r2(n)))$^{-1}$
using Group˙ZF˙3˙4˙L2 Group˙ZF˙3˙4˙L7 group0˙4˙L6A
group˙oper˙assoc by simp
with A1 A2 have (s1∘r1)(n)·((s2∘r2)(n))$^{-1}$ = s1(r2(n))·
s1(r1(n)·(r2(n))$^{-1}$)·δ(s1,⟨ r2(n),r1(n)·(r2(n))$^{-1}$⟩)·
(s2(r2(n)))$^{-1}$
using Group˙ZF˙3˙4˙L8 Group˙ZF˙3˙4˙L1 by simp
with A1 A2 isAbelian show thesis using
Group˙ZF˙3˙4˙L8 group0˙4˙L7 by simp
qed

The next lemma shows a formula that translates an expression in terms of the first group operation on almost homomorphisms and the group inverse in the group of almost homomorphisms to an expression using only the underlying group operations.

lemma (in group1) Group˙ZF˙3˙4˙L10: assumes A1: s ∈ AH r ∈ AH
and A2: n ∈ G
shows (s•(GroupInv(AH,Op1)(r)))(n) = s(n)·(r(n))$^{-1}$
proof -
from A1 A2 show thesis
using isAbelian Group˙ZF˙3˙2˙L13 Group˙ZF˙3˙2˙L12 Group˙ZF˙3˙2˙L14
by simp
qed

A neccessary condition for two a. h. to be almost equal.

lemma (in group1) Group˙ZF˙3˙4˙L11:
assumes A1: s≈r
shows –s(n)·(r(n))$^{-1}$. n∈G″ ∈ Fin(G)
proof -
from A1 have s∈AH r∈AH
using QuotientGroupRel˙def by auto
moreover from A1 have

```
    –(s•(GroupInv(AH,Op1)(r)))(n). n∈G″ ∈ Fin(G)
    using QuotientGroupRel˙def Finite1˙L18 by simp
  ultimately show thesis
    using Group˙ZF˙3˙4˙L10 by simp
qed
```

A sufficient condition for two a. h. to be almost equal.

```
lemma (in group1) Group˙ZF˙3˙4˙L12: assumes A1: s∈AH  r∈AH
  and A2: –s(n)·(r(n))^-1. n∈G″ ∈ Fin(G)
  shows s≈r
proof -
  from groupAssum isAbelian A1 A2 show thesis
    using Group˙ZF˙3˙2˙L15 AlmostHoms˙def
    Group˙ZF˙3˙4˙L10 Finite1˙L19 QuotientGroupRel˙def
    by simp
qed
```

Another sufficient consdition for two a.h. to be almost equal. It is actually just an expansion of the definition of the quotient group relation.

```
lemma (in group1) Group˙ZF˙3˙4˙L12A: assumes s∈AH  r∈AH
  and s•(GroupInv(AH,Op1)(r)) ∈ FR
  shows s≈r  r≈s
proof  -
  from assms show s≈r using assms QuotientGroupRel˙def
    by simp
  then show r≈s by (rule Group˙ZF˙3˙3˙L3A)
qed
```

Another necessary condition for two a.h. to be almost equal. It is actually just an expansion of the definition of the quotient group relation.

```
lemma (in group1) Group˙ZF˙3˙4˙L12B: assumes s≈r
  shows s•(GroupInv(AH,Op1)(r)) ∈ FR
  using assms QuotientGroupRel˙def by simp
```

The next lemma states the essential condition for the composition of a. h. to be congruent with respect to the quotient group relation for the subgroup of finite range functions.

```
lemma (in group1) Group˙ZF˙3˙4˙L13:
  assumes A1: s1≈s2  r1≈r2
  shows (s1∘r1) ≈ (s2∘r2)
proof -
  have –s1(r2(n))· (s2(r2(n)))^-1. n∈G″ ∈ Fin(G)
  proof -
    from A1 have ∀n∈G. r2(n) ∈ G
      using QuotientGroupRel˙def AlmostHoms˙def apply˙funtype
      by auto
    moreover from A1 have –s1(n)·(s2(n))^-1. n∈G″ ∈ Fin(G)
      using Group˙ZF˙3˙4˙L11 by simp
```

```
    ultimately show thesis by (rule Finite1˙L6B)
  qed
  moreover have –s1(r1(n)·(r2(n))^-1). n ∈ G″ ∈ Fin(G)
  proof -
    from A1 have ∀n∈G. s1(n)∈G
      using QuotientGroupRel˙def AlmostHoms˙def apply˙funtype
      by auto
    moreover from A1 have –r1(n)·(r2(n))^-1. n∈G″ ∈ Fin(G)
      using Group˙ZF˙3˙4˙L11 by simp
    ultimately show thesis by (rule Finite1˙L6C)
  qed
  ultimately have
    –s1(r2(n))· (s2(r2(n)))^-1·s1(r1(n)·(r2(n))^-1).
    n∈G″ ∈ Fin(G)
    using group˙oper˙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: shows
  Congruent2(QuotientGroupRel(AH,Op1,FR),Op2)
proof -
  show thesis using Group˙ZF˙3˙4˙L13 Congruent2˙def
    by simp
qed
```

The homomorphism difference for the identity function is equal to the neutral element of the group (denoted e in the group1 context).

```
lemma (in group1) Group˙ZF˙3˙4˙L14: assumes A1: x ∈ G×G
  shows δ(id(G),x) = 1
proof -
  from A1 show thesis using
    group0˙2˙L1 monoid0.group0˙1˙L1 HomDiff˙def id˙conv group0˙2˙L6
    by simp
qed
```

The identity function ($I(x) = x$) on G is an almost homomorphism.

```
lemma (in group1) Group˙ZF˙3˙4˙L15: shows id(G) ∈ AH
proof -
  have G×G ≠ 0 using group0˙2˙L1 monoid0.group0˙1˙L3A
    by blast
  then show thesis using Group˙ZF˙3˙4˙L14 group0˙2˙L2
    id˙type AlmostHoms˙def by simp
qed
```

Almost homomorphisms form a monoid with composition. The identity function on the group is the neutral element there.

```
lemma (in group1) Group˙ZF˙3˙4˙L16:
  shows
  IsAmonoid(AH,Op2)
  monoid0(AH,Op2)
  id(G) = TheNeutralElement(AH,Op2)
proof-
  let i = TheNeutralElement(G→G,Composition(G))
  have
    IsAmonoid(G→G,Composition(G))
    monoid0(G→G,Composition(G))
    using monoid0˙def Group˙ZF˙2˙5˙L2 by auto
  moreover have AH –is closed under″ Composition(G)
    using Group˙ZF˙3˙4˙L6 by simp
  moreover have AH ⊆ G→G
    using AlmostHoms˙def by auto
  moreover have i ∈ AH
    using Group˙ZF˙2˙5˙L2 Group˙ZF˙3˙4˙L15 by simp
  moreover have id(G) =  i
    using Group˙ZF˙2˙5˙L2 by simp
  ultimately show
    IsAmonoid(AH,Op2)
    monoid0(AH,Op2)
    id(G) = TheNeutralElement(AH,Op2)
    using monoid0.group0˙1˙T1 group0˙1˙L6 AlHomOp2˙def monoid0˙def
    by auto
qed
```

We can project the monoid of almost homomorphisms with composition to the group of almost homomorphisms divided by the subgroup of finite range functions. The class of the identity function is the neutral element of the

quotient (monoid).

```
theorem (in group1) Group˙ZF˙3˙4˙T2:
  assumes A1: R = QuotientGroupRel(AH,Op1,FR)
  shows
  IsAmonoid(AH//R,ProjFun2(AH,R,Op2))
  R–id(G)″ = TheNeutralElement(AH//R,ProjFun2(AH,R,Op2))
proof -
  have group0(AH,Op1) using Group˙ZF˙3˙2˙L10A group0˙def
    by simp
  with A1 groupAssum isAbelian show
    IsAmonoid(AH//R,ProjFun2(AH,R,Op2))
    R–id(G)″ = TheNeutralElement(AH//R,ProjFun2(AH,R,Op2))
    using Group˙ZF˙3˙3˙L2 group0.Group˙ZF˙2˙4˙L3 Group˙ZF˙3˙4˙L13A
      Group˙ZF˙3˙4˙L16 monoid0.Group˙ZF˙2˙2˙T1 Group˙ZF˙2˙2˙L1
    by auto
qed
```

31.5 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
```

32 Direct product

theory DirectProduct˙ZF imports func˙ZF

begin

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

32.1 Definition

In group theory the notion of direct product provides a natural way of creating a new group from two given groups.

Given $(G, \cdot)$ and $(H, \circ)$ a new operation $(G \times H, \times)$ is defined as $(g, h) \times (g', h') = (g \cdot g', h \circ h')$.

```
definition
  DirectProduct(P,Q,G,H) ≡
  –⟨x,⟨P⟨fst(fst(x)),fst(snd(x))⟩ , Q⟨snd(fst(x)),snd(snd(x))⟩⟩⟩.
  x ∈ (G×H)×(G×H)″
```

We define a context called direct0 which holds an assumption that P, Q are binary operations on G, H, resp. and denotes R as the direct product of (G, P) and (H, Q).

```
locale direct0 =
  fixes P Q G H
  assumes Pfun: P : G×G→G
  assumes Qfun: Q : H×H→H
  fixes R
  defines Rdef [simp]: R ≡ DirectProduct(P,Q,G,H)
```

The direct product of binary operations is a binary operation.

```
lemma (in direct0) DirectProduct˙ZF˙1˙L1:
  shows R : (G×H)×(G×H)→G×H
proof -
  from Pfun Qfun have ∀x∈(G×H)×(G×H).
    ⟨P⟨fst(fst(x)),fst(snd(x))⟩,Q⟨snd(fst(x)),snd(snd(x))⟩⟩ ∈ G×H
    by auto
  then show thesis using ZF˙fun˙from˙total DirectProduct˙def
    by simp
qed
```

And it has the intended value.

```
lemma (in direct0) DirectProduct˙ZF˙1˙L2:
  shows ∀x∈(G×H). ∀y∈(G×H).
  R⟨x,y⟩ = ⟨P⟨fst(x),fst(y)⟩,Q⟨snd(x),snd(y)⟩⟩
  using DirectProduct˙def DirectProduct˙ZF˙1˙L1 ZF˙fun˙from˙tot˙val
  by simp
```

And the value belongs to the set the operation is defined on.

```
lemma (in direct0) DirectProduct˙ZF˙1˙L3:
  shows ∀x∈(G×H). ∀y∈(G×H). R⟨x,y⟩ ∈ G×H
  using DirectProduct˙ZF˙1˙L1 by simp
```

32.2 Associative and commutative operations

If P and Q are both associative or commutative operations, the direct product of P and Q has the same property.

Direct product of commutative operations is commutative.

```
lemma (in direct0) DirectProduct˙ZF˙2˙L1:
  assumes P –is commutative on″ G and Q –is commutative on″ H
  shows R –is commutative on″ G×H
proof -
  from assms have ∀x∈(G×H). ∀y∈(G×H). R⟨x,y⟩ = R⟨y,x⟩
    using DirectProduct˙ZF˙1˙L2 IsCommutative˙def by simp
  then show thesis using IsCommutative˙def by simp
qed
```

Direct product of associative operations is associative.

```
lemma (in direct0) DirectProduct˙ZF˙2˙L2:
  assumes P –is associative on″ G and Q –is associative on″ H
```

```
  shows R –is associative on″ G×H
proof -
  have ∀x∈G×H. ∀y∈G×H. ∀z∈G×H. R⟨R⟨x,y⟩,z⟩ =
    ⟨P⟨P⟨fst(x),fst(y)⟩,fst(z)⟩,Q⟨Q⟨snd(x),snd(y)⟩,snd(z)⟩⟩
    using DirectProduct˙ZF˙1˙L2 DirectProduct˙ZF˙1˙L3
    by auto
  moreover have ∀x∈G×H. ∀y∈G×H. ∀z∈G×H. R⟨x,R⟨y,z⟩⟩ =
    ⟨P⟨fst(x),P⟨fst(y),fst(z)⟩⟩,Q⟨snd(x),Q⟨snd(y),snd(z)⟩⟩⟩
    using DirectProduct˙ZF˙1˙L2 DirectProduct˙ZF˙1˙L3 by auto
  ultimately have ∀x∈G×H. ∀y∈G×H. ∀z∈G×H. R⟨R⟨x,y⟩,z⟩ = R⟨x,R⟨y,z⟩⟩
    using assms IsAssociative˙def by simp
  then show thesis
    using DirectProduct˙ZF˙1˙L1 IsAssociative˙def by simp
qed

end
```

33 Ordered groups - introduction

theory OrderedGroup˙ZF imports Group˙ZF˙1 AbelianGroup˙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˙IML.thy that subsets of integers are bounded iff they are finite.

33.1 Ordered groups

This section defines ordered groups and various related notions.

An ordered group is a group equipped with a partial order that is "translation invariant", that is if $a \leq b$ then $a \cdot g \leq b \cdot g$ and $g \cdot a \leq g \cdot b$.

```
definition
  IsAnOrdGroup(G,P,r) ≡
  (IsAgroup(G,P) ∧ r⊆G×G ∧ IsPartOrder(G,r) ∧ (∀g∈G. ∀a b.
  ⟨ a,b⟩ ∈ r ⟶ ⟨ P⟨ a,g⟩,P⟨ b,g⟩ ⟩ ∈ r ∧ ⟨ P⟨ g,a⟩,P⟨ g,b⟩ ⟩ ∈ r ) )
```

We define the set of nonnegative elements in the obvious way as $G^+ = \{x \in G : 1 \leq x\}$.

```
definition
  Nonnegative(G,P,r) ≡ –x∈G. ⟨ TheNeutralElement(G,P),x⟩ ∈ r″
```

The PositiveSet(G,P,r) is a set similar to Nonnegative(G,P,r), but without the unit.

definition
PositiveSet(G,P,r) ≡
–x∈G. ⟨ TheNeutralElement(G,P),x⟩ ∈ r ∧ TheNeutralElement(G,P)≠ x″

We also define the absolute value as a ZF-function that is the identity on G^+ and the group inverse on the rest of the group.

definition
AbsoluteValue(G,P,r) ≡ id(Nonnegative(G,P,r)) ∪
restrict(GroupInv(G,P),G - Nonnegative(G,P,r))

The odd functions are defined as those having property $f(a^{-1}) = (f(a))^{-1}$. This looks a bit strange in the multiplicative notation, I have to admit. For linearly 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

definition
OddExtension(G,P,r,f) ≡
(f ∪ –⟨a, GroupInv(G,P)(f(GroupInv(G,P)(a)))⟩.
a ∈ GroupInv(G,P)(PositiveSet(G,P,r))″ ∪
–⟨TheNeutralElement(G,P),TheNeutralElement(G,P)⟩″)

We will use a similar notation for ordered groups as for the generic groups. G^+ denotes the set of nonnegative elements (that satisfy $1 \leq a$) and G_+ is the set of (strictly) positive elements. -A is the set inverses of elements from A. I hope that using additive notation for this notion is not too shocking here. The symbol f° denotes the odd extension of f. For a function defined on G_+ this is the unique odd function on G that is equal to f on G_+.

locale group3 =

fixes G and P and r

assumes ordGroupAssum: IsAnOrdGroup(G,P,r)

fixes unit (**1**)
defines unit˙def [simp]: **1** ≡ TheNeutralElement(G,P)

fixes groper (infixl · 70)
defines groper˙def [simp]: a · b ≡ P⟨ a,b⟩

fixes inv (˙$^{-1}$ [90] 91)
defines inv˙def [simp]: x^{-1} ≡ GroupInv(G,P)(x)

fixes lesseq (infix ≤ 68)
defines lesseq˙def [simp]: a ≤ b ≡ ⟨ a,b⟩ ∈ r

fixes sless (infix ¡ 68)
defines sless˙def [simp]: a ¡ b ≡ a≤b ∧ a≠b

```
  fixes nonnegative (G⁺)
  defines nonnegative˙def [simp]: G⁺ ≡ Nonnegative(G,P,r)

  fixes positive (G₊)
  defines positive˙def [simp]: G₊ ≡ PositiveSet(G,P,r)

  fixes setinv (- ˙ 72)
  defines setninv˙def [simp]: -A ≡ GroupInv(G,P)(A)

  fixes abs (— ˙ —)
  defines abs˙def [simp]: —a— ≡ AbsoluteValue(G,P,r)(a)

  fixes oddext (˙ °)
  defines oddext˙def [simp]: f° ≡ OddExtension(G,P,r,f)
```

In group3 context we can use the theorems proven in the group0 context.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L1: shows group0(G,P)
  using ordGroupAssum IsAnOrdGroup˙def group0˙def by simp
```

Ordered group (carrier) is not empty. This is a property of monoids, but it is good to have it handy in the group3 context.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L1A: shows G≠0
  using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L1 monoid0.group0˙1˙L3A
  by blast
```

The next lemma is just to see the definition of the nonnegative set in our notation.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L2:
  shows g∈G⁺ ⟷ 1≤g
  using ordGroupAssum IsAnOrdGroup˙def Nonnegative˙def
  by auto
```

The next lemma is just to see the definition of the positive set in our notation.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L2A:
  shows g∈G₊ ⟷ (1≤g ∧ g≠1)
  using ordGroupAssum IsAnOrdGroup˙def PositiveSet˙def
  by auto
```

For total order if g is not in G^+, then it has to be less or equal the unit.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L2B:
  assumes A1: r –is total on″ G and A2: a∈G-G⁺
  shows a≤1
proof -
  from A2 have a∈G   1 ∈ G  ¬(1≤a)
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2 OrderedGroup˙ZF˙1˙L2
    by auto
  with A1 show thesis using IsTotal˙def by auto
```

qed

The group order is reflexive.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L3: assumes g∈G
  shows g≤g
  using ordGroupAssum assms IsAnOrdGroup˙def IsPartOrder˙def refl˙def
  by simp
```

1 is nonnegative.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L3A: shows 1∈G⁺
  using OrderedGroup˙ZF˙1˙L2 OrderedGroup˙ZF˙1˙L3
    OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2 by simp
```

In this context $a \leq b$ implies that both a and b belong to G.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L4:
  assumes a≤b shows a∈G b∈G
  using ordGroupAssum assms IsAnOrdGroup˙def by auto
```

Similarly in this context $a \leq b$ implies that both a and b belong to G.

```
lemma (in group3) less˙are˙members:
  assumes a¡b shows a∈G b∈G
  using ordGroupAssum assms IsAnOrdGroup˙def by auto
```

It is good to have transitivity handy.

```
lemma (in group3) Group˙order˙transitive:
  assumes A1: a≤b  b≤c shows a≤c
proof -
  from ordGroupAssum have trans(r)
    using IsAnOrdGroup˙def IsPartOrder˙def
    by simp
  moreover from A1 have ⟨ a,b⟩ ∈ r ∧ ⟨ b,c⟩ ∈ r by simp
  ultimately have ⟨ a,c⟩ ∈ r by (rule Fol1˙L3)
  thus thesis by simp
qed
```

The order in an ordered group is antisymmetric.

```
lemma (in group3) group˙order˙antisym:
  assumes A1: a≤b  b≤a shows a=b
proof -
  from ordGroupAssum A1 have
    antisym(r)  ⟨ a,b⟩ ∈ r  ⟨ b,a⟩ ∈ r
    using IsAnOrdGroup˙def IsPartOrder˙def by auto
  then show a=b by (rule Fol1˙L4)
qed
```

Transitivity for the strict order: if $a < b$ and $b \leq c$, then $a < c$.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L4A:
  assumes A1: a¡b  and A2: b≤c
```

```
  shows a¡c
proof -
  from A1 A2 have a≤b  b≤c by auto
  then have a≤c by (rule Group˙order˙transitive)
  moreover from A1 A2 have a≠c using group˙order˙antisym by auto
  ultimately show a¡c by simp
qed
```

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¡b and c∈G
  shows
  a·c ¡ b·c
  c·a ¡ c·b
  using ordGroupAssum assms 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 assms ordGroupAssum IsAnOrdGroup˙def Order˙ZF˙1˙L3
  by simp
```

For linearly ordered groups elements in the nonnegative set are greater than those in the complement.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L4B:
  assumes r –is total on″ G
  and a∈G⁺ and b ∈ G-G⁺
  shows b≤a
proof -
  from assms have b≤1 1≤a
    using OrderedGroup˙ZF˙1˙L2 OrderedGroup˙ZF˙1˙L2B by auto
  then show thesis by (rule Group˙order˙transitive)
qed
```

If $a \leq 1$ and $a \neq 1$, then $a \in G \setminus G^+$.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L4C:
  assumes A1: a≤1 and A2: a≠1
  shows a ∈ G-G⁺
proof -
  { assume a ∉ G-G⁺
    with ordGroupAssum A1 A2 have False
      using OrderedGroup˙ZF˙1˙L4 OrderedGroup˙ZF˙1˙L2
 OrderedGroup˙ZF˙1˙L4 IsAnOrdGroup˙def IsPartOrder˙def antisym˙def
      by auto
  } thus thesis by auto
qed
```

An element smaller than an element in $G \setminus G^+$ is in $G \setminus G^+$.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L4D:
  assumes A1: a∈G-G⁺ and A2: b≤a
  shows b∈G-G⁺
proof -
  { assume b ∉ G - G⁺
    with A2 have 1≤b b≤a
      using OrderedGroup˙ZF˙1˙L4 OrderedGroup˙ZF˙1˙L2 by auto
    then have 1≤a by (rule Group˙order˙transitive)
    with A1 have False using OrderedGroup˙ZF˙1˙L2 by simp
  } thus thesis by auto
qed
```

The nonnegative set is contained in the group.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L4E: shows G⁺ ⊆ G
  using OrderedGroup˙ZF˙1˙L2 OrderedGroup˙ZF˙1˙L4 by auto
```

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
```

33.2 Inequalities

This section developes some simple tools to deal with inequalities.

Taking negative on both sides reverses the inequality, case with an inverse on one side.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L5AD:
  assumes A1: b ∈ G and A2: a≤b⁻¹
  shows b ≤ a⁻¹
proof -
  from A2 have (b⁻¹)⁻¹ ≤ a⁻¹
    using OrderedGroup˙ZF˙1˙L5 by simp
  with A1 show b ≤ a⁻¹
    using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
    by simp
qed
```

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

```
lemma (in group3) OrderedGroup˙ZF˙1˙L5AE:
  assumes A1: a∈G  b∈G  c∈G and A2: a·b ≤ a·c
  shows b≤c
proof -
  from ordGroupAssum A1 A2 have a⁻¹·(a·b) ≤ a⁻¹·(a·c)
    using OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group
      IsAnOrdGroup˙def by simp
  with A1 show b≤c
    using OrderedGroup˙ZF˙1˙L1 group0.inv˙cancel˙two
    by simp
qed
```

We can cancel the same element on both sides of an inequality, 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.inv˙cancel˙two
    by simp
  with ordGroupAssum A2 T have c⁻¹·a ≤ c⁻¹·(c·b)
    using IsAnOrdGroup˙def by simp
  with A1 T show c⁻¹·a ≤ b
    using OrderedGroup˙ZF˙1˙L1 group0.inv˙cancel˙two
    by simp
```

qed

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

```
lemma (in group3) OrderedGroup˙ZF˙1˙L5I:
  assumes a≤b and c≤d
  shows a·d⁻¹ ≤ b·c⁻¹
  using assms OrderedGroup˙ZF˙1˙L5 OrderedGroup˙ZF˙1˙L5B
  by simp
```

We can put an element on the other side of an inequality changing its sign, version with the inverse.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L5J:
  assumes A1: a∈G  b∈G and A2: c ≤ a·b⁻¹
  shows c·b ≤ a
proof -
  from ordGroupAssum A1 A2 have c·b ≤ a·b⁻¹·b
    using IsAnOrdGroup˙def by simp
  with A1 show c·b ≤ a
    using OrderedGroup˙ZF˙1˙L1 group0.inv˙cancel˙two
    by simp
qed
```

We can put an element on the other side of an inequality changing its sign, version with the inverse.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L5JA:
  assumes A1: a∈G  b∈G and A2: c ≤ a⁻¹·b
  shows a·c≤ b
proof -
  from ordGroupAssum A1 A2 have a·c ≤ a·(a⁻¹·b)
    using IsAnOrdGroup˙def by simp
  with A1 show a·c≤ b
    using OrderedGroup˙ZF˙1˙L1 group0.inv˙cancel˙two
    by simp
qed
```

A special case of OrderedGroup˙ZF˙1˙L5J where $c = 1$.

```
corollary (in group3) OrderedGroup˙ZF˙1˙L5K:
  assumes A1: a∈G  b∈G and A2: 1 ≤ a·b⁻¹
  shows b ≤ a
proof -
  from A1 A2 have 1·b ≤ a
    using OrderedGroup˙ZF˙1˙L5J by simp
  with A1 show b ≤ a
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2
    by simp
qed
```

A special case of OrderedGroup˙ZF˙1˙L5JA where $c = 1$.

```
corollary (in group3) OrderedGroup˙ZF˙1˙L5KA:
  assumes A1: a∈G  b∈G and A2: 1 ≤ a⁻¹·b
  shows a ≤ b
proof -
  from A1 A2 have a·1 ≤ b
    using OrderedGroup˙ZF˙1˙L5JA by simp
  with A1 show a ≤ b
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2
    by simp
qed
```

If the order is total, the elements that do not belong to the positive set are negative. We also show here that the group inverse of an element that does not belong to the nonnegative set does belong to the nonnegative set.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L6:
  assumes A1: r –is total on″ G and A2: a∈G-G⁺
  shows a≤1  a⁻¹ ∈ G⁺  restrict(GroupInv(G,P),G-G⁺)(a) ∈ G⁺
proof -
  from A2 have T1: a∈G a∉G⁺ 1∈G
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2 by auto
  with A1 show a≤1 using OrderedGroup˙ZF˙1˙L2 IsTotal˙def
    by auto
  then show a⁻¹ ∈ G⁺ using OrderedGroup˙ZF˙1˙L5A OrderedGroup˙ZF˙1˙L2
    by simp
  with A2 show restrict(GroupInv(G,P),G-G⁺)(a) ∈ G⁺
    using restrict by simp
qed
```

If a property is invariant with respect to taking the inverse and it is true on the nonnegative set, than it is true on the whole group.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L7:
  assumes A1: r –is total on″ G
  and A2: ∀a∈G⁺.∀b∈G⁺. Q(a,b)
  and A3: ∀a∈G.∀b∈G. Q(a,b)⟶Q(a⁻¹,b)
  and A4: ∀a∈G.∀b∈G. Q(a,b)⟶Q(a,b⁻¹)
  and A5: a∈G b∈G
  shows Q(a,b)
proof -
  { assume A6: a∈G⁺ have Q(a,b)
    proof -
      { assume b∈G⁺
 with A6 A2 have Q(a,b) by simp }
      moreover
      { assume b∉G⁺
 with A1 A2 A4 A5 A6 have Q(a,(b⁻¹)⁻¹)
   using OrderedGroup˙ZF˙1˙L6 OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group
   by simp
 with A5 have Q(a,b) using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
   by simp }
```

```
      ultimately show Q(a,b) by auto
    qed }
  moreover
  { assume a∉G⁺
    with A1 A5 have T1: a⁻¹ ∈ G⁺ using OrderedGroup˙ZF˙1˙L6 by simp
    have Q(a,b)
    proof -
      { assume b∈G⁺
 with A2 A3 A5 T1 have Q((a⁻¹)⁻¹,b)
   using OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group by simp
 with A5 have Q(a,b) using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
   by simp }
      moreover
      { assume b∉G⁺
 with A1 A2 A3 A4 A5 T1 have Q((a⁻¹)⁻¹,(b⁻¹)⁻¹)
   using OrderedGroup˙ZF˙1˙L6 OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group
   by simp
 with A5 have Q(a,b) using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
   by simp }
      ultimately show Q(a,b) by auto
    qed }
  ultimately show Q(a,b) by auto
qed
```

A lemma about splitting the ordered group "plane" into 6 subsets. Useful for proofs by cases.

```
lemma (in group3) OrdGroup˙6cases: assumes A1: r –is total on″ G
  and A2:  a∈G  b∈G
  shows
  1≤a ∧ 1≤b  ∨  a≤1 ∧ b≤1  ∨
  a≤1 ∧ 1≤b ∧ 1 ≤ a·b  ∨ a≤1 ∧ 1≤b ∧ a·b ≤ 1  ∨
  1≤a ∧ b≤1 ∧ 1 ≤ a·b  ∨  1≤a ∧ b≤1 ∧ a·b ≤ 1
proof -
  from A1 A2 have
    1≤a ∨ a≤1
    1≤b ∨ b≤1
    1 ≤ a·b ∨ a·b ≤ 1
    using OrderedGroup˙ZF˙1˙L1 group0.group˙op˙closed group0.group0˙2˙L2
      IsTotal˙def by auto
  then show thesis by auto
qed
```

The next lemma shows what happens when one element of a totally ordered group is not greater or equal than another.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L8:
  assumes A1: r –is total on″ G
  and A2: a∈G  b∈G
  and A3: ¬(a≤b)
  shows b ≤ a  a⁻¹ ≤ b⁻¹  a≠b  b¡a
```

```
proof -
  from A1 A2 A3 show I: b ≤ a using IsTotal˙def
    by auto
  then show a⁻¹ ≤ b⁻¹ using OrderedGroup˙ZF˙1˙L5 by simp
  from A2 have a ≤ a using OrderedGroup˙ZF˙1˙L3 by simp
  with I A3 show a≠b  b ¡ a by auto
qed
```

If one element is greater or equal and not equal to another, then it is not smaller or equal.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L8AA:
  assumes A1: a≤b and A2: a≠b
  shows ¬(b≤a)
proof -
  { note A1
    moreover assume b≤a
    ultimately have a=b by (rule group˙order˙antisym)
    with A2 have False by simp
  } thus ¬(b≤a) by auto
qed
```

A special case of OrderedGroup˙ZF˙1˙L8 when one of the elements is the unit.

```
corollary (in group3) OrderedGroup˙ZF˙1˙L8A:
  assumes A1: r –is total on″ G
  and A2: a∈G and A3: ¬(1≤a)
  shows 1 ≤ a⁻¹  1≠a  a≤1
proof -
  from A1 A2 A3 have I:
    r –is total on″ G
    1∈G  a∈G
     ¬(1≤a)
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2
    by auto
  then have 1⁻¹ ≤ a⁻¹
    by (rule OrderedGroup˙ZF˙1˙L8)
  then show 1 ≤ a⁻¹
    using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙one by simp
  from I show 1≠a by (rule OrderedGroup˙ZF˙1˙L8)
  from A1 I show a≤1 using IsTotal˙def
    by auto
qed
```

A negative element can not be nonnegative.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L8B:
  assumes A1: a≤1  and A2: a≠1 shows ¬(1≤a)
proof -
  { assume  1≤a
    with A1 have a=1 using group˙order˙antisym
```

```
    by auto
  with A2 have False by simp
 } thus thesis by auto
qed
```

An element is greater or equal than another iff the difference is nonpositive.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L9:
 assumes A1: a∈G  b∈G
 shows a≤b ⟷ a·b⁻¹ ≤ 1
proof
 assume a ≤ b
 with ordGroupAssum A1 have a·b⁻¹ ≤ b·b⁻¹
  using OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group
  IsAnOrdGroup˙def by simp
 with A1 show a·b⁻¹ ≤ 1
  using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L6
  by simp
next assume A2: a·b⁻¹ ≤ 1
 with ordGroupAssum A1 have a·b⁻¹·b ≤ 1·b
  using IsAnOrdGroup˙def by simp
 with A1 show a ≤ b
  using OrderedGroup˙ZF˙1˙L1
   group0.inv˙cancel˙two group0.group0˙2˙L2
  by simp
qed
```

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

```
lemma (in group3) OrderedGroup˙ZF˙1˙L9A:
 assumes A1: a∈G  b∈G  c∈G
 shows a·b ≤ c  ⟷ a ≤ c·b⁻¹
proof
 assume a·b ≤ c
 with ordGroupAssum A1 have a·b·b⁻¹ ≤ c·b⁻¹
  using OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group IsAnOrdGroup˙def
  by simp
 with A1 show a ≤ c·b⁻¹
  using OrderedGroup˙ZF˙1˙L1 group0.inv˙cancel˙two by simp
next assume a ≤ c·b⁻¹
 with ordGroupAssum A1 have a·b ≤ c·b⁻¹·b
  using OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group IsAnOrdGroup˙def
  by simp
 with A1 show a·b ≤ c
   using OrderedGroup˙ZF˙1˙L1 group0.inv˙cancel˙two by simp
qed
```

A one side version of the previous lemma with weaker assuptions.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L9B:
 assumes A1: a∈G  b∈G and A2: a·b⁻¹ ≤ c
 shows a ≤ c·b
```

```
proof -
  from A1 A2 have a∈G  b⁻¹∈G  c∈G
    using OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group
      OrderedGroup˙ZF˙1˙L4 by auto
  with A1 A2 show a ≤ c·b
    using OrderedGroup˙ZF˙1˙L9A OrderedGroup˙ZF˙1˙L1
      group0.group˙inv˙of˙inv by simp
qed
```

We can put en element on the other side of inequality, changing its sign.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L9C:
  assumes A1: a∈G  b∈G and A2: c≤a·b
  shows
  c·b⁻¹ ≤ a
  a⁻¹·c ≤ b
proof -
  from ordGroupAssum A1 A2 have
    c·b⁻¹ ≤ a·b·b⁻¹
    a⁻¹·c ≤ a⁻¹·(a·b)
    using OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group IsAnOrdGroup˙def
    by auto
  with A1 show
    c·b⁻¹ ≤ a
    a⁻¹·c ≤ b
    using OrderedGroup˙ZF˙1˙L1 group0.inv˙cancel˙two
    by auto
qed
```

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

```
lemma (in group3) OrderedGroup˙ZF˙1˙L9D: assumes A1: a≤b
  shows 1 ≤ b·a⁻¹
proof -
  from A1 have T: a∈G  b∈G   a⁻¹ ∈ G
    using OrderedGroup˙ZF˙1˙L4 OrderedGroup˙ZF˙1˙L1
      group0.inverse˙in˙group by auto
  with ordGroupAssum A1 have a·a⁻¹ ≤ b·a⁻¹
    using IsAnOrdGroup˙def by simp
  with T show 1 ≤ b·a⁻¹
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L6
    by simp
qed
```

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

```
lemma (in group3) OrderedGroup˙ZF˙1˙L9E:
  assumes A1: a≤b  a≠b
  shows 1 ≤ b·a⁻¹  1 ≠ b·a⁻¹  b·a⁻¹ ∈ G₊
proof -
  from A1 have T: a∈G  b∈G using OrderedGroup˙ZF˙1˙L4
```

```
    by auto
  from A1 show I: 1 ≤ b·a⁻¹ using OrderedGroup˙ZF˙1˙L9D
    by simp
  { assume b·a⁻¹ = 1
    with T have a=b
      using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L11A
      by auto
    with A1 have False by simp
  } then show 1 ≠ b·a⁻¹ by auto
  then have b·a⁻¹ ≠ 1 by auto
  with I show b·a⁻¹ ∈ G₊ using OrderedGroup˙ZF˙1˙L2A
    by simp
qed
```

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

```
lemma (in group3) OrderedGroup˙ZF˙1˙L9F:
  assumes A1: a∈G  b∈G and A2: 1 ≤ b·a⁻¹
  shows a≤b
proof -
  from A1 A2 have 1·a ≤ b
    using OrderedGroup˙ZF˙1˙L4 OrderedGroup˙ZF˙1˙L9A
    by simp
  with A1 show a≤b
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2
    by simp
qed
```

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

```
lemma (in group3) OrderedGroup˙ZF˙1˙L10:
  assumes a∈G  b∈G and c≤d
  shows a·c·b ≤ a·d·b
  using ordGroupAssum assms IsAnOrdGroup˙def by simp
```

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

```
lemma (in group3) OrderedGroup˙ZF˙1˙L11:
  assumes A1: 1≤a  1≤b
  and A2: 1 ≠ a  1 ≠ b
  shows 1 ≠ a·b
proof -
  from A1 have T1: a∈G  b∈G
    using OrderedGroup˙ZF˙1˙L4 by auto
  { assume 1 = a·b
    with A1 T1 have a≤1  1≤a
      using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L9 OrderedGroup˙ZF˙1˙L5AA
      by auto
    then have a = 1 by (rule group˙order˙antisym)
    with A2 have False by simp
  } then show 1 ≠ a·b by auto
qed
```

A product of nonnegative elements is nonnegative.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L12:
  assumes A1: 1 ≤ a  1 ≤ b
  shows 1 ≤ a·b
proof -
  from A1 have 1·1 ≤ a·b
    using  OrderedGroup˙ZF˙1˙L5B by simp
  then show 1 ≤ a·b
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2
    by simp
qed
```

If a is not greater than b, then 1 is not greater than $b \cdot a^{-1}$.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L12A:
  assumes A1: a≤b shows 1 ≤ b·a⁻¹
proof -
  from A1 have T: 1 ∈ G  a∈G  b∈G
    using OrderedGroup˙ZF˙1˙L4 OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2
    by auto
  with A1 have 1·a ≤ b
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2
    by simp
  with T show 1 ≤ b·a⁻¹ using OrderedGroup˙ZF˙1˙L9A
    by simp
qed
```

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

```
lemma (in group3) OrderedGroup˙ZF˙1˙L12B:
  assumes A1: a∈G  b∈G and  A2: a·b⁻¹ ¡ c
  shows a ¡ c·b
proof -
  from A1 A2 have a·b⁻¹·b ¡ c·b
    using group˙strict˙ord˙transl˙inv by auto
  moreover from A1 have a·b⁻¹·b = a
    using OrderedGroup˙ZF˙1˙L1 group0.inv˙cancel˙two
    by simp
  ultimately show a ¡ c·b
    by auto
qed
```

We can multiply the sides of two inequalities, first of them strict and we get a strict inequality.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L12C:
  assumes A1: a¡b and A2: c≤d
  shows a·c ¡ b·d
proof -
  from A1 A2 have T: a∈G  b∈G  c∈G  d∈G
    using OrderedGroup˙ZF˙1˙L4 by auto
```

```
  with ordGroupAssum A2 have a·c ≤ a·d
    using IsAnOrdGroup˙def by simp
  moreover from A1 T have a·d ¡ b·d
    using group˙strict˙ord˙transl˙inv by simp
  ultimately show a·c ¡ b·d
    by (rule group˙strict˙ord˙transit)
qed
```

We can multiply the sides of two inequalities, second of them strict and we get a strict inequality.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L12D:
  assumes A1: a≤b and A2: c¡d
  shows a·c ¡ b·d
proof -
  from A1 A2 have T: a∈G  b∈G  c∈G  d∈G
    using OrderedGroup˙ZF˙1˙L4 by auto
  with A2 have a·c ¡ a·d
    using group˙strict˙ord˙transl˙inv by simp
  moreover from ordGroupAssum A1 T have a·d ≤ b·d
     using IsAnOrdGroup˙def by simp
   ultimately show a·c ¡ b·d
     by (rule OrderedGroup˙ZF˙1˙L4A)
qed
```

33.3 The set of positive elements

In this section we study G_+ - the set of elements that are (strictly) greater than the unit. The most important result is that every linearly ordered group can decomposed into $\{1\}$, G_+ and the set of those elements $a \in G$ such that $a^{-1} \in G_+$. Another property of linearly ordered groups that we prove here is that if $G_+ \neq \emptyset$, then it is infinite. This allows to show that nontrivial linearly ordered groups are infinite.

The positive set is closed under the group operation.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L13: shows G+ –is closed under˝ P
proof -
  { fix a b assume a∈G+  b∈G+
    then have T1: 1 ≤ a·b   and 1 ≠ a·b
      using PositiveSet˙def OrderedGroup˙ZF˙1˙L11 OrderedGroup˙ZF˙1˙L12
      by auto
    moreover from T1 have a·b ∈ G
      using OrderedGroup˙ZF˙1˙L4 by simp
    ultimately have a·b ∈ G+ using PositiveSet˙def by simp
  } then show G+ –is closed under˝ P using IsOpClosed˙def
    by simp
qed
```

For totally ordered groups every nonunit element is positive or its inverse is positive.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L14:
  assumes A1: r –is total on″ G and A2: a∈G
  shows a=1 ∨ a∈G+ ∨ a⁻¹∈G+
proof -
  { assume A3: a≠1
    moreover from A1 A2 have a≤1 ∨ 1≤a
      using IsTotal˙def OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2
      by simp
    moreover from A3 A2 have T1: a⁻¹ ≠ 1
      using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L8B
      by simp
    ultimately have a⁻¹∈G+ ∨ a∈G+
      using OrderedGroup˙ZF˙1˙L5A OrderedGroup˙ZF˙1˙L2A
      by auto
  } thus a=1 ∨ a∈G+ ∨ a⁻¹∈G+ by auto
qed
```

If an element belongs to the positive set, then it is not the unit and its inverse does not belong to the positive set.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L15:
   assumes A1: a∈G+  shows a≠1  a⁻¹∉G+
proof -
  from A1 show T1: a≠1 using PositiveSet˙def by auto
  { assume a⁻¹ ∈ G+
    with A1 have a≤1  1≤a
      using OrderedGroup˙ZF˙1˙L5AA PositiveSet˙def by auto
    then have a=1 by (rule group˙order˙antisym)
    with T1 have False by simp
  } then show a⁻¹∉G+ by auto
qed
```

If a^{-1} is positive, then a can not be positive or the unit.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L16:
  assumes A1: a∈G and A2: a⁻¹∈G+ shows a≠1  a∉G+
proof -
  from A2 have a⁻¹≠1  (a⁻¹)⁻¹ ∉ G+
    using OrderedGroup˙ZF˙1˙L15 by auto
  with A1 show a≠1  a∉G+
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L8C group0.group˙inv˙of˙inv
    by auto
qed
```

For linearly ordered groups each element is either the unit, positive or its inverse is positive.

```
lemma (in group3) OrdGroup˙decomp:
  assumes A1: r –is total on″ G and A2: a∈G
  shows Exactly˙1˙of˙3˙holds (a=1,a∈G+,a⁻¹∈G+)
proof -
```

```
  from A1 A2 have a=1 ∨ a∈G+ ∨ a⁻¹∈G+
    using OrderedGroup˙ZF˙1˙L14 by simp
  moreover from A2 have a=1 ⟶ (a∉G+ ∧ a⁻¹∉G+)
    using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙one
    PositiveSet˙def by simp
  moreover from A2 have a∈G+ ⟶ (a≠1 ∧ a⁻¹∉G+)
    using OrderedGroup˙ZF˙1˙L15 by simp
  moreover from A2 have a⁻¹∈G+ ⟶ (a≠1 ∧ a∉G+)
    using OrderedGroup˙ZF˙1˙L16 by simp
  ultimately show Exactly˙1˙of˙3˙holds (a=1,a∈G+,a⁻¹∈G+)
    by (rule Fol1˙L5)
qed
```

A if a is a nonunit element that is not positive, then a^{-1} is is positive. This is useful for some proofs by cases.

```
lemma (in group3) OrdGroup˙cases:
  assumes A1: r –is total on″ G and A2: a∈G
  and A3: a≠1  a∉G+
  shows a⁻¹ ∈ G+
proof -
  from A1 A2 have a=1 ∨ a∈G+ ∨ a⁻¹∈G+
    using OrderedGroup˙ZF˙1˙L14 by simp
  with A3 show a⁻¹ ∈ G+ by auto
qed
```

Elements from $G \setminus G_+$ are not greater that the unit.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L17:
  assumes A1: r –is total on″ G and A2: a ∈ G-G+
  shows a≤1
proof -
  { assume a=1
    with A2 have a≤1 using OrderedGroup˙ZF˙1˙L3 by simp }
  moreover
  { assume a≠1
    with A1 A2 have a≤1
      using PositiveSet˙def OrderedGroup˙ZF˙1˙L8A
      by auto }
  ultimately show a≤1 by auto
qed
```

The next lemma allows to split proofs that something holds for all $a \in G$ into cases $a = 1$, $a \in G_+$, $-a \in G_+$.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L18:
  assumes A1: r –is total on″ G and A2: b∈G
  and A3: Q(1) and A4: ∀a∈G+. Q(a) and A5: ∀a∈G+. Q(a⁻¹)
  shows Q(b)
proof -
  from A1 A2 A3 A4 A5 have Q(b) ∨ Q((b⁻¹)⁻¹)
```

```
    using OrderedGroup˙ZF˙1˙L14 by auto
  with A2 show Q(b) using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
    by simp
qed
```

All elements greater or equal than an element of G_+ belong to G_+.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L19:
  assumes A1: a ∈ G+ and A2: a≤b
  shows b ∈ G+
proof -
  from A1 have I: 1≤a  and II: a≠1
    using OrderedGroup˙ZF˙1˙L2A by auto
  from I A2 have 1≤b by (rule Group˙order˙transitive)
  moreover have b≠1
  proof -
    { assume b=1
      with I A2 have 1≤a  a≤1
 by auto
      then have 1=a by (rule group˙order˙antisym)
      with II have False by simp
    } then show b≠1 by auto
  qed
  ultimately show b ∈ G+
    using OrderedGroup˙ZF˙1˙L2A by simp
qed
```

The inverse of an element of G_+ cannot be in G_+.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L20:
  assumes A1: r –is total on" G and A2: a ∈ G+
  shows a⁻¹ ∉ G+
proof -
  from A2 have a∈G using PositiveSet˙def
    by simp
  with A1 have Exactly˙1˙of˙3˙holds (a=1,a∈G+,a⁻¹∈G+)
    using OrdGroup˙decomp by simp
  with A2 show a⁻¹ ∉ G+ by (rule Fol1˙L7)
qed
```

The set of positive elements of a nontrivial linearly ordered group is not empty.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L21:
  assumes A1: r –is total on" G and A2: G ≠ –1"
  shows G+ ≠ 0
proof -
  have 1 ∈ G using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2
    by simp
  with A2 obtain a where a∈G  a≠1 by auto
  with A1 have a∈G+ ∨ a⁻¹∈G+
    using OrderedGroup˙ZF˙1˙L14 by auto
```

```
  then show G+ ≠ 0 by auto
qed
```

If $b \in G_+$, then $a < a \cdot b$. Multiplying a by a positive elemnt increases a.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L22:
  assumes A1: a∈G  b∈G+
  shows a≤a·b  a ≠ a·b  a·b ∈ G
proof -
  from ordGroupAssum A1 have a·1 ≤ a·b
    using OrderedGroup˙ZF˙1˙L2A IsAnOrdGroup˙def
    by simp
  with A1 show a≤a·b
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2
    by simp
  then show a·b ∈ G
    using OrderedGroup˙ZF˙1˙L4 by simp
  { from A1 have a∈G  b∈G
      using PositiveSet˙def by auto
    moreover assume a = a·b
    ultimately have b = 1
      using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L7
      by simp
    with A1 have False using PositiveSet˙def
      by simp
  } then show a ≠ a·b by auto
qed
```

If G is a nontrivial linearly ordered hroup, then for every element of G we can find one in G_+ that is greater or equal.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L23:
  assumes A1: r –is total on" G and A2: G ≠ –1"
  and A3: a∈G
  shows ∃b∈G+. a≤b
proof -
  { assume A4: a∈G+ then have a≤a
      using PositiveSet˙def OrderedGroup˙ZF˙1˙L3
      by simp
    with A4 have ∃b∈G+. a≤b by auto }
  moreover
  { assume a∉G+
    with A1 A3 have I: a≤1 using OrderedGroup˙ZF˙1˙L17
      by simp
    from A1 A2 obtain b where II: b∈G+
      using OrderedGroup˙ZF˙1˙L21 by auto
    then have 1≤b using PositiveSet˙def by simp
    with I have a≤b by (rule Group˙order˙transitive)
    with II have ∃b∈G+. a≤b by auto }
  ultimately show thesis by auto
qed
```

The G^+ is G_+ plus the unit.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L24: shows G⁺ = G₊∪–1″
  using OrderedGroup˙ZF˙1˙L2 OrderedGroup˙ZF˙1˙L2A OrderedGroup˙ZF˙1˙L3A
  by auto
```

What is $-G_+$, really?

```
lemma (in group3) OrderedGroup˙ZF˙1˙L25: shows
  (-G₊) = –a⁻¹. a∈G₊″
  (-G₊) ⊆ G
proof -
  from ordGroupAssum have I: GroupInv(G,P) : G→G
    using IsAnOrdGroup˙def group0˙2˙T2 by simp
  moreover have G₊ ⊆ G using PositiveSet˙def by auto
  ultimately show
    (-G₊) = –a⁻¹. a∈G₊″
    (-G₊) ⊆ G
    using func˙imagedef func1˙1˙L6 by auto
qed
```

If the inverse of a is in G_+, then a is in the inverse of G_+.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L26:
  assumes A1: a∈G and A2: a⁻¹ ∈ G₊
  shows a ∈ (-G₊)
proof -
  from A1 have a⁻¹ ∈ G  a = (a⁻¹)⁻¹ using OrderedGroup˙ZF˙1˙L1
    group0.inverse˙in˙group group0.group˙inv˙of˙inv
    by auto
  with A2 show a ∈ (-G₊) using OrderedGroup˙ZF˙1˙L25
    by auto
qed
```

If a is in the inverse of G_+, then its inverse is in G_+.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L27:
  assumes a ∈ (-G₊)
  shows a⁻¹ ∈ G₊
  using assms OrderedGroup˙ZF˙1˙L25 PositiveSet˙def
    OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
  by auto
```

A linearly ordered group can be decomposed into G_+, $\{1\}$ and $-G_+$

```
lemma (in group3) OrdGroup˙decomp2:
  assumes A1: r –is total on″ G
  shows
  G = G₊ ∪ (-G₊)∪ –1″
  G₊∩(-G₊) = 0
  1 ∉ G₊∪(-G₊)
proof -
  { fix a assume A2: a∈G
```

```
    with A1 have a∈G+ ∨ a⁻¹∈G+ ∨ a=1
      using OrderedGroup˙ZF˙1˙L14 by auto
    with A2 have a∈G+ ∨ a∈(-G+) ∨ a=1
      using OrderedGroup˙ZF˙1˙L26 by auto
    then have a ∈ (G+ ∪ (-G+)∪ −1″)
      by auto
  } then have G ⊆ G+ ∪ (-G+)∪ −1″
    by auto
  moreover have G+ ∪ (-G+)∪ −1″ ⊆ G
    using OrderedGroup˙ZF˙1˙L25 PositiveSet˙def
      OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2
    by auto
  ultimately show G = G+ ∪ (-G+)∪ −1″ by auto
  { let A = G+∩(-G+)
    assume G+∩(-G+) ≠ 0
    then have A≠0 by simp
    then obtain a where a∈A by blast
    then have False using OrderedGroup˙ZF˙1˙L15 OrderedGroup˙ZF˙1˙L27
      by auto
  } then show G+∩(-G+) = 0 by auto
  show 1 ∉ G+∪(-G+)
    using OrderedGroup˙ZF˙1˙L27
      OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙one
      OrderedGroup˙ZF˙1˙L2A by auto
qed
```

If $a \cdot b^{-1}$ is nonnegative, then $b \leq a$. This maybe used to recover the order from the set of nonnegative elements and serve as a way to define order by prescibing that set (see the "Alternative definitions" section).

```
lemma (in group3) OrderedGroup˙ZF˙1˙L28:
  assumes A1: a∈G  b∈G and A2: a·b⁻¹ ∈ G⁺
  shows b≤a
proof -
  from A2 have 1 ≤ a·b⁻¹ using OrderedGroup˙ZF˙1˙L2
    by simp
  with A1 show b≤a using OrderedGroup˙ZF˙1˙L5K
    by simp
qed
```

A special case of OrderedGroup˙ZF˙1˙L28 when $a \cdot b^{-1}$ is positive.

```
corollary (in group3) OrderedGroup˙ZF˙1˙L29:
  assumes A1: a∈G  b∈G and A2: a·b⁻¹ ∈ G+
  shows b≤a  b≠a
proof -
  from A2 have 1 ≤ a·b⁻¹ and I: a·b⁻¹ ≠ 1
    using OrderedGroup˙ZF˙1˙L2A by auto
  with A1 show b≤a using OrderedGroup˙ZF˙1˙L5K
    by simp
  from A1 I show b≠a
```

```
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L6
    by auto
qed
```

A bit stronger that OrderedGroup˙ZF˙1˙L29, adds case when two elements are equal.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L30:
  assumes a∈G  b∈G and a=b ∨ b·a⁻¹ ∈ G₊
  shows a≤b
  using assms OrderedGroup˙ZF˙1˙L3 OrderedGroup˙ZF˙1˙L29
  by auto
```

A different take on decomposition: we can have $a = b$ or $a < b$ or $b < a$.

```
lemma (in group3) OrderedGroup˙ZF˙1˙L31:
  assumes A1: r –is total on″ G and A2: a∈G  b∈G
  shows a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
proof -
  from A2 have a·b⁻¹ ∈ G using OrderedGroup˙ZF˙1˙L1
    group0.inverse˙in˙group group0.group˙op˙closed
    by simp
  with A1 have a·b⁻¹ = 1 ∨ a·b⁻¹ ∈ G₊ ∨ (a·b⁻¹)⁻¹ ∈ G₊
    using OrderedGroup˙ZF˙1˙L14 by simp
  moreover
  { assume a·b⁻¹ = 1
    then have a·b⁻¹·b = 1·b by simp
    with A2 have a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
      using OrderedGroup˙ZF˙1˙L1
 group0.inv˙cancel˙two group0.group0˙2˙L2 by auto }
  moreover
  { assume a·b⁻¹ ∈ G₊
    with A2 have a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
      using OrderedGroup˙ZF˙1˙L29 by auto }
  moreover
  { assume (a·b⁻¹)⁻¹ ∈ G₊
    with A2 have b·a⁻¹ ∈ G₊ using OrderedGroup˙ZF˙1˙L1
      group0.group0˙2˙L12 by simp
    with A2 have a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
      using OrderedGroup˙ZF˙1˙L29 by auto }
  ultimately show a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
    by auto
qed
```

33.4 Intervals and bounded sets

Intervals here are the closed intervals of the form $\{x \in G.a \leq x \leq b\}$.

A bounded set can be translated to put it in G^+ and then it is still bounded above.

```
lemma (in group3) OrderedGroup˙ZF˙2˙L1:
  assumes A1: ∀g∈A. L≤g ∧ g≤M
  and A2: S = RightTranslation(G,P,L⁻¹)
  and A3: a ∈ S(A)
  shows a ≤ M·L⁻¹   1≤a
proof -
  from A3 have A≠0 using func1˙1˙L13A by fast
  then obtain g where g∈A by auto
  with A1 have T1: L∈G M∈G L⁻¹∈G
    using OrderedGroup˙ZF˙1˙L4 OrderedGroup˙ZF˙1˙L1
    group0.inverse˙in˙group by auto
  with A2 have S : G→G using OrderedGroup˙ZF˙1˙L1 group0.group0˙5˙L1
    by simp
  moreover from A1 have T2: A⊆G using OrderedGroup˙ZF˙1˙L4 by auto
  ultimately have S(A) = –S(b). b∈A″ using func˙imagedef
    by simp
  with A3 obtain b where T3: b∈A a = S(b) by auto
  with A1 ordGroupAssum T1 have b·L⁻¹≤M·L⁻¹ L·L⁻¹≤b·L⁻¹
    using IsAnOrdGroup˙def by auto
  with T3 A2 T1 T2 show a≤M·L⁻¹ 1≤a
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙5˙L2 group0.group0˙2˙L6
    by auto
qed
```

Every bounded set is an image of a subset of an interval that starts at 1.

```
lemma (in group3) OrderedGroup˙ZF˙2˙L2:
  assumes A1: IsBounded(A,r)
  shows ∃B.∃g∈G⁺.∃T∈G→G. A = T(B) ∧ B ⊆ Interval(r,1,g)
proof -
  { assume A2: A=0
    let B = 0
    let g = 1
    let T = ConstantFunction(G,1)
    have g∈G⁺ using OrderedGroup˙ZF˙1˙L3A by simp
    moreover have T : G→G
      using func1˙3˙L1 OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2 by simp
    moreover from A2 have A = T(B) by simp
    moreover have B ⊆ Interval(r,1,g) by simp
    ultimately have
      ∃B.∃g∈G⁺.∃T∈G→G. A = T(B) ∧ B ⊆ Interval(r,1,g)
      by auto }
  moreover
  { assume A3: A≠0
    with A1 have ∃L. ∀x∈A. L≤x and ∃U. ∀x∈A. x≤U
      using IsBounded˙def IsBoundedBelow˙def IsBoundedAbove˙def
      by auto
    then obtain L U where D1: ∀x∈A. L≤x ∧ x≤U
      by auto
    with A3 have T1: A⊆G using OrderedGroup˙ZF˙1˙L4 by auto
```

from A3 obtain a where a∈A by auto
with D1 have T2: L≤a a≤U by auto
then have T3: L∈G L^{-1}∈ G U∈G
using OrderedGroup˙ZF˙1˙L4 OrderedGroup˙ZF˙1˙L1
group0.inverse˙in˙group by auto
let T = RightTranslation(G,P,L)
let B = RightTranslation(G,P,L^{-1})(A)
let g = U·L^{-1}
have g∈G^{+}
proof -
from T2 have L≤U using Group˙order˙transitive by fast
with ordGroupAssum T3 have L·L^{-1}≤g
using IsAnOrdGroup˙def by simp
with T3 show thesis using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L6
OrderedGroup˙ZF˙1˙L2 by simp
qed
moreover from T3 have T : G→G
using OrderedGroup˙ZF˙1˙L1 group0.group0˙5˙L1
by simp
moreover have A = T(B)
proof -
from T3 T1 have T(B) = −a·L^{-1}·L. a∈A″
using OrderedGroup˙ZF˙1˙L1 group0.group0˙5˙L6
by simp
moreover from T3 T1 have ∀a∈A. a·L^{-1}·L = a·(L^{-1}·L)
using OrderedGroup˙ZF˙1˙L1 group0.group˙oper˙assoc by auto
ultimately have T(B) = −a·(L^{-1}·L). a∈A″ by simp
with T3 have T(B) = −a·**1**. a∈A″
using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L6 by simp
moreover from T1 have ∀a∈A. a·**1**=a
using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2 by auto
ultimately show thesis by simp
qed
moreover have B ⊆ Interval(r,**1**,g)
proof
fix y assume A4: y ∈ B
let S = RightTranslation(G,P,L^{-1})
from D1 have T4: ∀x∈A. L≤x ∧ x≤U by simp
moreover have T5: S = RightTranslation(G,P,L^{-1})
by simp
moreover from A4 have T6: y ∈ S(A) by simp
ultimately have y≤U·L^{-1} using OrderedGroup˙ZF˙2˙L1
by blast
moreover from T4 T5 T6 have **1**≤y by (rule OrderedGroup˙ZF˙2˙L1)
ultimately show y ∈ Interval(r,**1**,g) using Interval˙def by auto
qed
ultimately have
∃B.∃g∈G^{+}.∃T∈G→G. A = T(B) ∧ B ⊆ Interval(r,**1**,g)
by auto }

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

If every interval starting at 1 is finite, then every bounded set is finite. I find it interesting that this does not require the group to be linearly ordered (the order to be total).

```
theorem (in group3) OrderedGroup˙ZF˙2˙T1:
  assumes A1: ∀g∈G⁺. Interval(r,1,g) ∈ Fin(G)
  and A2: IsBounded(A,r)
  shows A ∈ Fin(G)
proof -
  from A2 have
    ∃B.∃g∈G⁺.∃T∈G→G. A = T(B) ∧ B ⊆ Interval(r,1,g)
    using OrderedGroup˙ZF˙2˙L2 by simp
  then obtain B g T where D1: g∈G⁺ B ⊆ Interval(r,1,g)
    and D2: T : G→G A = T(B) by auto
  from D1 A1 have B∈Fin(G) using Fin˙subset˙lemma by blast
  with D2 show thesis using Finite1˙L6A by simp
qed
```

In linearly ordered groups finite sets are bounded.

```
theorem (in group3) ord˙group˙fin˙bounded:
  assumes r –is total on″ G and B∈Fin(G)
  shows IsBounded(B,r)
  using ordGroupAssum assms IsAnOrdGroup˙def IsPartOrder˙def Finite˙ZF˙1˙T1
  by simp
```

For nontrivial linearly ordered groups if for every element G we can find one in A that is greater or equal (not necessarily strictly greater), then A can neither be finite nor bounded above.

```
lemma (in group3) OrderedGroup˙ZF˙2˙L2A:
  assumes A1: r –is total on″ G and A2: G ≠ –1″
  and A3: ∀a∈G. ∃b∈A. a≤b
  shows
  ∀a∈G. ∃b∈A. a≠b ∧ a≤b
  ¬IsBoundedAbove(A,r)
  A ∉ Fin(G)
proof -
  { fix a
    from A1 A2 obtain c where c ∈ G₊
      using OrderedGroup˙ZF˙1˙L21 by auto
    moreover assume a∈G
    ultimately have
      a·c ∈ G  and I: a ¡ a·c
      using OrderedGroup˙ZF˙1˙L22 by auto
    with A3 obtain b where II: b∈A  and III: a·c ≤ b
      by auto
    moreover from I III have a¡b by (rule OrderedGroup˙ZF˙1˙L4A)
```

```
    ultimately have ∃b∈A. a≠b ∧ a≤b by auto
  } thus ∀a∈G. ∃b∈A. a≠b ∧ a≤b by simp
  with ordGroupAssum A1 show
    ¬IsBoundedAbove(A,r)
    A ∉ Fin(G)
    using IsAnOrdGroup˙def IsPartOrder˙def
    OrderedGroup˙ZF˙1˙L1A Order˙ZF˙3˙L14 Finite˙ZF˙1˙1˙L3
    by auto
qed
```

Nontrivial linearly ordered groups are infinite. Recall that Fin(A) is the collection of finite subsets of A. In this lemma we show that $G \notin$ Fin(G), that is that G is not a finite subset of itself. This is a way of saying that G is infinite. We also show that for nontrivial linearly ordered groups G_+ is infinite.

```
theorem (in group3) Linord˙group˙infinite:
  assumes A1: r –is total on″ G and A2: G ≠ –1″
  shows
  G+ ∉ Fin(G)
  G ∉ Fin(G)
proof -
  from A1 A2 show I: G+ ∉ Fin(G)
    using OrderedGroup˙ZF˙1˙L23 OrderedGroup˙ZF˙2˙L2A
    by simp
  { assume G ∈ Fin(G)
    moreover have G+ ⊆ G using PositiveSet˙def by auto
    ultimately have G+ ∈ Fin(G) using Fin˙subset˙lemma
      by blast
    with I have False by simp
  } then show G ∉ Fin(G) by auto
qed
```

A property of nonempty subsets of linearly ordered groups that don't have a maximum: for any element in such subset we can find one that is strictly greater.

```
lemma (in group3) OrderedGroup˙ZF˙2˙L2B:
  assumes A1: r –is total on″ G and A2: A⊆G and
  A3: ¬HasAmaximum(r,A) and A4: x∈A
  shows ∃y∈A. x¡y
proof -
  from ordGroupAssum assms have
    antisym(r)
    r –is total on″ G
    A⊆G  ¬HasAmaximum(r,A)  x∈A
    using IsAnOrdGroup˙def IsPartOrder˙def
    by auto
  then have ∃y∈A. ⟨x,y⟩ ∈ r ∧ y≠x
    using Order˙ZF˙4˙L16 by simp
```

```
  then show ∃y∈A. x¡y by auto
qed
```

In linearly ordered groups $G \setminus G_+$ is bounded above.

```
lemma (in group3) OrderedGroup˙ZF˙2˙L3:
  assumes A1: r –is total on″ G shows IsBoundedAbove(G-G+,r)
proof -
  from A1 have ∀a∈G-G+. a≤1
    using OrderedGroup˙ZF˙1˙L17 by simp
  then show IsBoundedAbove(G-G+,r)
    using IsBoundedAbove˙def by auto
qed
```

In linearly ordered groups if $A \cap G_+$ is finite, then A is bounded above.

```
lemma (in group3) OrderedGroup˙ZF˙2˙L4:
  assumes A1: r –is total on″ G and A2: A⊆G
  and A3: A ∩ G+ ∈ Fin(G)
  shows IsBoundedAbove(A,r)
proof -
  have A ∩ (G-G+) ⊆ G-G+ by auto
  with A1 have IsBoundedAbove(A ∩ (G-G+),r)
    using OrderedGroup˙ZF˙2˙L3 Order˙ZF˙3˙L13
    by blast
  moreover from A1 A3 have IsBoundedAbove(A ∩ G+,r)
    using ord˙group˙fin˙bounded IsBounded˙def
    by simp
  moreover from A1 ordGroupAssum have
    r –is total on″ G  trans(r)  r⊆G×G
    using IsAnOrdGroup˙def IsPartOrder˙def by auto
  ultimately have IsBoundedAbove(A ∩ (G-G+) ∪ A ∩ G+,r)
    using Order˙ZF˙3˙L3 by simp
  moreover from A2 have A = A ∩ (G-G+) ∪ A ∩ G+
    by auto
  ultimately show  IsBoundedAbove(A,r) by simp
qed
```

If a set $-A \subseteq G$ is bounded above, then A is bounded below.

```
lemma (in group3) OrderedGroup˙ZF˙2˙L5:
  assumes A1: A⊆G and A2: IsBoundedAbove(-A,r)
  shows IsBoundedBelow(A,r)
proof -
  { assume A = 0 then have IsBoundedBelow(A,r)
      using IsBoundedBelow˙def by auto }
  moreover
  { assume A3: A≠0
    from ordGroupAssum have I: GroupInv(G,P) : G→G
      using IsAnOrdGroup˙def group0˙2˙T2 by simp
    with A1 A2 A3 obtain u where D: ∀a∈(-A). a≤u
      using func1˙1˙L15A IsBoundedAbove˙def by auto
```

```
    { fix b assume b∈A
      with A1 I D have b^-1 ≤ u and T: b∈G
 using func˙imagedef by auto
      then have u^-1≤(b^-1)^-1 using OrderedGroup˙ZF˙1˙L5
 by simp
      with T have u^-1≤b
 using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
 by simp
    } then have ∀b∈A. ⟨u^-1,b⟩ ∈ r by simp
    then have IsBoundedBelow(A,r)
      using Order˙ZF˙3˙L9 by blast }
  ultimately show thesis by auto
qed
```

If $a \leq b$, then the image of the interval $a..b$ by any function is nonempty.

```
lemma (in group3) OrderedGroup˙ZF˙2˙L6:
  assumes a≤b and f:G→G
  shows f(Interval(r,a,b)) ≠ 0
  using ordGroupAssum assms OrderedGroup˙ZF˙1˙L4
    Order˙ZF˙2˙L6 Order˙ZF˙2˙L2A
    IsAnOrdGroup˙def IsPartOrder˙def func1˙1˙L15A
  by auto

end
```

34 More on ordered groups

```
theory OrderedGroup˙ZF˙1 imports OrderedGroup˙ZF

begin
```

In this theory we continue the OrderedGroup˙ZF theory development.

34.1 Absolute value and the triangle inequality

The goal of this section is to prove the triangle inequality for ordered groups.

Absolute value maps G into G.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L1:
  shows AbsoluteValue(G,P,r) : G→G
proof -
  let f = id(G^+)
  let g = restrict(GroupInv(G,P),G-G^+)
  have f : G^+→G^+ using id˙type by simp
  then have f : G^+→G using OrderedGroup˙ZF˙1˙L4E fun˙weaken˙type
    by blast
  moreover have g : G-G^+→G
  proof -
```

```
    from ordGroupAssum have GroupInv(G,P) : G→G
      using IsAnOrdGroup˙def group0˙2˙T2 by simp
    moreover have G-G⁺ ⊆ G by auto
    ultimately show thesis using restrict˙type2 by simp
  qed
  moreover have G⁺∩(G-G⁺) = 0 by blast
  ultimately have f ∪ g : G⁺∪(G-G⁺)→G∪G
    by (rule fun˙disjoint˙Un)
  moreover have G⁺∪(G-G⁺) = G using OrderedGroup˙ZF˙1˙L4E
    by auto
  ultimately show AbsoluteValue(G,P,r) : G→G
    using AbsoluteValue˙def by simp
qed
```

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

```
lemma (in group3) OrderedGroup˙ZF˙3˙L2:
  assumes A1: a∈G⁺ shows —a— = a
proof -
  from ordGroupAssum have GroupInv(G,P) : G→G
    using IsAnOrdGroup˙def group0˙2˙T2 by simp
  with A1 show thesis using
    func1˙1˙L1 OrderedGroup˙ZF˙1˙L4E fun˙disjoint˙apply1
    AbsoluteValue˙def id˙conv by simp
qed
```

The absolute value of the unit is the unit. In the additive totation that would be $|0| = 0$.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L2A:
  shows —1— = 1 using OrderedGroup˙ZF˙1˙L3A OrderedGroup˙ZF˙3˙L2
  by simp
```

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

```
lemma (in group3) OrderedGroup˙ZF˙3˙L2B:
  assumes a∈G₊ shows —a— = a
  using assms PositiveSet˙def Nonnegative˙def OrderedGroup˙ZF˙3˙L2
  by auto
```

If $a \in G \setminus G^+$, then $|a| = a^{-1}$.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L3:
   assumes A1: a ∈ G-G⁺ shows —a— = a⁻¹
proof -
  have domain(id(G⁺)) = G⁺
    using id˙type func1˙1˙L1 by auto
  with A1 show thesis using fun˙disjoint˙apply2 AbsoluteValue˙def
    restrict by simp
qed
```

For elements that not greater than the unit, the absolute value is the inverse.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L3A:
  assumes A1: a≤1
  shows —a— = a⁻¹
proof -
  { assume a=1 then have —a— = a⁻¹
     using OrderedGroup˙ZF˙3˙L2A OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙one
      by simp }
  moreover
  { assume a≠1
   with A1 have —a— = a⁻¹ using OrderedGroup˙ZF˙1˙L4C OrderedGroup˙ZF˙3˙L3
      by simp }
  ultimately show —a— = a⁻¹ by blast
qed
```

In linearly ordered groups the absolute value of any element is in G^+.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L3B:
  assumes A1: r –is total on″ G and A2: a∈G
  shows —a— ∈ G⁺
proof -
  { assume a ∈ G⁺ then have —a— ∈ G⁺
      using OrderedGroup˙ZF˙3˙L2 by simp }
  moreover
  { assume a ∉ G⁺
    with A1 A2 have —a— ∈ G⁺ using OrderedGroup˙ZF˙3˙L3
      OrderedGroup˙ZF˙1˙L6 by simp }
  ultimately show —a— ∈ G⁺ by blast
qed
```

For linearly ordered groups (where the order is total), the absolute value maps the group into the positive set.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L3C:
  assumes A1: r –is total on″ G
  shows AbsoluteValue(G,P,r) : G→G⁺
proof-
  have AbsoluteValue(G,P,r) : G→G using OrderedGroup˙ZF˙3˙L1
    by simp
  moreover from A1 have T2:
    ∀g∈G. AbsoluteValue(G,P,r)(g)  ∈ G⁺
    using OrderedGroup˙ZF˙3˙L3B by simp
  ultimately show thesis by (rule func1˙1˙L1A)
qed
```

If the absolute value is the unit, then the elemnent is the unit.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L3D:
  assumes A1: a∈G and A2: —a— = 1
  shows a = 1
proof -
  { assume a ∈ G⁺
    with A2 have a = 1 using OrderedGroup˙ZF˙3˙L2 by simp }
```

```
  moreover
  { assume a ∉ G+
    with A1 A2 have a = 1 using
      OrderedGroup˙ZF˙3˙L3 OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L8A
      by auto }
  ultimately show a = 1 by blast
qed
```

In linearly ordered groups the unit is not greater than the absolute value of any element.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L3E:
  assumes r –is total on″ G and a∈G
  shows 1 ≤ —a—
  using assms OrderedGroup˙ZF˙3˙L3B OrderedGroup˙ZF˙1˙L2 by simp
```

If b is greater than both a and a^{-1}, then b is greater than $|a|$.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L4:
  assumes A1: a≤b and A2: a⁻¹≤ b
  shows —a—≤ b
proof -
  { assume a∈G+
    with A1 have —a—≤ b using OrderedGroup˙ZF˙3˙L2 by simp }
  moreover
  { assume a∉G+
    with A1 A2 have —a—≤ b
      using OrderedGroup˙ZF˙1˙L4 OrderedGroup˙ZF˙3˙L3 by simp }
  ultimately show —a—≤ b by blast
qed
```

In linearly ordered groups $a \leq |a|$.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L5:
  assumes A1: r –is total on″ G and A2: a∈G
  shows a ≤ —a—
proof -
  { assume a ∈ G+
    with A2 have a ≤ —a—
      using OrderedGroup˙ZF˙3˙L2 OrderedGroup˙ZF˙1˙L3 by simp }
  moreover
  { assume a ∉ G+
    with A1 A2 have a ≤ —a—
      using OrderedGroup˙ZF˙3˙L3B OrderedGroup˙ZF˙1˙L4B by simp }
  ultimately show a ≤ —a— by blast
qed
```

$a^{-1} \leq |a|$ (in additive notation it would be $-a \leq |a|$.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L6:
  assumes A1: a∈G shows a⁻¹ ≤ —a—
proof -
```

```
  { assume a ∈ G⁺
    then have T1: 1≤a and T2: —a— = a using OrderedGroup˙ZF˙1˙L2
      OrderedGroup˙ZF˙3˙L2 by auto
    then have a⁻¹≤1⁻¹ using OrderedGroup˙ZF˙1˙L5 by simp
    then have T3: a⁻¹≤1
      using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙one by simp
    from T3 T1 have a⁻¹≤a by (rule Group˙order˙transitive)
    with T2 have a⁻¹ ≤ —a— by simp }
  moreover
  { assume A2: a ∉ G⁺
    from A1 have —a— ∈ G
      using OrderedGroup˙ZF˙3˙L1 apply˙funtype by auto
    with ordGroupAssum have —a— ≤ —a—
      using IsAnOrdGroup˙def IsPartOrder˙def refl˙def by simp
    with A1 A2 have a⁻¹ ≤ —a— using OrderedGroup˙ZF˙3˙L3 by simp }
  ultimately show a⁻¹ ≤ —a— by blast
qed
```

Some inequalities about the product of two elements of a linearly ordered group and its absolute value.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L6A:
  assumes r –is total on″ G and a∈G  b∈G
  shows
  a·b ≤—a—·—b—
  a·b⁻¹ ≤—a—·—b—
  a⁻¹·b ≤—a—·—b—
  a⁻¹·b⁻¹ ≤—a—·—b—
  using assms OrderedGroup˙ZF˙3˙L5 OrderedGroup˙ZF˙3˙L6
    OrderedGroup˙ZF˙1˙L5B by auto
```

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

```
lemma (in group3) OrderedGroup˙ZF˙3˙L7:
  assumes r –is total on″ G and a∈G
  shows —a⁻¹—≤—a—
 using assms OrderedGroup˙ZF˙3˙L5 OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
    OrderedGroup˙ZF˙3˙L6 OrderedGroup˙ZF˙3˙L4 by simp
```

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

```
lemma (in group3) OrderedGroup˙ZF˙3˙L7A:
  assumes A1: r –is total on″ G and A2: a∈G
  shows —a⁻¹— = —a—
proof -
  from A2 have a⁻¹∈G using OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group
    by simp
  with A1 have —(a⁻¹)⁻¹— ≤ —a⁻¹— using OrderedGroup˙ZF˙3˙L7 by simp
  with A1 A2 have —a⁻¹— ≤ —a—  —a— ≤ —a⁻¹—
    using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv OrderedGroup˙ZF˙3˙L7
    by auto
  then show thesis by (rule group˙order˙antisym)
```

qed

$|a \cdot b^{-1}| = |b \cdot a^{-1}|$. It doesn't look so strange in the additive notation: $|a - b| = |b - a|$.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L7B:
  assumes A1: r –is total on″ G and A2: a∈G b∈G
  shows —a·b⁻¹— = —b·a⁻¹—
proof -
  from A1 A2 have —(a·b⁻¹)⁻¹— = —a·b⁻¹— using
    OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group group0.group0˙2˙L1
    monoid0.group0˙1˙L1 OrderedGroup˙ZF˙3˙L7A by simp
  moreover from A2 have (a·b⁻¹)⁻¹ =  b·a⁻¹
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L12 by simp
  ultimately show thesis by simp
qed
```

Triangle inequality for linearly ordered abelian groups. It would be nice to drop commutativity or give an example that shows we can't do that.

```
theorem (in group3) OrdGroup˙triangle˙ineq:
  assumes A1: P –is commutative on″ G
  and A2: r –is total on″ G and A3: a∈G  b∈G
  shows —a·b— ≤ —a—·—b—
proof -
  from A1 A2 A3 have
    a ≤ —a— b ≤ —b— a⁻¹ ≤ —a— b⁻¹ ≤ —b—
    using OrderedGroup˙ZF˙3˙L5 OrderedGroup˙ZF˙3˙L6 by auto
  then have a·b ≤ —a—·—b— a⁻¹·b⁻¹ ≤ —a—·—b—
    using OrderedGroup˙ZF˙1˙L5B by auto
  with A1 A3 show —a·b— ≤ —a—·—b—
    using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙two IsCommutative˙def
    OrderedGroup˙ZF˙3˙L4 by simp
qed
```

We can multiply the sides of an inequality with absolute value.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L7C:
  assumes 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 assms OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group
OrderedGroup˙ZF˙3˙L7A OrderedGroup˙ZF˙3˙L7C by simp

Triangle inequality with three integers.

lemma (in group3) OrdGroup˙triangle˙ineq3:
assumes A1: P –is commutative on″ G
and A2: r –is total on″ G and A3: a∈G b∈G c∈G
shows —a·b·c— ≤ —a—·—b—·—c—
proof -
from A3 have T: a·b ∈ G —c— ∈ G
using OrderedGroup˙ZF˙1˙L1 group0.group˙op˙closed
OrderedGroup˙ZF˙3˙L1 apply˙funtype by auto
with A1 A2 A3 have —a·b·c— ≤ —a·b—·—c—
using OrdGroup˙triangle˙ineq by simp
moreover from ordGroupAssum A1 A2 A3 T have
—a·b—·—c— ≤ —a—·—b—·—c—
using OrdGroup˙triangle˙ineq IsAnOrdGroup˙def by simp
ultimately show —a·b·c— ≤ —a—·—b—·—c—
by (rule Group˙order˙transitive)
qed

Some variants of the triangle inequality.

lemma (in group3) OrderedGroup˙ZF˙3˙L7D:
assumes A1: P –is commutative on″ G
and A2: r –is total on″ G and A3: a∈G b∈G
and A4: —a·b^{-1}— ≤ c
shows
—a— ≤ c·—b—
—a— ≤ —b—·c
c^{-1}·a ≤ b
a·c^{-1} ≤ b
a ≤ b·c
proof -
from A3 A4 have
T: a·b^{-1} ∈ G —b— ∈ G c∈G c^{-1} ∈ G
using OrderedGroup˙ZF˙1˙L1
group0.inverse˙in˙group group0.group0˙2˙L1 monoid0.group0˙1˙L1
OrderedGroup˙ZF˙3˙L1 apply˙funtype OrderedGroup˙ZF˙1˙L4
by auto
from A3 have —a— = —a·b^{-1}·b—
using OrderedGroup˙ZF˙1˙L1 group0.inv˙cancel˙two
by simp
with A1 A2 A3 T have —a— ≤ —a·b^{-1}—·—b—
using OrdGroup˙triangle˙ineq by simp

```
  with T A4 show —a— ≤ c·—b— using OrderedGroup˙ZF˙1˙L5C
    by blast
  with T A1 show —a— ≤ —b—·c
    using IsCommutative˙def by simp
  from A2 T have a·b⁻¹ ≤ —a·b⁻¹—
    using OrderedGroup˙ZF˙3˙L5 by simp
  moreover note A4
  ultimately have I: a·b⁻¹ ≤ c
    by (rule Group˙order˙transitive)
  with A3 show c⁻¹·a ≤ b
    using OrderedGroup˙ZF˙1˙L5H by simp
  with A1 A3 T show a·c⁻¹ ≤ b
    using IsCommutative˙def by simp
  from A1 A3 T I show a ≤ b·c
    using OrderedGroup˙ZF˙1˙L5H IsCommutative˙def
    by auto
qed
```

Some more variants of the triangle inequality.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L7E:
  assumes A1: P –is commutative on″ G
  and A2: r –is total on″ G and A3: a∈G  b∈G
  and A4: —a·b⁻¹— ≤ c
  shows b·c⁻¹ ≤ a
proof -
  from A3 have a·b⁻¹ ∈ G
    using OrderedGroup˙ZF˙1˙L1
      group0.inverse˙in˙group group0.group˙op˙closed
    by auto
  with A2 have —(a·b⁻¹)⁻¹— = —a·b⁻¹—
    using OrderedGroup˙ZF˙3˙L7A by simp
  moreover from A3 have (a·b⁻¹)⁻¹ = b·a⁻¹
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L12
    by simp
  ultimately have —b·a⁻¹— = —a·b⁻¹—
    by simp
  with A1 A2 A3 A4 show b·c⁻¹ ≤ a
    using OrderedGroup˙ZF˙3˙L7D by simp
qed
```

An application of the triangle inequality with four group elements.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L7F:
  assumes A1: P –is commutative on″ G
  and A2: r –is total on″ G and
  A3: a∈G  b∈G  c∈G  d∈G
  shows —a·c⁻¹— ≤ —a·b—·—c·d—·—b·d⁻¹—
proof -
  from A3 have T:
    a·c⁻¹ ∈ G  a·b ∈ G  c·d ∈ G  b·d⁻¹ ∈ G
```

```
    (c·d)^-1 ∈ G  (b·d^-1)^-1 ∈ G
    using OrderedGroup˙ZF˙1˙L1
      group0.inverse˙in˙group group0.group˙op˙closed
    by auto
  with A1 A2 have —(a·b)·(c·d)^-1·(b·d^-1)^-1— ≤ —a·b—·—(c·d)^-1—·—(b·d^-1)^-1—
    using OrdGroup˙triangle˙ineq3 by simp
  moreover from A2 T have —(c·d)^-1— =—c·d— and —(b·d^-1)^-1— = —b·d^-1—
    using OrderedGroup˙ZF˙3˙L7A by auto
  moreover from A1 A3 have (a·b)·(c·d)^-1·(b·d^-1)^-1 = a·c^-1
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙4˙L8
    by simp
  ultimately show —a·c^-1— ≤ —a·b—·—c·d—·—b·d^-1—
    by simp
qed
```

$|a| \leq L$ implies $L^{-1} \leq a$ (it would be $-L \leq a$ in the additive notation).

```
lemma (in group3) OrderedGroup˙ZF˙3˙L8:
  assumes A1:  a∈G and A2: —a—≤L
   shows
  L^-1≤a
proof -
  from A1 have I: a^-1 ≤ —a— using OrderedGroup˙ZF˙3˙L6 by simp
  from I A2 have a^-1 ≤ L by (rule Group˙order˙transitive)
  then have L^-1≤(a^-1)^-1 using OrderedGroup˙ZF˙1˙L5 by simp
  with A1 show L^-1≤a using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
    by simp
qed
```

In linearly ordered groups $|a| \leq L$ implies $a \leq L$ (it would be $a \leq L$ in the additive notation).

```
lemma (in group3) OrderedGroup˙ZF˙3˙L8A:
  assumes A1: r –is total on″ G
  and A2: a∈G and A3: —a—≤L
  shows
  a≤L
  1≤L
proof -
  from A1 A2 have I: a ≤ —a— using OrderedGroup˙ZF˙3˙L5 by simp
  from I A3 show a≤L by (rule Group˙order˙transitive)
  from A1 A2 A3 have 1 ≤ —a—  —a—≤L
     using OrderedGroup˙ZF˙3˙L3B OrderedGroup˙ZF˙1˙L2 by auto
   then show 1≤L by (rule Group˙order˙transitive)
qed
```

A somewhat generalized version of the above lemma.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L8B:
  assumes A1: a∈G and A2: —a—≤L and A3: 1≤c
  shows (L·c)^-1 ≤ a
proof -
```

```
  from A1 A2 A3 have c⁻¹·L⁻¹ ≤ 1·a
    using OrderedGroup˙ZF˙3˙L8 OrderedGroup˙ZF˙1˙L5AB
    OrderedGroup˙ZF˙1˙L5B by simp
  with A1 A2 A3 show (L·c)⁻¹ ≤ a
    using OrderedGroup˙ZF˙1˙L4 OrderedGroup˙ZF˙1˙L1
      group0.group˙inv˙of˙two group0.group0˙2˙L2
    by simp
qed
```

If b is between a and $a \cdot c$, then $b \cdot a^{-1} \leq c$.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L8C:
  assumes A1: a≤b and A2: c∈G and A3: b≤c·a
  shows —b·a⁻¹— ≤ c
proof -
  from A1 A2 A3 have b·a⁻¹ ≤ c
    using OrderedGroup˙ZF˙1˙L9C OrderedGroup˙ZF˙1˙L4
    by simp
  moreover have (b·a⁻¹)⁻¹ ≤ c
  proof -
    from A1 have T: a∈G  b∈G
      using OrderedGroup˙ZF˙1˙L4 by auto
    with A1 have a·b⁻¹ ≤ 1
      using OrderedGroup˙ZF˙1˙L9 by blast
    moreover
    from A1 A3 have a≤c·a
      by (rule Group˙order˙transitive)
    with ordGroupAssum T have a·a⁻¹ ≤ c·a·a⁻¹
      using OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group
      IsAnOrdGroup˙def by simp
    with T A2 have 1 ≤ c
      using OrderedGroup˙ZF˙1˙L1
 group0.group0˙2˙L6 group0.inv˙cancel˙two
      by simp
    ultimately have a·b⁻¹ ≤ c
      by (rule Group˙order˙transitive)
    with T show (b·a⁻¹)⁻¹ ≤ c
      using OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L12
      by simp
  qed
  ultimately show —b·a⁻¹— ≤ c
    using OrderedGroup˙ZF˙3˙L4 by simp
qed
```

For linearly ordered groups if the absolute values of elements in a set are bounded, then the set is bounded.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L9:
  assumes A1: r –is total on˝ G
  and A2: A⊆G and A3: ∀a∈A. —a— ≤ L
  shows IsBounded(A,r)
```

```
proof -
  from A1 A2 A3 have
    ∀a∈A. a≤L  ∀a∈A. L⁻¹≤a
    using OrderedGroup˙ZF˙3˙L8 OrderedGroup˙ZF˙3˙L8A by auto
  then show IsBounded(A,r) using
    IsBoundedAbove˙def IsBoundedBelow˙def IsBounded˙def
    by auto
qed
```

A slightly more general version of the previous lemma, stating the same fact for a set defined by separation.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L9A:
  assumes A1: r –is total on″ G
  and A2: ∀x∈X. b(x)∈G ∧ —b(x)—≤L
  shows IsBounded(–b(x). x∈X″,r)
proof -
  from A2 have –b(x). x∈X″ ⊆ G ∀a∈–b(x). x∈X″. —a— ≤ L
    by auto
  with A1 show thesis using OrderedGroup˙ZF˙3˙L9 by blast
qed
```

A special form of the previous lemma stating a similar fact for an image of a set by a function with values in a linearly ordered group.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L9B:
  assumes A1: r –is total on″ G
  and A2: f:X→G and A3: A⊆X
  and A4: ∀x∈A. —f(x)— ≤ L
  shows IsBounded(f(A),r)
proof -
  from A2 A3 A4 have ∀x∈A. f(x) ∈ G ∧  —f(x)— ≤ L
    using apply˙funtype by auto
  with A1 have  IsBounded(–f(x). x∈A″,r)
    by (rule OrderedGroup˙ZF˙3˙L9A)
  with A2 A3 show IsBounded(f(A),r)
    using func˙imagedef by simp
qed
```

For linearly ordered groups if $l \leq a \leq u$ then $|a|$ is smaller than the greater of $|l|, |u|$.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L10:
  assumes A1: r –is total on″ G
  and A2: l≤a  a≤u
  shows
  —a— ≤ GreaterOf(r,—l—,—u—)
proof -
  from A2 have T1: —l— ∈ G —a— ∈ G —u— ∈ G
    using OrderedGroup˙ZF˙1˙L4 OrderedGroup˙ZF˙3˙L1 apply˙funtype
    by auto
```

```
  { assume A3: a∈G⁺
    with A2 have 1≤a a≤u
      using OrderedGroup˙ZF˙1˙L2 by auto
    then have 1≤u by (rule Group˙order˙transitive)
    with A2 A3 have —a—≤—u—
      using OrderedGroup˙ZF˙1˙L2 OrderedGroup˙ZF˙3˙L2 by simp
    moreover from A1 T1 have —u— ≤ GreaterOf(r,—l—,—u—)
      using Order˙ZF˙3˙L2 by simp
    ultimately have —a— ≤ GreaterOf(r,—l—,—u—)
      by (rule Group˙order˙transitive) }
  moreover
  { assume A4: a∉G⁺
    with A2 have T2:
      l∈G —l— ∈ G —a— ∈ G —u— ∈ G a ∈ G-G⁺
      using OrderedGroup˙ZF˙1˙L4 OrderedGroup˙ZF˙3˙L1 apply˙funtype
      by auto
    with A2 have l ∈ G-G⁺ using OrderedGroup˙ZF˙1˙L4D by fast
    with T2 A2 have —a— ≤ —l—
      using OrderedGroup˙ZF˙3˙L3 OrderedGroup˙ZF˙1˙L5
      by simp
    moreover from A1 T2 have —l— ≤ GreaterOf(r,—l—,—u—)
      using Order˙ZF˙3˙L2 by simp
    ultimately have —a— ≤ GreaterOf(r,—l—,—u—)
      by (rule Group˙order˙transitive) }
  ultimately show thesis by blast
qed
```

For linearly ordered groups if a set is bounded then the absolute values are bounded.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L10A:
  assumes A1: r –is total on″ G
  and A2: IsBounded(A,r)
  shows ∃L. ∀a∈A. —a— ≤ L
proof -
  { assume A = 0 then have thesis by auto }
  moreover
  { assume A3: A≠0
    with A2 have ∃u. ∀g∈A. g≤u and ∃l.∀g∈A. l≤g
      using IsBounded˙def IsBoundedAbove˙def IsBoundedBelow˙def
      by auto
    then obtain u l where ∀g∈A. l≤g ∧  g≤u
      by auto
    with A1 have ∀a∈A. —a— ≤ GreaterOf(r,—l—,—u—)
      using OrderedGroup˙ZF˙3˙L10 by simp
    then have thesis by auto }
  ultimately show thesis by blast
qed
```

A slightly more general version of the previous lemma, stating the same fact

for a set defined by separation.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L11:
  assumes r –is total on″ G
  and IsBounded(–b(x).x∈X″,r)
  shows ∃L. ∀x∈X. —b(x)— ≤ L
  using assms OrderedGroup˙ZF˙3˙L10A by blast
```

Absolute values of elements of a finite image of a nonempty set are bounded by an element of the group.

```
lemma (in group3) OrderedGroup˙ZF˙3˙L11A:
  assumes A1: r –is total on″ G
  and A2: X≠0 and A3: –b(x). x∈X″ ∈ Fin(G)
  shows ∃L∈G. ∀x∈X. —b(x)— ≤ L
proof -
  from A1 A3 have ∃L. ∀x∈X. —b(x)— ≤ L
     using  ord˙group˙fin˙bounded OrderedGroup˙ZF˙3˙L11
     by simp
  then obtain L where I: ∀x∈X. —b(x)— ≤ L
    using OrderedGroup˙ZF˙3˙L11 by auto
  from A2 obtain x where x∈X by auto
  with I show thesis using OrderedGroup˙ZF˙1˙L4
    by blast
qed
```

In totally 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
```

34.2 Maximum absolute value of a set

Quite often when considering inequalities we prefer to talk about the absolute values instead of raw elements of a set. This section formalizes some material that is useful for that.

If a set has a maximum and minimum, then the greater of the absolute value of the maximum and minimum belongs to the image of the set by the

absolute value function.

```
lemma (in group3) OrderedGroup˙ZF˙4˙L1:
  assumes A ⊆ G
  and HasAmaximum(r,A) HasAminimum(r,A)
  and M = GreaterOf(r,—Minimum(r,A)—,—Maximum(r,A)—)
  shows M ∈ AbsoluteValue(G,P,r)(A)
  using ordGroupAssum assms IsAnOrdGroup˙def IsPartOrder˙def
    Order˙ZF˙4˙L3 Order˙ZF˙4˙L4 OrderedGroup˙ZF˙3˙L1
    func˙imagedef GreaterOf˙def by auto
```

If a set has a maximum and minimum, then the greater of the absolute value of the maximum and minimum bounds absolute values of all elements of the set.

```
lemma (in group3) OrderedGroup˙ZF˙4˙L2:
  assumes A1: r –is total on″ G
  and A2: HasAmaximum(r,A)  HasAminimum(r,A)
  and A3: a∈A
  shows —a—≤ GreaterOf(r,—Minimum(r,A)—,—Maximum(r,A)—)
proof -
  from ordGroupAssum A2 A3 have
    Minimum(r,A)≤ a a≤ Maximum(r,A)
    using IsAnOrdGroup˙def IsPartOrder˙def Order˙ZF˙4˙L3 Order˙ZF˙4˙L4
    by auto
  with A1 show thesis by (rule OrderedGroup˙ZF˙3˙L10)
qed
```

If a set has a maximum and minimum, then the greater of the absolute value of the maximum and minimum bounds absolute values of all elements of the set. In this lemma the absolute values of ekements of a set are represented as the elements of the image of the set by the absolute value function.

```
lemma (in group3) OrderedGroup˙ZF˙4˙L3:
  assumes r –is total on″ G and A ⊆ G
  and HasAmaximum(r,A) HasAminimum(r,A)
  and b ∈ AbsoluteValue(G,P,r)(A)
  shows b≤ GreaterOf(r,—Minimum(r,A)—,—Maximum(r,A)—)
  using assms OrderedGroup˙ZF˙3˙L1 func˙imagedef OrderedGroup˙ZF˙4˙L2
  by auto
```

If a set has a maximum and minimum, then the set of absolute values also has a maximum.

```
lemma (in group3) OrderedGroup˙ZF˙4˙L4:
  assumes A1: r –is total on″ G and A2: A ⊆ G
  and A3: HasAmaximum(r,A) HasAminimum(r,A)
  shows HasAmaximum(r,AbsoluteValue(G,P,r)(A))
proof -
  let M = GreaterOf(r,—Minimum(r,A)—,—Maximum(r,A)—)
  from A2 A3 have M ∈ AbsoluteValue(G,P,r)(A)
```

```
    using OrderedGroup˙ZF˙4˙L1 by simp
  moreover from A1 A2 A3 have
    ∀b ∈ AbsoluteValue(G,P,r)(A). b ≤ M
    using OrderedGroup˙ZF˙4˙L3 by simp
  ultimately show thesis using HasAmaximum˙def by auto
qed
```

If a set has a maximum and a minimum, then all absolute values are bounded by the maximum of the set of absolute values.

```
lemma (in group3) OrderedGroup˙ZF˙4˙L5:
  assumes A1: r –is total on″ G and A2: A ⊆ G
  and A3: HasAmaximum(r,A) HasAminimum(r,A)
  and A4: a∈A
  shows —a— ≤ Maximum(r,AbsoluteValue(G,P,r)(A))
proof -
  from A2 A4 have —a— ∈ AbsoluteValue(G,P,r)(A)
    using OrderedGroup˙ZF˙3˙L1 func˙imagedef by auto
  with ordGroupAssum A1 A2 A3 show thesis using
    IsAnOrdGroup˙def IsPartOrder˙def OrderedGroup˙ZF˙4˙L4
    Order˙ZF˙4˙L3 by simp
qed
```

34.3 Alternative definitions

Sometimes it is usful to define the order by prescibing the set of positive or nonnegative elements. This section deals with two such definitions. One takes a subset H of G that is closed under the group operation, $1 \notin H$ and for every $a \in H$ we have either $a \in H$ or $a^{-1} \in H$. Then the order is defined as $a \leq b$ iff $a = b$ or $a^{-1}b \in H$. For abelian groups this makes a linearly ordered group. We will refer to order defined this way in the comments as the order defined by a positive set. The context used in this section is the group0 context defined in Group˙ZF theory. Recall that f in that context denotes the group operation (unlike in the previous sections where the group operation was denoted P.

The order defined by a positive set is the same as the order defined by a nonnegative set.

```
lemma (in group0) OrderedGroup˙ZF˙5˙L1:
  assumes A1: r = –p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H″
  shows ⟨a,b⟩ ∈ r  ⟷ a∈G ∧ b∈G ∧ a⁻¹·b ∈ H ∪ –1″
proof
  assume ⟨a,b⟩ ∈ r
  with A1 show a∈G ∧ b∈G ∧ a⁻¹·b ∈ H ∪ –1″
    using group0˙2˙L6 by auto
next assume a∈G ∧ b∈G ∧ a⁻¹·b ∈ H ∪ –1″
   then have a∈G ∧ b∈G ∧ b=(a⁻¹)⁻¹ ∨  a∈G ∧ b∈G ∧ a⁻¹·b ∈ H
    using  inverse˙in˙group group0˙2˙L9 by auto
```

```
  with A1 show ⟨a,b⟩ ∈ r using group˙inv˙of˙inv
    by auto
qed
```

The relation defined by a positive set is antisymmetric.

```
lemma (in group0) OrderedGroup˙ZF˙5˙L2:
  assumes A1: r = –p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H″
  and A2: ∀a∈G. a≠1 ⟶ (a∈H) Xor (a⁻¹∈H)
  shows antisym(r)
proof -
  { fix a b assume A3: ⟨a,b⟩ ∈ r  ⟨b,a⟩ ∈ r
    with A1 have T: a∈G  b∈G by auto
    { assume A4: a≠b
      with A1 A3 have a⁻¹·b ∈ G  a⁻¹·b ∈ H  (a⁻¹·b)⁻¹ ∈ H
 using inverse˙in˙group group0˙2˙L1 monoid0.group0˙1˙L1 group0˙2˙L12
 by auto
      with A2 have a⁻¹·b = 1 using Xor˙def by auto
      with T A4 have False using group0˙2˙L11 by auto
    } then have a=b by auto
  } then show antisym(r) by (rule antisymI)
qed
```

The relation defined by a positive set is transitive.

```
lemma (in group0) OrderedGroup˙ZF˙5˙L3:
  assumes A1: r = –p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H″
  and A2: H⊆G  H –is closed under″ P
  shows trans(r)
proof -
  { fix a b c assume ⟨a,b⟩ ∈ r  ⟨b,c⟩ ∈ r
    with A1 have
      a∈G ∧ b∈G ∧ a⁻¹·b ∈ H ∪ –1″
      b∈G ∧ c∈G ∧ b⁻¹·c ∈ H ∪ –1″
      using OrderedGroup˙ZF˙5˙L1 by auto
    with A2 have
      I: a∈G  b∈G  c∈G
      and (a⁻¹·b)·(b⁻¹·c) ∈  H ∪ –1″
      using inverse˙in˙group group0˙2˙L17 IsOpClosed˙def
      by auto
    moreover from I have a⁻¹·c = (a⁻¹·b)·(b⁻¹·c)
      by (rule group0˙2˙L14A)
    ultimately have ⟨a,c⟩ ∈ G×G  a⁻¹·c  ∈  H ∪ –1″
      by auto
    with A1 have ⟨a,c⟩ ∈ r using OrderedGroup˙ZF˙5˙L1
      by auto
  } then have ∀ a b c. ⟨a, b⟩ ∈ r ∧ ⟨b, c⟩ ∈ r ⟶ ⟨a, c⟩ ∈ r
    by blast
  then show  trans(r) by (rule Fol1˙L2)
qed
```

The relation defined by a positive set is translation invariant. With our

definition this step requires the group to be abelian.

```
lemma (in group0) OrderedGroup˙ZF˙5˙L4:
  assumes A1: r = –p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H″
  and A2: P –is commutative on″ G
  and A3: ⟨a,b⟩ ∈ r  and A4: c∈G
  shows ⟨a·c,b·c⟩ ∈ r ∧ ⟨c·a,c·b⟩ ∈ r
proof
  from A1 A3 A4 have
    I: a∈G  b∈G  a·c ∈ G  b·c ∈ G
    and II: a⁻¹·b ∈ H ∪ –1″
    using OrderedGroup˙ZF˙5˙L1 group˙op˙closed
    by auto
  with A2 A4 have (a·c)⁻¹·(b·c) ∈ H ∪ –1″
    using group0˙4˙L6D by simp
  with A1 I show ⟨a·c,b·c⟩ ∈ r using  OrderedGroup˙ZF˙5˙L1
    by auto
  with A2 A4 I show ⟨c·a,c·b⟩ ∈ r
    using IsCommutative˙def by simp
qed
```

If $H \subseteq G$ is closed under the group operation $1 \notin H$ and for every $a \in H$ we have either $a \in H$ or $a^{-1} \in H$, then the relation "$\leq$" defined by $a \leq b \Leftrightarrow a^{-1}b \in H$ orders the group G. In such order H may be the set of positive or nonnegative elements.

```
lemma (in group0) OrderedGroup˙ZF˙5˙L5:
  assumes A1: P –is commutative on″ G
  and A2: H⊆G  H –is closed under″ P
  and A3: ∀a∈G. a≠1 ⟶ (a∈H) Xor (a⁻¹∈H)
  and A4: r = –p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H″
  shows
  IsAnOrdGroup(G,P,r)
  r –is total on″ G
  Nonnegative(G,P,r) = PositiveSet(G,P,r) ∪ –1″
proof -
  from groupAssum A2 A3 A4 have
    IsAgroup(G,P)  r ⊆ G×G  IsPartOrder(G,r)
    using refl˙def OrderedGroup˙ZF˙5˙L2 OrderedGroup˙ZF˙5˙L3
      IsPartOrder˙def by auto
  moreover from A1 A4 have
    ∀g∈G. ∀a b. ⟨ a,b⟩ ∈ r ⟶ ⟨a·g,b·g⟩ ∈ r ∧ ⟨g·a,g·b⟩ ∈ r
    using OrderedGroup˙ZF˙5˙L4 by blast
  ultimately show IsAnOrdGroup(G,P,r)
    using IsAnOrdGroup˙def by simp
  then show Nonnegative(G,P,r) = PositiveSet(G,P,r) ∪ –1″
    using group3˙def group3.OrderedGroup˙ZF˙1˙L24
    by simp
  { fix a b
    assume T: a∈G  b∈G
    then have T1: a⁻¹·b ∈ G
```

```
    using inverse˙in˙group group˙op˙closed by simp
  { assume ⟨ a,b⟩ ∉ r
    with A4 T have I: a≠b and II: a⁻¹·b ∉ H
 by auto
    from A3 T T1 I have (a⁻¹·b ∈ H) Xor ((a⁻¹·b)⁻¹ ∈ H)
 using group0˙2˙L11 by auto
    with A4 T II have ⟨ b,a⟩ ∈ r
 using Xor˙def group0˙2˙L12 by simp
  } then have ⟨ a,b⟩ ∈ r ∨ ⟨ b,a⟩ ∈ r by auto
 } then show r –is total on″ G using IsTotal˙def
  by simp
qed
```

If the set defined as in OrderedGroup˙ZF˙5˙L4 does not contain the neutral element, then it is the positive set for the resulting order.

```
lemma (in group0) OrderedGroup˙ZF˙5˙L6:
 assumes P –is commutative on″ G
 and H⊆G and 1 ∉ H
 and r = –p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H″
 shows PositiveSet(G,P,r) = H
 using assms group˙inv˙of˙one group0˙2˙L2 PositiveSet˙def
 by auto
```

The next definition describes how we construct an order relation from the prescribed set of positive elements.

```
definition
 OrderFromPosSet(G,P,H) ≡
 –p ∈ G×G. fst(p) = snd(p) ∨ P⟨GroupInv(G,P)(fst(p)),snd(p)⟩ ∈ H ″
```

The next theorem rephrases lemmas OrderedGroup˙ZF˙5˙L5 and OrderedGroup˙ZF˙5˙L6 using the definition of the order from the positive set OrderFromPosSet. To summarize, this is what it says: Suppose that $H \subseteq G$ is a set closed under that group operation such that $1 \notin H$ and for every nonunit group element a either $a \in H$ or $a^{-1} \in H$. Define the order as $a \leq b$ iff $a = b$ or $a^{-1} \cdot b \in H$. Then this order makes G into a linearly ordered group such H is the set of positive elements (and then of course $H \cup \{1\}$ is the set of nonnegative elements).

```
theorem (in group0) Group˙ord˙by˙positive˙set:
 assumes P –is commutative on″ G
 and H⊆G   H –is closed under″ P   1 ∉ H
 and ∀a∈G. a≠1 ⟶ (a∈H) Xor (a⁻¹∈H)
 shows
 IsAnOrdGroup(G,P,OrderFromPosSet(G,P,H))
 OrderFromPosSet(G,P,H) –is total on″ G
 PositiveSet(G,P,OrderFromPosSet(G,P,H)) = H
 Nonnegative(G,P,OrderFromPosSet(G,P,H)) = H ∪ –1″
 using assms OrderFromPosSet˙def OrderedGroup˙ZF˙5˙L5 OrderedGroup˙ZF˙5˙L6
 by auto
```

34.4 Odd Extensions

In this section we verify properties of odd extensions of functions defined on G_+. An odd extension of a function $f : G_+ \to G$ is a function $f^\circ : G \to G$ defined by $f^\circ(x) = f(x)$ if $x \in G_+$, $f(1) = 1$ and $f^\circ(x) = (f(x^{-1}))^{-1}$ for $x < 1$. Such function is the unique odd function that is equal to f when restricted to G_+.

The next lemma is just to see the definition of the odd extension in the notation used in the group1 context.

```
lemma (in group3) OrderedGroup˙ZF˙6˙L1:
  shows f° = f ∪ –⟨a, (f(a⁻¹))⁻¹⟩. a ∈ -G₊″ ∪ –⟨1,1⟩″
  using OddExtension˙def by simp
```

A technical lemma that states that from a function defined on G_+ with values in G we have $(f(a^{-1}))^{-1} \in G$.

```
lemma (in group3) OrderedGroup˙ZF˙6˙L2:
  assumes f: G₊→G and a∈-G₊
  shows
  f(a⁻¹) ∈ G
  (f(a⁻¹))⁻¹ ∈ G
  using assms OrderedGroup˙ZF˙1˙L27 apply˙funtype
    OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group
  by auto
```

The main theorem about odd extensions. It basically says that the odd extension of a function is what we want to to be.

```
lemma (in group3) odd˙ext˙props:
  assumes A1: r –is total on″ G and A2: f: G₊→G
  shows
  f° : G → G
  ∀a∈G₊. (f°)(a) = f(a)
  ∀a∈(-G₊). (f°)(a) = (f(a⁻¹))⁻¹
  (f°)(1) = 1
proof -
  from A1 A2 have I:
    f: G₊→G
    ∀a∈-G₊. (f(a⁻¹))⁻¹ ∈ G
    G₊∩(-G₊) = 0
    1 ∉ G₊∪(-G₊)
    f° = f ∪ –⟨a, (f(a⁻¹))⁻¹⟩. a ∈ -G₊″ ∪ –⟨1,1⟩″
    using OrderedGroup˙ZF˙6˙L2 OrdGroup˙decomp2 OrderedGroup˙ZF˙6˙L1
    by auto
  then have f°: G₊ ∪ (-G₊) ∪ –1″ →G∪G∪–1″
    by (rule func1˙1˙L11E)
  moreover from A1 have
    G₊ ∪ (-G₊) ∪ –1″ = G
    G∪G∪–1″ = G
```

```
    using OrdGroup˙decomp2 OrderedGroup˙ZF˙1˙L1 group0.group0˙2˙L2
    by auto
  ultimately show f° : G → G by simp
  from I show ∀a∈G+. (f°)(a) = f(a)
    by (rule func1˙1˙L11E)
  from I show ∀a∈(-G+). (f°)(a) = (f(a⁻¹))⁻¹
    by (rule func1˙1˙L11E)
  from I show (f°)(1) = 1
    by (rule func1˙1˙L11E)
qed
```

Odd extensions are odd, of course.

```
lemma (in group3) oddext˙is˙odd:
  assumes A1: r –is total on″ G and A2: f: G+→G
  and A3: a∈G
  shows (f°)(a⁻¹) = ((f°)(a))⁻¹
proof -
  from A1 A3 have a∈G+ ∨ a ∈ (-G+) ∨ a=1
    using OrdGroup˙decomp2 by blast
  moreover
  { assume a∈G+
    with A1 A2 have a⁻¹ ∈ -G+ and  (f°)(a) = f(a)
      using OrderedGroup˙ZF˙1˙L25 odd˙ext˙props by auto
    with A1 A2 have
      (f°)(a⁻¹) = (f((a⁻¹)⁻¹))⁻¹  and (f(a))⁻¹ = ((f°)(a))⁻¹
      using odd˙ext˙props by auto
    with A3 have (f°)(a⁻¹) = ((f°)(a))⁻¹
      using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
      by simp }
  moreover
  { assume A4: a ∈ -G+
    with A1 A2  have a⁻¹ ∈ G+ and  (f°)(a) = (f(a⁻¹))⁻¹
      using OrderedGroup˙ZF˙1˙L27 odd˙ext˙props
      by auto
    with A1 A2 A4 have (f°)(a⁻¹) = ((f°)(a))⁻¹
      using odd˙ext˙props OrderedGroup˙ZF˙6˙L2
 OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
      by simp }
  moreover
  { assume a = 1
    with A1 A2 have (f°)(a⁻¹) = ((f°)(a))⁻¹
      using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙one
 odd˙ext˙props by simp
  }
  ultimately show (f°)(a⁻¹) = ((f°)(a))⁻¹
    by auto
qed
```

Another way of saying that odd extensions are odd.

```
lemma (in group3) oddext˙is˙odd˙alt:
  assumes A1: r –is total on″ G and A2: f: G+→G
  and A3: a∈G
  shows ((f°)(a⁻¹))⁻¹ = (f°)(a)
proof -
  from A1 A2 have
    f° : G → G
    ∀a∈G. (f°)(a⁻¹) = ((f°)(a))⁻¹
    using odd˙ext˙props oddext˙is˙odd by auto
  then have ∀a∈G. ((f°)(a⁻¹))⁻¹ = (f°)(a)
    using OrderedGroup˙ZF˙1˙L1 group0.group0˙6˙L2 by simp
  with A3 show ((f°)(a⁻¹))⁻¹ = (f°)(a) by simp
qed
```

34.5 Functions with infinite limits

In this section we consider functions $f : G \to G$ with the property that for $f(x)$ is arbitrarily large for large enough x. More precisely, for every $a \in G$ there exist $b \in G_+$ such that for every $x \geq b$ we have $f(x) \geq a$. In a sense this means that $\lim_{x\to\infty} f(x) = \infty$, hence the title of this section. We also prove dual statements for functions such that $\lim_{x\to-\infty} f(x) = -\infty$.

If an image of a set by a function with infinite positive limit is bounded above, then the set itself is bounded above.

```
lemma (in group3) OrderedGroup˙ZF˙7˙L1:
  assumes A1: r –is total on″ G and A2: G ≠ –1″ and
  A3: f:G→G and
  A4: ∀a∈G.∃b∈G+.∀x. b≤x ⟶ a ≤ f(x) and
  A5: A⊆G and
  A6: IsBoundedAbove(f(A),r)
  shows IsBoundedAbove(A,r)
proof -
  { assume ¬IsBoundedAbove(A,r)
    then have I: ∀u. ∃x∈A. ¬(x≤u)
      using IsBoundedAbove˙def by auto
    have ∀a∈G. ∃y∈f(A). a≤y
    proof -
      { fix a assume a∈G
 with A4 obtain b where
   II: b∈G+ and III: ∀x. b≤x ⟶ a ≤ f(x)
   by auto
 from I obtain x where IV: x∈A and ¬(x≤b)
   by auto
 with A1 A5 II have
   r –is total on″ G
   x∈G  b∈G  ¬(x≤b)
   using PositiveSet˙def by auto
 with III have a ≤ f(x)
```

```
    using OrderedGroup˙ZF˙1˙L8 by blast
  with A3 A5 IV have ∃y∈f(A). a≤y
    using func˙imagedef by auto
      } thus thesis by simp
    qed
    with A1 A2 A6 have False using OrderedGroup˙ZF˙2˙L2A
      by simp
  } thus thesis by auto
qed
```

If an image of a set defined by separation by a function with infinite positive limit is bounded above, then the set itself is bounded above.

```
lemma (in group3) OrderedGroup˙ZF˙7˙L2:
  assumes A1: r –is total on″ G and A2: G ≠ –1″ and
  A3: X≠0 and A4: f:G→G and
  A5: ∀a∈G.∃b∈G+.∀y. b≤y ⟶ a ≤ f(y) and
  A6: ∀x∈X. b(x) ∈ G ∧ f(b(x)) ≤ U
  shows ∃u.∀x∈X. b(x) ≤ u
proof -
  let A = –b(x). x∈X″
  from A6 have I: A⊆G by auto
  moreover note assms
  moreover have IsBoundedAbove(f(A),r)
  proof -
    from A4 A6 I have ∀z∈f(A). ⟨z,U⟩ ∈ r
      using func˙imagedef by simp
    then show IsBoundedAbove(f(A),r)
      by (rule Order˙ZF˙3˙L10)
  qed
  ultimately have IsBoundedAbove(A,r) using OrderedGroup˙ZF˙7˙L1
    by simp
  with A3 have ∃u.∀y∈A. y ≤ u
    using IsBoundedAbove˙def by simp
  then show ∃u.∀x∈X. b(x) ≤ u by auto
qed
```

If the image of a set defined by separation by a function with infinite negative limit is bounded below, then the set itself is bounded above. This is dual to OrderedGroup˙ZF˙7˙L2.

```
lemma (in group3) OrderedGroup˙ZF˙7˙L3:
  assumes A1: r –is total on″ G and A2: G ≠ –1″ and
  A3: X≠0 and A4: f:G→G and
  A5: ∀a∈G.∃b∈G+.∀y. b≤y ⟶ f(y⁻¹) ≤ a and
  A6: ∀x∈X. b(x) ∈ G ∧ L ≤ f(b(x))
  shows ∃l.∀x∈X. l ≤ b(x)
proof -
  let g = GroupInv(G,P) O f O GroupInv(G,P)
  from ordGroupAssum have I: GroupInv(G,P) : G→G
    using IsAnOrdGroup˙def group0˙2˙T2 by simp
```

```
  with A4 have II: ∀x∈G. g(x) = (f(x⁻¹))⁻¹
    using func1˙1˙L18 by simp
  note A1 A2 A3
  moreover from A4 I have g : G→G
    using comp˙fun by blast
  moreover have ∀a∈G.∃b∈G₊.∀y. b≤y ⟶ a ≤ g(y)
  proof -
  { fix a assume A7: a∈G
    then have a⁻¹ ∈ G
      using OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group
      by simp
    with A5 obtain b where
      III: b∈G₊ and ∀y. b≤y ⟶ f(y⁻¹) ≤ a⁻¹
      by auto
    with II A7 have ∀y. b≤y ⟶ a ≤ g(y)
      using OrderedGroup˙ZF˙1˙L5AD OrderedGroup˙ZF˙1˙L4
      by simp
    with III have ∃b∈G₊.∀y. b≤y ⟶ a ≤ g(y)
      by auto
  } then show ∀a∈G.∃b∈G₊.∀y. b≤y ⟶ a ≤ g(y)
    by simp
  qed
  moreover have ∀x∈X. b(x)⁻¹ ∈ G ∧ g(b(x)⁻¹) ≤ L⁻¹
  proof-
    { fix x assume x∈X
      with A6 have
 T: b(x) ∈ G  b(x)⁻¹ ∈ G and L ≤ f(b(x))
 using OrderedGroup˙ZF˙1˙L1 group0.inverse˙in˙group
 by auto
      then have (f(b(x)))⁻¹ ≤ L⁻¹
 using OrderedGroup˙ZF˙1˙L5 by simp
      moreover from II T have (f(b(x)))⁻¹ =  g(b(x)⁻¹)
 using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
 by simp
      ultimately have g(b(x)⁻¹) ≤ L⁻¹ by simp
      with T have b(x)⁻¹ ∈ G ∧ g(b(x)⁻¹) ≤ L⁻¹
 by simp
    } then show ∀x∈X. b(x)⁻¹ ∈ G ∧ g(b(x)⁻¹) ≤ L⁻¹
      by simp
  qed
  ultimately have ∃u.∀x∈X. (b(x))⁻¹ ≤ u
    by (rule OrderedGroup˙ZF˙7˙L2)
  then have ∃u.∀x∈X. u⁻¹ ≤ (b(x)⁻¹)⁻¹
    using OrderedGroup˙ZF˙1˙L5 by auto
  with A6 show ∃l.∀x∈X. l ≤ b(x)
    using OrderedGroup˙ZF˙1˙L1 group0.group˙inv˙of˙inv
    by auto
qed
```

The next lemma combines OrderedGroup˙ZF˙7˙L2 and OrderedGroup˙ZF˙7˙L3

to show that if an image of a set defined by separation by a function with infinite limits is bounded, then the set itself i bounded.

lemma (in group3) OrderedGroup˙ZF˙7˙L4:
  assumes A1: r –is total on″ G and A2: G ≠ –**1**″ and
  A3: X≠0 and A4: f:G→G and
  A5: ∀a∈G.∃b∈G_+.∀y. b≤y ⟶ a ≤ f(y) and
  A6: ∀a∈G.∃b∈G_+.∀y. b≤y ⟶ f(y^{-1}) ≤ a and
  A7: ∀x∈X. b(x) ∈ G ∧ L ≤ f(b(x)) ∧ f(b(x)) ≤ U
shows ∃M.∀x∈X. —b(x)— ≤ M
proof -
  from A7 have
    I: ∀x∈X. b(x) ∈ G ∧ f(b(x)) ≤ U and
    II: ∀x∈X. b(x) ∈ G ∧ L ≤ f(b(x))
    by auto
  from A1 A2 A3 A4 A5 I have ∃u.∀x∈X. b(x) ≤ u
    by (rule OrderedGroup˙ZF˙7˙L2)
  moreover from A1 A2 A3 A4 A6 II have ∃l.∀x∈X. l ≤ b(x)
    by (rule OrderedGroup˙ZF˙7˙L3)
  ultimately have ∃u l. ∀x∈X. l≤b(x) ∧ b(x) ≤ u
    by auto
  with A1 have ∃u l.∀x∈X. —b(x)— ≤ GreaterOf(r,—l—,—u—)
    using OrderedGroup˙ZF˙3˙L10 by blast
  then show ∃M.∀x∈X. —b(x)— ≤ M
    by auto
qed

end

35 Rings - introduction

theory Ring˙ZF imports AbelianGroup˙ZF

begin

This theory file covers basic facts about rings.

35.1 Definition and basic properties

In this section we define what is a ring and list the basic properties of rings.

We say that three sets (R, A, M) form a ring if (R, A) is an abelian group, (R, M) is a monoid and A is distributive with respect to M on R. A represents the additive operation on R. As such it is a subset of $(R \times R) \times R$ (recall that in ZF set theory functions are sets). Similarly M represents the multiplicative operation on R and is also a subset of $(R \times R) \times R$. We don't require the multiplicative operation to be commutative in the definition of a ring.

definition
IsAring(R,A,M) ≡ IsAgroup(R,A) ∧ (A –is commutative on″ R) ∧
IsAmonoid(R,M) ∧ IsDistributive(R,A,M)

We also define the notion of having no zero divisors. In standard notation the ring has no zero divisors if for all $a, b \in R$ we have $a \cdot b = 0$ implies $a = 0$ or $b = 0$.

definition
HasNoZeroDivs(R,A,M) ≡ (∀ a∈R. ∀ b∈R.
M⟨ a,b⟩ = TheNeutralElement(R,A) ⟶
a = TheNeutralElement(R,A) ∨ b = TheNeutralElement(R,A))

Next we define a locale that will be used when considering rings.

locale ring0 =

fixes R and A and M

assumes ringAssum: IsAring(R,A,M)

fixes ringa (infixl + 90)
defines ringa˙def [simp]: 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]: a^2 ≡ 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 assms IsAring˙def IsDistributive˙def by auto

Zero and one of the ring are elements of the ring. The negative of zero is zero.

lemma (in ring0) Ring˙ZF˙1˙L2:
shows **0**∈R **1**∈R (-**0**) = **0**
using Ring˙ZF˙1˙L1 group0.group0˙2˙L2 monoid0.unit˙is˙neutral
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 assms Ring˙ZF˙1˙L1 group0.inverse˙in˙group group0.group˙inv˙of˙inv
group0.group0˙2˙L6 group0.group0˙2˙L2 monoid0.unit˙is˙neutral
Ring˙ZF˙1˙L2 ring˙oper˙distr
by auto

Properties that require two elements of a ring.

lemma (in ring0) Ring˙ZF˙1˙L4: assumes A1: a∈R b∈R
shows
a+b ∈ R
a-b ∈ R
a·b ∈ R
a+b = b+a
using ringAssum assms Ring˙ZF˙1˙L1 Ring˙ZF˙1˙L3
group0.group0˙2˙L1 monoid0.group0˙1˙L1
IsAring˙def IsCommutative˙def
by auto

Cancellation of an element on both sides of equality. This is a property of groups, written in the (additive) notation we use for the additive operation in rings.

```
lemma (in ring0) ring˙cancel˙add:
  assumes A1: a∈R b∈R and A2: a + b = a
  shows b = 0
  using assms Ring˙ZF˙1˙L1 group0.group0˙2˙L7 by simp
```

Any element of a ring multiplied by zero is zero.

```
lemma (in ring0) Ring˙ZF˙1˙L6:
  assumes A1: x∈R shows 0·x = 0   x·0 = 0
proof -
  let a = x·1
  let b = x·0
  let c = 1·x
  let d = 0·x
  from A1 have
    a + b = x·(1 + 0)   c + d = (1 + 0)·x
    using Ring˙ZF˙1˙L2 ring˙oper˙distr by auto
  moreover have x·(1 + 0) = a (1 + 0)·x = c
    using Ring˙ZF˙1˙L2 Ring˙ZF˙1˙L3 by auto
  ultimately have a + b = a and T1: c + d = c
    by auto
  moreover from A1 have
    a ∈ R  b ∈ R and T2: c ∈ R  d ∈ R
    using Ring˙ZF˙1˙L2 Ring˙ZF˙1˙L4 by auto
  ultimately have b = 0 using ring˙cancel˙add
    by blast
  moreover from T2 T1 have d = 0 using ring˙cancel˙add
    by blast
  ultimately show x·0 = 0  0·x = 0 by auto
qed
```

Negative can be pulled out of a product.

```
lemma (in ring0) Ring˙ZF˙1˙L7:
  assumes A1: a∈R  b∈R
  shows
  (-a)·b = -(a·b)
  a·(-b) = -(a·b)
  (-a)·b = a·(-b)
proof -
  from A1 have I:
    a·b ∈ R (-a) ∈ R ((-a)·b) ∈ R
    (-b) ∈ R a·(-b) ∈ R
    using Ring˙ZF˙1˙L3 Ring˙ZF˙1˙L4 by auto
  moreover have (-a)·b + a·b = 0
    and II: a·(-b) + a·b = 0
  proof -
```

```
    from A1 I have
      (-a)·b + a·b = ((-a)+ a)·b
      a·(-b) + a·b= a·((-b)+b)
      using ring˙oper˙distr by auto
    moreover from A1 have
      ((-a)+ a)·b = 0
      a·((-b)+b) = 0
      using 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 assms Ring˙ZF˙1˙L3 Ring˙ZF˙1˙L7 Ring˙ZF˙1˙L4
  by simp
```

Subtraction is distributive with respect to multiplication.

```
lemma (in ring0) Ring˙ZF˙1˙L8: assumes a∈R  b∈R  c∈R
  shows
  a·(b-c) = a·b - a·c
  (b-c)·a = b·a - c·a
  using assms Ring˙ZF˙1˙L3 ring˙oper˙distr Ring˙ZF˙1˙L7 Ring˙ZF˙1˙L4
  by auto
```

Other basic properties involving two elements of a ring.

```
lemma (in ring0) Ring˙ZF˙1˙L9: assumes a∈R  b∈R
  shows
  (-b)-a = (-a)-b
  (-(a+b)) = (-a)-b
  (-(a-b)) = ((-a)+b)
  a-(-b) = a+b
  using assms 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 assms 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 assms Ring˙ZF˙1˙L3 Ring˙ZF˙1˙L10 by simp
```

Associativity of addition and multiplication.

```
lemma (in ring0) Ring˙ZF˙1˙L11:
  assumes a∈R  b∈R  c∈R
  shows
  a+b+c = a+(b+c)
  a·b·c = a·(b·c)
  using assms 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 assms HasNoZeroDivs˙def by auto
```

In rings with no zero divisors we can cancel nonzero factors.

```
lemma (in ring0) Ring˙ZF˙1˙L12A:
  assumes A1: HasNoZeroDivs(R,A,M) and A2: a∈R  b∈R  c∈R
```

```
  and A3: a·c = b·c and A4: c≠0
  shows a=b
proof -
  from A2 have T: a·c ∈ R  a-b ∈ R
    using Ring˙ZF˙1˙L4 by auto
  with A1 A2 A3 have a-b = 0 ∨ c=0
    using Ring˙ZF˙1˙L3 Ring˙ZF˙1˙L8 HasNoZeroDivs˙def
    by simp
  with A2 A4 have a∈R  b∈R  a-b = 0
    by auto
  then show a=b by (rule Ring˙ZF˙1˙L9A)
qed
```

In rings with no zero divisors if two elements are different, then after multiplying by a nonzero element they are still different.

```
lemma (in ring0) Ring˙ZF˙1˙L12B:
  assumes A1: HasNoZeroDivs(R,A,M)
  a∈R  b∈R  c∈R  a≠b  c≠0
  shows  a·c ≠ b·c
  using A1 Ring˙ZF˙1˙L12A by auto
```

In rings with no zero divisors multiplying a nonzero element by a nonone element changes the value.

```
lemma (in ring0) Ring˙ZF˙1˙L12C:
  assumes A1: HasNoZeroDivs(R,A,M) and
  A2: a∈R  b∈R and A3: 0≠a  1≠b
  shows a ≠ a·b
proof -
  { assume a = a·b
    with A1 A2 have a = 0 ∨ b-1 = 0
      using Ring˙ZF˙1˙L3 Ring˙ZF˙1˙L2 Ring˙ZF˙1˙L8
 Ring˙ZF˙1˙L3 Ring˙ZF˙1˙L2 Ring˙ZF˙1˙L4 HasNoZeroDivs˙def
      by simp
    with A2 A3 have False
      using Ring˙ZF˙1˙L2 Ring˙ZF˙1˙L9A by auto
  } then show a ≠ a·b by auto
qed
```

If a square is nonzero, then the element is nonzero.

```
lemma (in ring0) Ring˙ZF˙1˙L13:
  assumes a∈R  and a^2 ≠ 0
  shows a≠0
  using assms Ring˙ZF˙1˙L2 Ring˙ZF˙1˙L6 by auto
```

Square of an element and its opposite are the same.

```
lemma (in ring0) Ring˙ZF˙1˙L14:
  assumes a∈R shows (-a)^2 = ((a)^2)
  using assms Ring˙ZF˙1˙L7A by simp
```

Adding zero to a set that is closed under addition results in a set that is also closed under addition. This is a property of groups.

```
lemma (in ring0) Ring˙ZF˙1˙L15:
  assumes H ⊆ R and H –is closed under″ A
  shows (H ∪ –0″) –is closed under″ A
  using assms 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 assms Ring˙ZF˙1˙L2 Ring˙ZF˙1˙L6 IsOpClosed˙def
  by auto
```

The ring is trivial iff $0 = 1$.

```
lemma (in ring0) Ring˙ZF˙1˙L17: shows R = –0″ ⟷ 0=1
proof
  assume R = –0″
  then show 0=1 using Ring˙ZF˙1˙L2
    by blast
next assume A1: 0 = 1
  then have R ⊆ –0″
    using Ring˙ZF˙1˙L3 Ring˙ZF˙1˙L6 by auto
  moreover have –0″ ⊆ R using Ring˙ZF˙1˙L2 by auto
  ultimately show R = –0″ by auto
qed
```

The sets $\{m \cdot x.x \in R\}$ and $\{-m \cdot x.x \in R\}$ are the same.

```
lemma (in ring0) Ring˙ZF˙1˙L18: assumes A1: m∈R
  shows –m·x. x∈R″ = –(-m)·x. x∈R″
proof
  { fix a assume a ∈ –m·x. x∈R″
    then obtain x where x∈R and a = m·x
      by auto
    with A1 have (-x) ∈ R  and a = (-m)·(-x)
      using Ring˙ZF˙1˙L3 Ring˙ZF˙1˙L7A by auto
    then have a ∈ –(-m)·x. x∈R″
      by auto
  } then show –m·x. x∈R″ ⊆ –(-m)·x. x∈R″
    by auto
next
  { fix a assume a ∈ –(-m)·x. x∈R″
    then obtain x where x∈R and a = (-m)·x
      by auto
    with A1 have (-x) ∈ R and a = m·(-x)
      using Ring˙ZF˙1˙L3 Ring˙ZF˙1˙L7 by auto
    then have a ∈ –m·x. x∈R″ by auto
```

```
  } then show –(-m)·x. x∈R″ ⊆ –m·x. x∈R″
    by auto
qed
```

35.2 Rearrangement lemmas

In happens quite often that we want to show a fact like $(a + b)c + d = (ac + d - e) + (bc + e)$in rings. This is trivial in romantic math and probably there is a way to make it trivial in formalized math. However, I don't know any other way than to tediously prove each such rearrangement when it is needed. This section collects facts of this type.

Rearrangements with two elements of a ring.

```
lemma (in ring0) Ring˙ZF˙2˙L1: assumes a∈R b∈R
  shows a+b·a = (b+1)·a
  using assms Ring˙ZF˙1˙L2 ring˙oper˙distr Ring˙ZF˙1˙L3 Ring˙ZF˙1˙L4
  by simp
```

Rearrangements with two elements and cancelling.

```
lemma (in ring0) Ring˙ZF˙2˙L1A: assumes a∈R b∈R
  shows
  a-b+b = a
  a+b-a = b
  (-a)+b+a = b
  (-a)+(b+a) = b
  a+(b-a) = b
  using assms Ring˙ZF˙1˙L1 group0.inv˙cancel˙two 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 -
  let B = b·c
  from ringAssum have A –is commutative on″ R
    using IsAring˙def by simp
  moreover from A1 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 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 assms 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 assms 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
```

36 More on rings

theory Ring˙ZF˙1 imports Ring˙ZF Group˙ZF˙3

begin

This theory is devoted to the part of ring theory specific the construction of real numbers in the Real˙ZF˙x series of theories. The goal is to show that classes of almost homomorphisms form a ring.

36.1 The ring of classes of almost homomorphisms

Almost homomorphisms do not form a ring as the regular homomorphisms do because the lifted group operation is not distributive with respect to composition – we have $s \circ (r \cdot q) \neq s \circ r \cdot s \circ q$ in general. However, we do have $s \circ (r \cdot q) \approx s \circ r \cdot s \circ q$ in the sense of the equivalence relation defined by the group of finite range functions (that is a normal subgroup of almost homomorphisms, if the group is abelian). This allows to define a natural ring structure on the classes of almost homomorphisms.

The next lemma provides a formula useful for proving that two sides of the distributive law equation for almost homomorphisms are almost equal.

```
lemma (in group1) Ring˙ZF˙1˙1˙L1:
  assumes A1: s∈AH r∈AH q∈AH and A2: n∈G
  shows
  ((s∘(r•q))(n))·(((s∘r)•(s∘q))(n))^-1= δ(s,⟨ r(n),q(n)⟩)
  ((r•q)∘s)(n) = ((r∘s)•(q∘s))(n)
proof -
  from groupAssum isAbelian A1 have T1:
    r•q ∈ AH s∘r ∈ AH  s∘q ∈ AH (s∘r)•(s∘q) ∈ AH
    r∘s ∈ AH q∘s ∈ AH  (r∘s)•(q∘s) ∈ AH
    using Group˙ZF˙3˙2˙L15 Group˙ZF˙3˙4˙T1 by auto
  from A1 A2 have T2: r(n) ∈ G q(n) ∈ G s(n) ∈ G
    s(r(n)) ∈ G s(q(n)) ∈ G δ(s,⟨ r(n),q(n)⟩) ∈ G
    s(r(n))·s(q(n)) ∈ G r(s(n)) ∈ G q(s(n)) ∈ G
    r(s(n))·q(s(n)) ∈ G
    using AlmostHoms˙def apply˙funtype Group˙ZF˙3˙2˙L4B
```

group0˙2˙L1 monoid0.group0˙1˙L1 by auto
with T1 A1 A2 isAbelian show
((s∘(r•q))(n))·(((s∘r)•(s∘q))(n))$^{-1}$= δ(s,⟨ r(n),q(n)⟩)
((r•q)∘s)(n) = ((r∘s)•(q∘s))(n)
using Group˙ZF˙3˙2˙L12 Group˙ZF˙3˙4˙L2 Group˙ZF˙3˙4˙L1 group0˙4˙L6A
by auto
qed

The sides of the distributive law equations for almost homomorphisms are almost equal.

lemma (in group1) Ring˙ZF˙1˙1˙L2:
assumes A1: s∈AH r∈AH q∈AH
shows
s∘(r•q) ≈ (s∘r)•(s∘q)
(r•q)∘s = (r∘s)•(q∘s)
proof -
from A1 have ∀n∈G. ⟨ r(n),q(n)⟩ ∈ G×G
using AlmostHoms˙def apply˙funtype by auto
moreover from A1 have −δ(s,x). x ∈ G×G″ ∈ Fin(G)
using AlmostHoms˙def by simp
ultimately have −δ(s,⟨ r(n),q(n)⟩). n∈G″ ∈ Fin(G)
by (rule Finite1˙L6B)
with A1 have
−((s∘(r•q))(n))·(((s∘r)•(s∘q))(n))$^{-1}$. n ∈ G″ ∈ Fin(G)
using Ring˙ZF˙1˙1˙L1 by simp
moreover from groupAssum isAbelian A1 A1 have
s∘(r•q) ∈ AH (s∘r)•(s∘q) ∈ AH
using Group˙ZF˙3˙2˙L15 Group˙ZF˙3˙4˙T1 by auto
ultimately show s∘(r•q) ≈ (s∘r)•(s∘q)
using Group˙ZF˙3˙4˙L12 by simp
from groupAssum isAbelian A1 have
(r•q)∘s : G→G (r∘s)•(q∘s) : G→G
using Group˙ZF˙3˙2˙L15 Group˙ZF˙3˙4˙T1 AlmostHoms˙def
by auto
moreover from A1 have
∀n∈G. ((r•q)∘s)(n) = ((r∘s)•(q∘s))(n)
using Ring˙ZF˙1˙1˙L1 by simp
ultimately show (r•q)∘s = (r∘s)•(q∘s)
using fun˙extension˙iff by simp
qed

The essential condition to show the distributivity for the operations defined on classes of almost homomorphisms.

lemma (in group1) Ring˙ZF˙1˙1˙L3:
assumes A1: R = QuotientGroupRel(AH,Op1,FR)
and A2: a ∈ AH//R b ∈ AH//R c ∈ AH//R
and A3: A = ProjFun2(AH,R,Op1) M = ProjFun2(AH,R,Op2)
shows M⟨a,A⟨ b,c⟩⟩ = A⟨M⟨ a,b⟩,M⟨ a,c⟩⟩ ∧
M⟨A⟨ b,c⟩,a⟩ = A⟨M⟨ b,a⟩,M⟨ c,a⟩⟩

```
proof
  from A2 obtain s q r where D1: s∈AH r∈AH q∈AH
    a = R–s″ b = R–q″ c = R–r″
    using quotient˙def by auto
  from A1 have T1:equiv(AH,R)
      using Group˙ZF˙3˙3˙L3 by simp
  with A1 A3 D1 groupAssum isAbelian have
    M⟨ a,A⟨ b,c⟩ ⟩ = R–s∘(q•r)″
    using Group˙ZF˙3˙3˙L4 EquivClass˙1˙L10
    Group˙ZF˙3˙2˙L15 Group˙ZF˙3˙4˙L13A by simp
  also have R–s∘(q•r)″ = R–(s∘q)•(s∘r)″
  proof -
    from T1 D1 have equiv(AH,R) s∘(q•r)≈(s∘q)•(s∘r)
      using Ring˙ZF˙1˙1˙L2 by auto
    with A1 show thesis using equiv˙class˙eq by simp
  qed
  also from A1 T1 D1 A3 have
    R–(s∘q)•(s∘r)″ = A⟨M⟨ a,b⟩,M⟨ a,c⟩⟩
    using Group˙ZF˙3˙3˙L4 Group˙ZF˙3˙4˙T1 EquivClass˙1˙L10
    Group˙ZF˙3˙3˙L3 Group˙ZF˙3˙4˙L13A EquivClass˙1˙L10 Group˙ZF˙3˙4˙T1
    by simp
  finally show M⟨a,A⟨ b,c⟩⟩ = A⟨M⟨ a,b⟩,M⟨ a,c⟩⟩ by simp
  from A1 A3 T1 D1 groupAssum isAbelian show
    M⟨A⟨ b,c⟩,a⟩ = A⟨M⟨ b,a⟩,M⟨ c,a⟩⟩
    using Group˙ZF˙3˙3˙L4 EquivClass˙1˙L10 Group˙ZF˙3˙4˙L13A
      Group˙ZF˙3˙2˙L15 Ring˙ZF˙1˙1˙L2 Group˙ZF˙3˙4˙T1 by simp
qed
```

The projection of the first group operation on almost homomorphisms is distributive with respect to the second group operation.

```
lemma (in group1) Ring˙ZF˙1˙1˙L4:
  assumes A1: R = QuotientGroupRel(AH,Op1,FR)
  and A2: A = ProjFun2(AH,R,Op1) M = ProjFun2(AH,R,Op2)
  shows IsDistributive(AH//R,A,M)
proof -
  from A1 A2 have ∀a∈(AH//R).∀b∈(AH//R).∀c∈(AH//R).
  M⟨a,A⟨ b,c⟩⟩ = A⟨M⟨ a,b⟩, M⟨ a,c⟩⟩ ∧
  M⟨A⟨ b,c⟩, a⟩ = A⟨M⟨ b,a⟩,M⟨ c,a⟩⟩
    using Ring˙ZF˙1˙1˙L3 by simp
  then show thesis using IsDistributive˙def by simp
qed
```

The classes of almost homomorphisms form a ring.

```
theorem (in group1) Ring˙ZF˙1˙1˙T1:
  assumes R = QuotientGroupRel(AH,Op1,FR)
  and A = ProjFun2(AH,R,Op1) M = ProjFun2(AH,R,Op2)
  shows IsAring(AH//R,A,M)
  using assms QuotientGroupOp˙def Group˙ZF˙3˙3˙T1 Group˙ZF˙3˙4˙T2
    Ring˙ZF˙1˙1˙L4 IsAring˙def by simp
```

end

37 Ordered rings

theory OrderedRing˙ZF imports Ring˙ZF OrderedGroup˙ZF˙1

begin

In this theory file we consider ordered rings.

37.1 Definition and notation

This section defines ordered rings and sets up appriopriate notation.

We define ordered ring as a commutative ring with linear order that is preserved by translations and such that the set of nonnegative elements is closed under multiplication. Note that this definition does not guarantee that there are no zero divisors in the ring.

definition
IsAnOrdRing(R,A,M,r) ≡
(IsAring(R,A,M) ∧ (M –is commutative on″ R) ∧
r⊆R×R ∧ IsLinOrder(R,r) ∧
(∀a b. ∀ c∈R. ⟨ a,b⟩ ∈ r ⟶ ⟨A⟨ a,c⟩,A⟨ b,c⟩⟩ ∈ r) ∧
(Nonnegative(R,A,r) –is closed under″ M))

The next context (locale) defines notation used for ordered rings. We do that by extending the notation defined in the ring0 locale and adding some assumptions to make sure we are talking about ordered rings in this context.

locale ring1 = ring0 +

assumes mult˙commut: M –is commutative on″ R

fixes r

assumes ordincl: r ⊆ R×R

assumes linord: IsLinOrder(R,r)

fixes lesseq (infix ≤ 68)
defines lesseq˙def [simp]: a ≤ b ≡ ⟨ a,b⟩ ∈ r

fixes sless (infix ¡ 68)
defines sless˙def [simp]: a ¡ b ≡ a≤b ∧ a≠b

assumes ordgroup: ∀a b. ∀ c∈R. a≤b ⟶ a+c ≤ b+c

```
  assumes pos˙mult˙closed: Nonnegative(R,A,r) –is closed under″ M

  fixes abs (— ˙ —)
  defines abs˙def [simp]: —a— ≡ AbsoluteValue(R,A,r)(a)

  fixes positiveset (R+)
  defines positiveset˙def [simp]: R+ ≡ PositiveSet(R,A,r)
```

The next lemma assures us that we are talking about ordered rings in the ring1 context.

```
lemma (in ring1) OrdRing˙ZF˙1˙L1: shows IsAnOrdRing(R,A,M,r)
  using ring0˙def ringAssum mult˙commut ordincl linord ordgroup
    pos˙mult˙closed IsAnOrdRing˙def by simp
```

We can use theorems proven in the ring1 context whenever we talk about an ordered ring.

```
lemma OrdRing˙ZF˙1˙L2: assumes IsAnOrdRing(R,A,M,r)
  shows ring1(R,A,M,r)
  using assms IsAnOrdRing˙def ring1˙axioms.intro ring0˙def ring1˙def
  by simp
```

In the ring1 context $a \leq b$ implies that a, b are elements of the ring.

```
lemma (in  ring1) OrdRing˙ZF˙1˙L3: assumes a≤b
  shows a∈R  b∈R
  using assms ordincl by auto
```

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 assms 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 assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L2
  IsOpClosed˙def by simp
```

Double nonnegative is nonnegative.

```
lemma (in  ring1) OrdRing˙ZF˙1˙L5A: assumes A1: 0≤a
```

```
  shows 0≤2·a
  using assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L5G
  OrdRing˙ZF˙1˙L3 Ring˙ZF˙1˙L3 by simp
```

A sufficient (somewhat redundant) condition for a structure to be an ordered ring. It says that a commutative ring that is a totally ordered group with respect to the additive operation such that set of nonnegative elements is closed under multiplication, is an ordered ring.

```
lemma OrdRing˙ZF˙1˙L6:
  assumes
  IsAring(R,A,M)
  M –is commutative on″ R
  Nonnegative(R,A,r) –is closed under″ M
  IsAnOrdGroup(R,A,r)
  r –is total on″ R
  shows IsAnOrdRing(R,A,M,r)
  using assms IsAnOrdGroup˙def Order˙ZF˙1˙L3 IsAnOrdRing˙def
  by simp
```

$a \leq b$ iff $a - b \leq 0$. This is a fact from OrderedGroup.thy, where it is stated in multiplicative notation.

```
lemma (in ring1) OrdRing˙ZF˙1˙L7:
  assumes a∈R  b∈R
  shows a≤b ⟷ a-b ≤ 0
  using assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L9
  by simp
```

Negative times positive is negative.

```
lemma (in ring1) OrdRing˙ZF˙1˙L8:
  assumes A1: a≤0  and A2: 0≤b
  shows a·b ≤ 0
proof -
  from A1 A2 have T1: a∈R  b∈R  a·b ∈ R
    using OrdRing˙ZF˙1˙L3 Ring˙ZF˙1˙L4 by auto
  from A1 A2 have 0≤(-a)·b
    using OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L5A OrdRing˙ZF˙1˙L5
    by simp
  with T1 show a·b ≤ 0
    using Ring˙ZF˙1˙L7 OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L5AA
    by simp
qed
```

We can multiply both sides of an inequality by a nonnegative ring element. This property is sometimes (not here) used to define ordered rings.

```
lemma (in ring1) OrdRing˙ZF˙1˙L9:
  assumes A1: a≤b and A2: 0≤c
  shows
  a·c ≤ b·c
```

```
  c·a ≤ c·b
proof -
  from A1 A2 have T1:
    a∈R  b∈R  c∈R  a·c ∈ R  b·c ∈ R
    using OrdRing˙ZF˙1˙L3 Ring˙ZF˙1˙L4 by auto
  with A1 A2 have (a-b)·c ≤ 0
    using OrdRing˙ZF˙1˙L7 OrdRing˙ZF˙1˙L8 by simp
  with T1 show a·c ≤ b·c
    using Ring˙ZF˙1˙L8 OrdRing˙ZF˙1˙L7 by simp
  with mult˙commut T1 show c·a ≤ c·b
    using IsCommutative˙def by simp
qed
```

A special case of OrdRing˙ZF˙1˙L9: we can multiply an inequality by a positive ring element.

```
lemma (in ring1) OrdRing˙ZF˙1˙L9A:
  assumes A1: a≤b and A2: c∈R+
  shows
  a·c ≤ b·c
  c·a ≤ c·b
proof -
  from A2 have 0 ≤ c using PositiveSet˙def
    by simp
  with A1 show a·c ≤ b·c  c·a ≤ c·b
    using OrdRing˙ZF˙1˙L9 by auto
qed
```

A square is nonnegative.

```
lemma (in ring1) OrdRing˙ZF˙1˙L10:
  assumes A1: a∈R shows 0≤(a^2)
proof -
  { assume 0≤a
    then have 0≤(a^2) using OrdRing˙ZF˙1˙L5 by simp}
  moreover
  { assume ¬(0≤a)
    with A1 have 0≤((-a)^2)
      using OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L8A
 OrdRing˙ZF˙1˙L5 by simp
    with A1 have 0≤(a^2) using Ring˙ZF˙1˙L14 by simp }
  ultimately show thesis by blast
qed
```

1 is nonnegative.

```
corollary (in ring1) ordring˙one˙is˙nonneg: shows 0 ≤ 1
proof -
  have 0 ≤ (1^2) 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 assms 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 assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L5AB
  by simp
```

A negative element cannot be a square.

```
lemma (in ring1) OrdRing˙ZF˙1˙L12:
  assumes A1: a≤0  a≠0
  shows ¬(∃b∈R. a = (b^2))
proof -
  { assume ∃b∈R. a = (b^2)
    with A1 have False using OrdRing˙ZF˙1˙L10 OrdRing˙ZF˙1˙L11
      by auto
  } then show thesis by auto
qed
```

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

```
lemma (in ring1) OrdRing˙ZF˙1˙L13: assumes a≤b
  shows 0 ≤ b-a
  using assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L9D
  by simp
```

If $a < b$, then $0 < b - a$.

```
lemma (in ring1) OrdRing˙ZF˙1˙L14: assumes a≤b  a≠b
  shows
  0 ≤ b-a  0 ≠ b-a
  b-a ∈ R+
  using assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L9E
  by auto
```

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

```
lemma (in ring1) OrdRing˙ZF˙1˙L15:
  assumes a∈R b∈R and 0 ≤ b-a
  shows a≤b
  using assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L9F
  by simp
```

A nonnegative number is does not decrease when multiplied by a number greater or equal 1.

```
lemma (in ring1) OrdRing˙ZF˙1˙L16:
  assumes A1: 0≤a and A2: 1≤b
  shows a≤a·b
proof -
  from A1 A2 have T: a∈R  b∈R  a·b ∈ R
    using OrdRing˙ZF˙1˙L3 Ring˙ZF˙1˙L4 by auto
  from A1 A2 have 0 ≤ a·(b-1)
    using OrdRing˙ZF˙1˙L13 OrdRing˙ZF˙1˙L5 by simp
  with T show a≤a·b
    using Ring˙ZF˙1˙L8 Ring˙ZF˙1˙L2 Ring˙ZF˙1˙L3 OrdRing˙ZF˙1˙L15
    by simp
qed
```

We can multiply the right hand side of an inequality between nonnegative ring elements by an element greater or equal 1.

```
lemma (in ring1) OrdRing˙ZF˙1˙L17:
  assumes A1: 0≤a and A2: a≤b and A3: 1≤c
  shows a≤b·c
proof -
  from A1 A2 have 0≤b by (rule ring˙ord˙transitive)
  with A3 have b≤b·c using OrdRing˙ZF˙1˙L16
    by simp
  with A2 show a≤b·c by (rule ring˙ord˙transitive)
qed
```

Strict order is preserved by translations.

```
lemma (in ring1) ring˙strict˙ord˙trans˙inv:
  assumes a¡b and c∈R
  shows
  a+c ¡ b+c
  c+a ¡ c+b
  using assms OrdRing˙ZF˙1˙L4 group3.group˙strict˙ord˙transl˙inv
  by auto
```

We can put an element on the other side of a strict inequality, changing its sign.

```
lemma (in ring1) OrdRing˙ZF˙1˙L18:
  assumes a∈R  b∈R and  a-b ¡ c
  shows a ¡ c+b
  using assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L12B
  by simp
```

We can add the sides of two inequalities, the first of them strict, and we get a strict inequality. Property of ordered groups.

```
lemma (in ring1) OrdRing˙ZF˙1˙L19:
  assumes a¡b and c≤d
  shows a+c ¡ b+d
  using assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L12C
```

by simp

We can add the sides of two inequalities, the second of them strict and we get a strict inequality. Property of ordered groups.

lemma (in ring1) OrdRing˙ZF˙1˙L20:
assumes a≤b and c¡d
shows a+c ¡ b+d
using assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L12D
by simp

37.2 Absolute value for ordered rings

Absolute value is defined for ordered groups as a function that is the identity on the nonnegative set and the negative of the element (the inverse in the multiplicative notation) on the rest. In this section we consider properties of absolute value related to multiplication in ordered rings.

Absolute value of a product is the product of absolute values: the case when both elements of the ring are nonnegative.

lemma (in ring1) OrdRing˙ZF˙2˙L1:
assumes **0**≤a **0**≤b
shows —a·b— = —a—·—b—
using assms OrdRing˙ZF˙1˙L5 OrdRing˙ZF˙1˙L4
group3.OrderedGroup˙ZF˙1˙L2 group3.OrderedGroup˙ZF˙3˙L2
by simp

The absolue value of an element and its negative are the same.

lemma (in ring1) OrdRing˙ZF˙2˙L2: assumes a∈R
shows —-a— = —a—
using assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙3˙L7A by simp

The next lemma states that $|a \cdot (-b)| = |(-a) \cdot b| = |(-a) \cdot (-b)| = |a \cdot b|$.

lemma (in ring1) OrdRing˙ZF˙2˙L3:
assumes a∈R b∈R
shows
—(-a)·b— = —a·b—
—a·(-b)— = —a·b—
—(-a)·(-b)— = —a·b—
using assms Ring˙ZF˙1˙L4 Ring˙ZF˙1˙L7 Ring˙ZF˙1˙L7A
OrdRing˙ZF˙2˙L2 by auto

This lemma allows to prove theorems for the case of positive and negative elements of the ring separately.

lemma (in ring1) OrdRing˙ZF˙2˙L4: assumes a∈R and ¬(**0**≤a)
shows **0** ≤ (-a) **0**≠a
using assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L8A
by auto

Absolute value of a product is the product of absolute values.

```
lemma (in ring1) OrdRing˙ZF˙2˙L5:
  assumes A1: a∈R b∈R
  shows —a·b— = —a—·—b—
proof -
  { assume A2: 0≤a have —a·b— = —a—·—b—
    proof -
      { assume 0≤b
 with A2 have —a·b— = —a—·—b—
   using OrdRing˙ZF˙2˙L1 by simp }
      moreover
      { assume ¬(0≤b)
 with A1 A2 have —a·(-b)— = —a—·—-b—
   using OrdRing˙ZF˙2˙L4 OrdRing˙ZF˙2˙L1 by simp
 with A1 have —a·b— = —a—·—b—
   using OrdRing˙ZF˙2˙L2 OrdRing˙ZF˙2˙L3 by simp }
      ultimately show thesis by blast
    qed }
  moreover
  { assume ¬(0≤a)
    with A1 have A3: 0 ≤ (-a)
      using OrdRing˙ZF˙2˙L4 by simp
    have —a·b— = —a—·—b—
    proof -
      { assume 0≤b
 with A3 have —(-a)·b— = —-a—·—b—
   using OrdRing˙ZF˙2˙L1 by simp
 with A1 have —a·b— = —a—·—b—
   using OrdRing˙ZF˙2˙L2 OrdRing˙ZF˙2˙L3 by simp }
      moreover
      { assume ¬(0≤b)
 with A1 A3 have —(-a)·(-b)— = —-a—·—-b—
   using OrdRing˙ZF˙2˙L4 OrdRing˙ZF˙2˙L1 by simp
 with A1 have —a·b— = —a—·—b—
   using OrdRing˙ZF˙2˙L2 OrdRing˙ZF˙2˙L3 by simp }
      ultimately show thesis by blast
    qed }
  ultimately show thesis by blast
qed
```

Triangle inequality. Property of linearly ordered abelian groups.

```
lemma (in ring1) ord˙ring˙triangle˙ineq:  assumes a∈R b∈R
  shows —a+b— ≤ —a—+—b—
  using assms OrdRing˙ZF˙1˙L4 group3.OrdGroup˙triangle˙ineq
  by simp
```

If $a \leq c$ and $b \leq c$, then $a + b \leq 2 \cdot c$.

```
lemma (in ring1) OrdRing˙ZF˙2˙L6:
  assumes a≤c  b≤c shows a+b ≤ 2·c
```

```
  using assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L5B
    OrdRing˙ZF˙1˙L3 Ring˙ZF˙1˙L3 by simp
```

37.3 Positivity in ordered rings

This section is about properties of the set of positive elements R_+.

The set of positive elements is closed under ring addition. This is a property of ordered groups, we just reference a theorem from OrderedGroup˙ZF theory in the proof.

```
lemma (in ring1) OrdRing˙ZF˙3˙L1: shows R+ –is closed under″ A
  using OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L13
  by simp
```

Every element of a ring can be either in the postitive set, equal to zero or its opposite (the additive inverse) is in the positive set. This is a property of ordered groups, we just reference a theorem from OrderedGroup˙ZF theory.

```
lemma (in ring1) OrdRing˙ZF˙3˙L2: assumes a∈R
  shows Exactly˙1˙of˙3˙holds (a=0, a∈R+, (-a) ∈ R+)
  using assms OrdRing˙ZF˙1˙L4 group3.OrdGroup˙decomp
  by simp
```

If a ring element $a \neq 0$, and it is not positive, then $-a$ is positive.

```
lemma (in ring1) OrdRing˙ZF˙3˙L2A: assumes a∈R  a≠0  a ∉ R+
  shows (-a) ∈  R+
  using assms OrdRing˙ZF˙1˙L4 group3.OrdGroup˙cases
  by simp
```

R_+ is closed under multiplication iff the ring has no zero divisors.

```
lemma (in ring1) OrdRing˙ZF˙3˙L3:
  shows (R+ –is closed under″ M)⟷ HasNoZeroDivs(R,A,M)
proof
  assume A1: HasNoZeroDivs(R,A,M)
  { fix a b assume a∈R+  b∈R+
    then have 0≤a  a≠0  0≤b  b≠0
      using PositiveSet˙def by auto
    with A1 have a·b ∈ R+
      using OrdRing˙ZF˙1˙L5 Ring˙ZF˙1˙L2 OrdRing˙ZF˙1˙L3 Ring˙ZF˙1˙L12
 OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L2A
      by simp
  } then show  R+ –is closed under″ M using IsOpClosed˙def
    by simp
next assume A2: R+ –is closed under″ M
  { fix a b assume A3: a∈R  b∈R  and a≠0  b≠0
    with A2 have —a·b— ∈ R+
      using OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙3˙L12 IsOpClosed˙def
        OrdRing˙ZF˙2˙L5 by simp
    with A3 have a·b ≠ 0
```

```
    using PositiveSet˙def Ring˙ZF˙1˙L4
 OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙3˙L2A
    by auto
  } then show HasNoZeroDivs(R,A,M) using HasNoZeroDivs˙def
    by auto
qed
```

Another (in addition to OrdRing˙ZF˙1˙L6 sufficient condition that defines order in an ordered ring starting from the positive set.

```
theorem (in ring0) ring˙ord˙by˙positive˙set:
  assumes
  A1: M –is commutative on″ R and
  A2: P⊆R  P –is closed under″ A  0 ∉ P and
  A3: ∀a∈R. a≠0 ⟶ (a∈P) Xor ((-a) ∈ P) and
  A4: P –is closed under″ M and
  A5: r = OrderFromPosSet(R,A,P)
  shows
  IsAnOrdGroup(R,A,r)
  IsAnOrdRing(R,A,M,r)
  r –is total on″ R
  PositiveSet(R,A,r) = P
  Nonnegative(R,A,r) = P ∪ –0″
  HasNoZeroDivs(R,A,M)
proof -
  from A2 A3 A5 show
    I: IsAnOrdGroup(R,A,r)  r –is total on″ R and
    II: PositiveSet(R,A,r) = P and
    III: Nonnegative(R,A,r) = P ∪ –0″
    using Ring˙ZF˙1˙L1 group0.Group˙ord˙by˙positive˙set
    by auto
  from A2 A4 III have Nonnegative(R,A,r) –is closed under″ M
    using Ring˙ZF˙1˙L16 by simp
  with ringAssum A1 I show IsAnOrdRing(R,A,M,r)
    using OrdRing˙ZF˙1˙L6 by simp
  with A4 II show HasNoZeroDivs(R,A,M)
    using OrdRing˙ZF˙1˙L2 ring1.OrdRing˙ZF˙3˙L3
    by auto
qed
```

Nontrivial ordered rings are infinite. More precisely we assume that the neutral element of the additive operation is not equal to the multiplicative neutral element and show that the the set of positive elements of the ring is not a finite subset of the ring and the ring is not a finite subset of itself.

```
theorem (in ring1) ord˙ring˙infinite: assumes 0≠1
  shows
  R+ ∉ Fin(R)
  R ∉ Fin(R)
  using assms Ring˙ZF˙1˙L17 OrdRing˙ZF˙1˙L4 group3.Linord˙group˙infinite
  by auto
```

If every element of a nontrivial ordered ring can be dominated by an element from B, then we B is not bounded and not finite.

```
lemma (in ring1) OrdRing˙ZF˙3˙L4:
  assumes 0≠1 and ∀a∈R. ∃b∈B. a≤b
  shows
  ¬IsBoundedAbove(B,r)
  B ∉ Fin(R)
  using assms Ring˙ZF˙1˙L17 OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙2˙L2A
  by auto
```

If m is greater or equal the multiplicative unit, then the set $\{m \cdot n : n \in R\}$ is infinite (unless the ring is trivial).

```
lemma (in ring1) OrdRing˙ZF˙3˙L5: assumes A1: 0≠1 and A2: 1≤m
  shows
  –m·x. x∈R₊″ ∉ Fin(R)
  –m·x. x∈R″ ∉ Fin(R)
  –(-m)·x. x∈R″ ∉ Fin(R)
proof -
  from A2 have T: m∈R using OrdRing˙ZF˙1˙L3 by simp
  from A2 have 0≤1  1≤m
    using ordring˙one˙is˙nonneg by auto
  then have I: 0≤m by (rule ring˙ord˙transitive)
  let B = –m·x. x∈R₊″
  { fix a assume A3: a∈R
    then have a≤0 ∨ (0≤a ∧ a≠0)
      using ord˙ring˙split2 by simp
    moreover
    { assume A4: a≤0
      from A1 have m·1 ∈ B using ordring˙one˙is˙pos
 by auto
      with T have m∈B using Ring˙ZF˙1˙L3 by simp
      moreover from A4 I have a≤m by (rule ring˙ord˙transitive)
      ultimately have ∃b∈B. a≤b by blast }
    moreover
    { assume A4: 0≤a ∧ a≠0
      with A3 have m·a ∈ B using PositiveSet˙def
 by auto
      moreover
      from A2 A4 have 1·a ≤ m·a using OrdRing˙ZF˙1˙L9
 by simp
      with A3 have a ≤ m·a using Ring˙ZF˙1˙L3
 by simp
      ultimately have ∃b∈B. a≤b by auto }
    ultimately have ∃b∈B. a≤b by auto
  } then have ∀a∈R. ∃b∈B. a≤b
    by simp
  with A1 show B ∉ Fin(R) using OrdRing˙ZF˙3˙L4
    by simp
  moreover have B ⊆ –m·x. x∈R″
```

```
    using PositiveSet˙def by auto
  ultimately show –m·x. x∈R″ ∉ Fin(R) using Fin˙subset
    by auto
  with T show –(-m)·x. x∈R″ ∉ Fin(R) using Ring˙ZF˙1˙L18
    by simp
qed
```

If m is less or equal than the negative of multiplicative unit, then the set $\{m \cdot n : n \in R\}$ is infinite (unless the ring is trivial).

```
lemma (in ring1) OrdRing˙ZF˙3˙L6: assumes A1: 0≠1 and A2: m ≤ -1
  shows –m·x. x∈R″ ∉ Fin(R)
proof -
  from A2 have (-(-1)) ≤ -m
    using OrdRing˙ZF˙1˙L4B by simp
  with A1 have –(-m)·x. x∈R″ ∉ Fin(R)
    using Ring˙ZF˙1˙L2 Ring˙ZF˙1˙L3 OrdRing˙ZF˙3˙L5
    by simp
  with A2 show –m·x. x∈R″ ∉ Fin(R)
    using OrdRing˙ZF˙1˙L3 Ring˙ZF˙1˙L18 by simp
qed
```

All elements greater or equal than an element of R_+ belong to R_+. Property of ordered groups.

```
lemma (in ring1) OrdRing˙ZF˙3˙L7: assumes A1: a ∈ R+ and A2: a≤b
  shows b ∈ R+
proof -
  from A1 A2 have
    group3(R,A,r)
    a ∈ PositiveSet(R,A,r)
    ⟨a,b⟩ ∈ r
    using OrdRing˙ZF˙1˙L4 by auto
  then have b ∈ PositiveSet(R,A,r)
    by (rule group3.OrderedGroup˙ZF˙1˙L19)
  then show b ∈ R+ by simp
qed
```

A special case of OrdRing˙ZF˙3˙L7: a ring element greater or equal than 1 is positive.

```
corollary (in ring1) OrdRing˙ZF˙3˙L8: assumes A1: 0≠1 and A2: 1≤a
  shows a ∈ R+
proof -
  from A1 A2 have 1 ∈ R+  1≤a
    using ordring˙one˙is˙pos by auto
  then show a ∈ R+ by (rule OrdRing˙ZF˙3˙L7)
qed
```

Adding a positive element to a strictly increases a. Property of ordered groups.

```
lemma (in ring1) OrdRing˙ZF˙3˙L9: assumes A1: a∈R  b∈R+
  shows a ≤ a+b  a ≠ a+b
  using assms OrdRing˙ZF˙1˙L4 group3.OrderedGroup˙ZF˙1˙L22
  by auto
```

A special case of OrdRing˙ZF˙3˙L9: in nontrivial rings adding one to a increases a.

```
corollary (in ring1) OrdRing˙ZF˙3˙L10: assumes A1: 0≠1 and A2: a∈R
  shows a ≤ a+1  a ≠ a+1
  using assms ordring˙one˙is˙pos OrdRing˙ZF˙3˙L9
  by auto
```

If a is not greater than b, then it is strictly less than $b + 1$.

```
lemma (in ring1) OrdRing˙ZF˙3˙L11: assumes A1: 0≠1 and A2: a≤b
  shows a¡ b+1
proof -
  from A1 A2 have I: b ¡ b+1
    using OrdRing˙ZF˙1˙L3 OrdRing˙ZF˙3˙L10 by auto
  with A2 show a¡ b+1 by (rule ring˙strict˙ord˙transit)
qed
```

For any ring element a the greater of a and 1 is a positive element that is greater or equal than m. If we add 1 to it we get a positive element that is strictly greater than m. This holds in nontrivial rings.

```
lemma (in ring1) OrdRing˙ZF˙3˙L12: assumes A1: 0≠1 and A2: a∈R
  shows
  a ≤ GreaterOf(r,1,a)
  GreaterOf(r,1,a) ∈ R+
  GreaterOf(r,1,a) + 1 ∈ R+
  a ≤ GreaterOf(r,1,a) + 1  a ≠ GreaterOf(r,1,a) + 1
proof -
  from linord have r –is total on″ R using IsLinOrder˙def
    by simp
  moreover from A2 have 1 ∈ R  a∈R
    using Ring˙ZF˙1˙L2 by auto
  ultimately have
    1 ≤ GreaterOf(r,1,a) and
    I: a ≤ GreaterOf(r,1,a)
    using Order˙ZF˙3˙L2 by auto
  with A1 show
    a ≤ GreaterOf(r,1,a) and
    GreaterOf(r,1,a) ∈ R+
    using OrdRing˙ZF˙3˙L8 by auto
  with A1 show GreaterOf(r,1,a) + 1 ∈ R+
    using ordring˙one˙is˙pos OrdRing˙ZF˙3˙L1 IsOpClosed˙def
    by simp
  from A1 I show
    a ≤ GreaterOf(r,1,a) + 1  a ≠ GreaterOf(r,1,a) + 1
```

```
  using OrdRing˙ZF˙3˙L11 by auto
qed
```

We can multiply strict inequality by a positive element.

```
lemma (in ring1) OrdRing˙ZF˙3˙L13:
 assumes A1: HasNoZeroDivs(R,A,M) and
 A2: a¡b and A3: c∈R+
 shows
 a·c ¡ b·c
 c·a ¡ c·b
proof -
 from A2 A3 have T: a∈R  b∈R  c∈R  c≠0
   using OrdRing˙ZF˙1˙L3 PositiveSet˙def by auto
 from A2 A3 have a·c ≤ b·c using OrdRing˙ZF˙1˙L9A
   by simp
 moreover from A1 A2 T have a·c ≠ b·c
   using Ring˙ZF˙1˙L12A by auto
 ultimately show a·c ¡ b·c by simp
 moreover from mult˙commut T have a·c = c·a and b·c = c·b
   using IsCommutative˙def by auto
 ultimately show c·a ¡ c·b by simp
qed
```

A sufficient condition for an element to be in the set of positive ring elements.

```
lemma (in ring1) OrdRing˙ZF˙3˙L14: assumes 0≤a and a≠0
 shows a ∈ R+
 using assms OrdRing˙ZF˙1˙L3 PositiveSet˙def
 by auto
```

If a ring has no zero divisors, the square of a nonzero element is positive.

```
lemma (in ring1) OrdRing˙ZF˙3˙L15:
 assumes HasNoZeroDivs(R,A,M) and a∈R  a≠0
 shows 0 ≤ a^2  a^2 ≠ 0  a^2 ∈ R+
 using assms OrdRing˙ZF˙1˙L10 Ring˙ZF˙1˙L12 OrdRing˙ZF˙3˙L14
 by auto
```

In rings with no zero divisors we can (strictly) increase a positive element by multiplying it by an element that is greater than 1.

```
lemma (in ring1) OrdRing˙ZF˙3˙L16:
 assumes HasNoZeroDivs(R,A,M) and a ∈ R+ and 1≤b  1≠b
 shows a≤a·b  a ≠ a·b
 using assms PositiveSet˙def OrdRing˙ZF˙1˙L16 OrdRing˙ZF˙1˙L3
   Ring˙ZF˙1˙L12C by auto
```

If the right hand side of an inequality is positive we can multiply it by a number that is greater than one.

```
lemma (in ring1) OrdRing˙ZF˙3˙L17:
  assumes A1: HasNoZeroDivs(R,A,M) and A2: b∈R+ and
```

```
  A3: a≤b  and A4: 1¡c
  shows a¡b·c
proof -
  from A1 A2 A4 have b ¡ b·c
    using OrdRing˙ZF˙3˙L16 by auto
  with A3 show a¡b·c by (rule ring˙strict˙ord˙transit)
qed
```

We can multiply a right hand side of an inequality between positive numbers by a number that is greater than one.

```
lemma (in ring1) OrdRing˙ZF˙3˙L18:
  assumes A1: HasNoZeroDivs(R,A,M) and A2: a ∈ R+ and
  A3: a≤b and A4: 1¡c
  shows a¡b·c
proof -
  from A2 A3 have b ∈ R+ using OrdRing˙ZF˙3˙L7
    by blast
  with A1 A3 A4 show a¡b·c
    using OrdRing˙ZF˙3˙L17 by simp
qed
```

In ordered rings with no zero divisors if at least one of a, b is not zero, then $0 < a^2 + b^2$, in particular $a^2 + b^2 \neq 0$.

```
lemma (in ring1) OrdRing˙ZF˙3˙L19:
  assumes A1: HasNoZeroDivs(R,A,M) and A2: a∈R  b∈R and
  A3: a ≠ 0 ∨ b ≠ 0
  shows 0 ¡ a^2 + b^2
proof -
  { assume a ≠ 0
    with A1 A2 have 0 ≤ a^2  a^2 ≠ 0
      using OrdRing˙ZF˙3˙L15 by auto
    then have 0 ¡ a^2 by auto
    moreover from A2 have 0 ≤ b^2
      using OrdRing˙ZF˙1˙L10 by simp
    ultimately have 0 + 0 ¡ a^2 + b^2
      using OrdRing˙ZF˙1˙L19 by simp
    then have 0 ¡ a^2 + b^2
      using Ring˙ZF˙1˙L2 Ring˙ZF˙1˙L3 by simp }
  moreover
  { assume A4: a = 0
    then have a^2 + b^2 = 0 + b^2
      using  Ring˙ZF˙1˙L2 Ring˙ZF˙1˙L6 by simp
    also from A2 have ... = b^2
      using Ring˙ZF˙1˙L4 Ring˙ZF˙1˙L3 by simp
    finally have a^2 + b^2 = b^2 by simp
    moreover
    from A3 A4 have b ≠ 0 by simp
    with A1 A2 have 0 ≤ b^2 and b^2 ≠ 0
      using OrdRing˙ZF˙3˙L15 by auto
```

```
    hence 0 ¡ b^2 by auto
    ultimately have 0 ¡ a^2 + b^2 by simp }
  ultimately show 0 ¡ a^2 + b^2
    by auto
qed
```

end

38 Cardinal numbers

theory Cardinal˙ZF imports ZF.CardinalArith func1

begin

This theory file deals with results on cardinal numbers (cardinals). Cardinals are a genaralization of the natural numbers, used to measure the cardinality (size) of sets. Contributed by Daniel de la Concepcion.

38.1 Some new ideas on cardinals

All the results of this section are done without assuming the Axiom of Choice. With the Axiom of Choice in play, the proofs become easier and some of the assumptions may be dropped.

Since General Topology Theory is closely related to Set Theory, it is very interesting to make use of all the possibilities of Set Theory to try to classify homeomorphic topological spaces. These ideas are generally used to prove that two topological spaces are not homeomorphic.

There exist cardinals which are the successor of another cardinal, but; as happens with ordinals, there are cardinals which are limit cardinal.

```
definition
    LimitC(i) ≡ Card(i) ∧ 0¡i ∧ (∀y. (y¡i∧Card(y)) ⟶ csucc(y)¡i)
```

Simple fact used a couple of times in proofs.

```
lemma nat˙less˙infty: assumes n∈nat and InfCard(X) shows n¡X
proof -
  from assms have n¡nat and nat≤X using lt˙def InfCard˙def by auto
  then show n¡X using lt˙trans2 by blast
qed
```

There are three types of cardinals, the zero one, the succesors of other cardinals and the limit cardinals.

```
lemma Card˙cases˙disj:
```

```
  assumes Card(i)
  shows i=0 — (∃j. Card(j) ∧ i=csucc(j)) — LimitC(i)
proof-
  from assms have D: Ord(i) using Card˙is˙Ord by auto
  {
    assume F: i≠0
    assume Contr: ˜LimitC(i)
    from F D have 0¡i using Ord˙0˙lt by auto
    with Contr assms have ∃y. y ¡ i ∧ Card(y) ∧ ¬ csucc(y) ¡ i
      using LimitC˙def by blast
    then obtain y where  y ¡ i ∧ Card(y) ∧ ¬ csucc(y) ¡ i  by blast
    with D have  y ¡ i  i≤csucc(y) and O: Card(y)
      using not˙lt˙imp˙le lt˙Ord Card˙csucc Card˙is˙Ord
      by auto
    with assms have csucc(y)≤ii≤csucc(y) using csucc˙le by auto
    then have i=csucc(y) using le˙anti˙sym by auto
    with O have ∃j. Card(j) ∧ i=csucc(j) by auto
  } thus thesis by auto
qed
```

Given an ordinal bounded by a cardinal in ordinal order, we can change to the order of sets.

```
lemma le˙imp˙lesspoll:
  assumes Card(Q)
  shows A ≤ Q ⟹ A ≲ Q
proof -
  assume A ≤ Q
  then have A¡Q∨A=Q using le˙iff by auto
  then have A≈Q∨A¡ Q using eqpoll˙refl by auto
  with assms have A≈Q∨A≺ Q using lt˙Card˙imp˙lesspoll by auto
  then show A≲Q using lesspoll˙def eqpoll˙imp˙lepoll by auto
qed
```

There are two types of infinite cardinals, the natural numbers and those that have at least one infinite strictly smaller cardinal.

```
lemma InfCard˙cases˙disj:
  assumes InfCard(Q)
  shows Q=nat ∨ (∃j. csucc(j)≲Q ∧ InfCard(j))
proof-
  {
    assume ∀j. ¬ csucc(j) ≲ Q ∨ ¬ InfCard(j)
    then have D: ¬ csucc(nat) ≲ Q using InfCard˙nat by auto
    with D assms have ¬(csucc(nat) ≤ Q) using le˙imp˙lesspoll InfCard˙is˙Card
      by auto
    with assms have Q¡(csucc(nat))
      using not˙le˙iff˙lt Card˙is˙Ord Card˙csucc Card˙is˙Ord
        Card˙is˙Ord InfCard˙is˙Card Card˙nat by auto
    with assms have Q≤nat using Card˙lt˙csucc˙iff InfCard˙is˙Card Card˙nat
      by auto
```

```
    with assms have Q=nat using InfCard˙def le˙anti˙sym by auto
  }
  thus thesis by auto
qed
```

A more readable version of standard Isabelle/ZF Ord˙linear˙lt

```
lemma Ord˙linear˙lt˙IML: assumes Ord(i) Ord(j)
  shows i¡j ∨ i=j ∨ j¡i
  using assms lt˙def Ord˙linear disjE by simp
```

A set is injective and not bijective to the successor of a cardinal if and only if it is injective and possibly bijective to the cardinal.

```
lemma Card˙less˙csucc˙eq˙le:
  assumes Card(m)
  shows A ≺ csucc(m) ⟷ A ≲ m
proof
  have S: Ord(csucc(m)) using Card˙csucc Card˙is˙Ord assms by auto
  {
    assume A: A ≺ csucc(m)
    with S have —A—≈A using lesspoll˙imp˙eqpoll by auto
    also from A have ...≺ csucc(m) by auto
    finally have —A—≺ csucc(m) by auto
    then have —A—≲csucc(m)˜(—A—≈csucc(m)) using lesspoll˙def by auto
     with S have ——A——≤csucc(m)—A—≠csucc(m) using lepoll˙cardinal˙le by
auto
     then have —A—≤csucc(m) —A—≠csucc(m) using Card˙def Card˙cardinal by
auto
    then have I: ˜(csucc(m)¡—A—) —A—≠csucc(m) using le˙imp˙not˙lt by auto
    from S have csucc(m)¡—A— ∨ —A—=csucc(m) ∨ —A—¡csucc(m)
      using Card˙cardinal Card˙is˙Ord Ord˙linear˙lt˙IML by auto
    with I have —A—¡csucc(m) by simp
    with assms have —A—≤m using Card˙lt˙csucc˙iff Card˙cardinal
      by auto
    then have —A—=m∨ —A—¡ m using le˙iff by auto
    then have —A—≈m∨—A—¡ m using eqpoll˙refl by auto
    then have —A—≈m∨—A—≺ m using lt˙Card˙imp˙lesspoll assms by auto
    then have T:—A—≲m using lesspoll˙def eqpoll˙imp˙lepoll by auto
    from A S have A≈—A— using lesspoll˙imp˙eqpoll eqpoll˙sym by auto
    also from T have ...≲m by auto
    finally show A≲m by simp
  }
  {
    assume A: A≲m
    from assms have m≺csucc(m) using lt˙Card˙imp˙lesspoll Card˙csucc Card˙is˙Ord
      lt˙csucc by auto
    with A show A≺csucc(m) using lesspoll˙trans1 by auto
  }
qed
```

If the successor of a cardinal is infinite, so is the original cardinal.

```
lemma csucc˙inf˙imp˙inf:
  assumes Card(j) and InfCard(csucc(j))
  shows InfCard(j)
proof-
  {
    assume f:Finite (j)
    then obtain n where n∈nat j≈n using Finite˙def by auto
    with assms(1) have TT: j=n n∈nat
      using cardinal˙cong nat˙into˙Card Card˙def by auto
    then have Q:succ(j)∈nat using nat˙succI by auto
    with f TT have T: Finite(succ(j)) Card(succ(j))
      using nat˙into˙Card nat˙succI by auto
    from T(2) have Card(succ(j))∧ j¡succ(j) using Card˙is˙Ord by auto
    moreover from this have Ord(succ(j)) using Card˙is˙Ord by auto
    moreover
    { fix x
      assume A: x¡succ(j)
      {
        assume Card(x)∧ j¡x
        with A have False using lt˙trans1 by auto
      }
      hence ˜(Card(x)∧ j¡x) by auto
    }
    ultimately have (μ L. Card(L) ∧ j ¡ L)=succ(j)
      by (rule Least˙equality)
    then have csucc(j)=succ(j) using csucc˙def by auto
    with Q have csucc(j)∈nat by auto
    then have csucc(j)¡nat using lt˙def Card˙nat Card˙is˙Ord by auto
    with assms(2) have False using InfCard˙def lt˙trans2 by auto
  }
  then have ˜(Finite (j)) by auto
  with assms(1) show thesis using Inf˙Card˙is˙InfCard by auto
qed
```

Since all the cardinals previous to nat are finite, it cannot be a successor cardinal; hence it is a LimitC cardinal.

```
corollary LimitC˙nat:
  shows LimitC(nat)
proof-
  note Card˙nat
  moreover have 0¡nat using lt˙def by auto
  moreover
  {
    fix y
    assume AS: y¡natCard(y)
    then have ord: Ord(y) unfolding lt˙def by auto
    then have Cacsucc: Card(csucc(y)) using Card˙csucc by auto
    {
```

```
    assume nat≤csucc(y)
    with Cacsucc have InfCard(csucc(y)) using InfCard˙def by auto
    with AS(2) have InfCard(y) using csucc˙inf˙imp˙inf by auto
    then have nat≤y using InfCard˙def by auto
    with AS(1) have False using lt˙trans2 by auto
  }
  hence ˜(nat≤csucc(y)) by auto
  then have csucc(y)¡nat using not˙le˙iff˙lt Ord˙nat Cacsucc Card˙is˙Ord by auto
 }
 ultimately show thesis using LimitC˙def by auto
qed
```

38.2 Main result on cardinals (without the Axiom of Choice)

If two sets are strictly injective to an infinite cardinal, then so is its union. For the case of successor cardinal, this theorem is done in the isabelle library in a more general setting; but that theorem is of not use in the case where LimitC(Q) and it also makes use of the Axiom of Choice. The mentioned theorem is in the theory file Cardinal˙AC.thy

Note that if Q is finite and different from 1, let's assume $Q = n$, then the union of A and B is not bounded by Q. Counterexample: two disjoint sets of $n-1$ elements each have a union of $2n-2$ elements which are more than n.

Note also that if $Q = 1$ then A and B must be empty and the union is then empty too; and Q cannot be 0 because no set is injective and not bijective to 0.

The proof is divided in two parts, first the case when both sets A and B are finite; and second, the part when at least one of them is infinite. In the first part, it is used the fact that a finite union of finite sets is finite. In the second part it is used the linear order on cardinals (ordinals). This proof can not be generalized to a setting with an infinite union easily.

```
lemma less˙less˙imp˙un˙less:
 assumes A≺Q and B≺Q and InfCard(Q)
 shows A ∪ B≺Q
proof-
{
 assume Finite (A) ∧ Finite(B)
 then have Finite(A ∪ B) using Finite˙Un by auto
 then obtain n where R: A ∪ B ≈n  n∈nat using Finite˙def
   by auto
 then have —A ∪ B—¡nat using lt˙def  cardinal˙cong
   nat˙into˙Card  Card˙def  Card˙nat Card˙is˙Ord by auto
 with assms(3) have T: —A ∪ B—¡Q using InfCard˙def lt˙trans2 by auto
 from R have Ord(n)A ∪ B ≲ n using nat˙into˙Card Card˙is˙Ord eqpoll˙imp˙lepoll
by auto
 then have A ∪ B≈—A ∪ B— using lepoll˙Ord˙imp˙eqpoll eqpoll˙sym by auto
```

```
  also from T assms(3) have ...≺Q using lt˙Card˙imp˙lesspoll InfCard˙is˙Card
    by auto
  finally have A ∪ B≺Q by simp
}
moreover
{
  assume ˜(Finite (A) ∧ Finite(B))
  hence A: ˜Finite (A) ∨ ˜Finite(B) by auto
  from assms have B: —A—≈A —B—≈B using lesspoll˙imp˙eqpoll lesspoll˙imp˙eqpoll
    InfCard˙is˙Card Card˙is˙Ord by auto
  from B(1) have Aeq: ∀x. (—A—≈x) ⟶ (A≈x)
    using eqpoll˙sym eqpoll˙trans by blast
  from B(2) have Beq: ∀x. (—B—≈x) ⟶ (B≈x)
    using eqpoll˙sym eqpoll˙trans by blast
  with A Aeq have ˜Finite(—A—)∨ ˜Finite(—B—) using Finite˙def
    by auto
  then have D: InfCard(—A—)∨InfCard(—B—)
    using Inf˙Card˙is˙InfCard Inf˙Card˙is˙InfCard  Card˙cardinal by blast
  {
    assume AS: —A— ¡ —B—
    {
      assume ˜InfCard(—A—)
      with D have InfCard(—B—) by auto
    }
    moreover
    {
      assume InfCard(—A—)
      then have nat≤—A— using InfCard˙def by auto
      with AS have nat¡—B— using lt˙trans1 by auto
      then have nat≤—B— using leI by auto
      then have InfCard(—B—) using InfCard˙def Card˙cardinal by auto
    }
    ultimately have INFB: InfCard(—B—) by auto
    then have 2¡—B— using nat˙less˙infty by simp
    then have AG: 2≲—B— using lt˙Card˙imp˙lesspoll Card˙cardinal lesspoll˙def
      by auto
    from B(2) have —B—≈B by simp
    also from assms(2) have ...≺Q by auto
    finally have TTT: —B—≺Q by simp
   from B(1) have Card(—B—) A ≲—A— using eqpoll˙sym Card˙cardinal eqpoll˙imp˙lepoll

      by auto
    with AS have A≺—B— using  lt˙Card˙imp˙lesspoll lesspoll˙trans1 by auto
    then have I1: A≲—B— using lesspoll˙def by auto
    from B(2) have I2: B≲—B— using  eqpoll˙sym eqpoll˙imp˙lepoll by auto
    have A ∪ B≲A+B using Un˙lepoll˙sum by auto
    also from I1 I2 have ...≲ —B— + —B— using sum˙lepoll˙mono by auto
    also from AG have ...≲—B— * —B— using sum˙lepoll˙prod by auto
    also from assms(3) INFB have ...≈—B— using InfCard˙square˙eqpoll
```

```
    by auto
  finally have A ∪ B≲—B— by simp
  also from TTT have ...≺Q by auto
  finally have A ∪ B≺Q by simp
}
moreover
{
  assume AS: —B— ¡ —A—
  {
    assume ˜InfCard(—B—)
    with D have InfCard(—A—) by auto
  }
  moreover
  {
    assume InfCard(—B—)
    then have nat≤—B— using InfCard˙def by auto
    with AS have nat¡—A— using lt˙trans1 by auto
    then have nat≤—A— using leI by auto
    then have InfCard(—A—) using InfCard˙def Card˙cardinal by auto
  }
  ultimately have INFB: InfCard(—A—) by auto
  then have 2¡—A— using nat˙less˙infty by simp
  then have AG: 2≲—A— using lt˙Card˙imp˙lesspoll Card˙cardinal lesspoll˙def
      by auto
  from B(1) have —A—≈A by simp
  also from assms(1) have ...≺Q by auto
  finally have TTT: —A—≺Q by simp
 from B(2) have Card(—A—) B ≲—B— using eqpoll˙sym Card˙cardinal eqpoll˙imp˙lepoll

    by auto
  with AS have B≺—A— using  lt˙Card˙imp˙lesspoll lesspoll˙trans1 by auto
  then have I1: B≲—A— using lesspoll˙def by auto
  from B(1) have I2: A≲—A— using eqpoll˙sym eqpoll˙imp˙lepoll by auto
  have A ∪ B≲A+B using Un˙lepoll˙sum by auto
  also from I1 I2 have ...≲ —A— + —A— using sum˙lepoll˙mono by auto
  also from AG have ...≲—A— * —A— using sum˙lepoll˙prod by auto
  also from INFB assms(3) have ...≈—A— using InfCard˙square˙eqpoll
     by auto
  finally have A ∪ B≲—A— by simp
  also from TTT have ...≺Q by auto
  finally have A ∪ B≺Q by simp
  }
  moreover
  {
    assume AS: —A—=—B—
    with D have INFB: InfCard(—A—) by auto
    then have 2¡—A— using nat˙less˙infty by simp
  then have AG: 2≲—A— using lt˙Card˙imp˙lesspoll Card˙cardinal using lesspoll˙def
      by auto
```

```
    from B(1) have —A—≈A by simp
    also from assms(1) have ...≺Q by auto
    finally have TTT: —A—≺Q by simp
   from AS B have I1: A≲—A—and I2:B≲—A— using eqpoll˙refl eqpoll˙imp˙lepoll
      eqpoll˙sym by auto
    have A ∪ B≲A+B using Un˙lepoll˙sum by auto
    also from I1 I2 have ...≲ —A— + —A— using sum˙lepoll˙mono by auto
    also from AG have ...≲—A— * —A— using sum˙lepoll˙prod  by auto
    also from assms(3) INFB have ...≈—A— using InfCard˙square˙eqpoll
      by auto
    finally have A ∪ B≲—A— by simp
    also from TTT have ...≺Q by auto
    finally have A ∪ B≺Q by simp
   }
   ultimately have A ∪ B≺Q using Ord˙linear˙lt˙IML Card˙cardinal Card˙is˙Ord by
auto
  }
  ultimately show A ∪ B≺Q by auto
qed
```

38.3 Choice axioms

We want to prove some theorems assuming that some version of the Axiom of Choice holds. To avoid introducing it as an axiom we will defin an appropriate predicate and put that in the assumptions of the theorems. That way technically we stay inside ZF.

The first predicate we define states that the axiom of Q-choice holds for subsets of K if we can find a choice function for every family of subsets of K whose (that family's) cardinality does not exceed Q.

```
definition
  AxiomCardinalChoice (–the axiom of"˙–choice holds for subsets"˙) where
  –the axiom of" Q –choice holds for subsets"K ≡ Card(Q) ∧ (∀ M N. (M ≲Q ∧
(∀t∈M. Nt≠0 ∧ Nt⊆K)) ⟶ (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)))
```

Next we define a general form of Q choice where we don't require a collection of files to be included in a file.

```
definition
  AxiomCardinalChoiceGen (–the axiom of"˙–choice holds") where
 –the axiom of" Q –choice holds" ≡ Card(Q) ∧ (∀ M N. (M ≲Q ∧  (∀t∈M. Nt≠0))
⟶ (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)))
```

The axiom of finite choice always holds.

```
theorem finite˙choice:
  assumes n∈nat
  shows –the axiom of" n –choice holds"
proof -
  note assms(1)
```

```
  moreover
  {
    fix M N assume M≲0 ∀t∈M. Nt≠0
    then have M=0 using lepoll˙0˙is˙0 by auto
     then have –⟨t,0⟩. t∈M″:Pi(M,λt. Nt) unfolding Pi˙def domain˙def function˙def
Sigma˙def by auto
    moreover from ‹M=0› have ∀t∈M. –⟨t,0⟩. t∈M″t∈Nt by auto
    ultimately have (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)) by auto
  }
  then have (∀ M N. (M ≲0 ∧  (∀t∈M. Nt≠0)) ⟶ (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M.
ft∈Nt)))
    by auto
  then have –the axiom of″ 0 –choice holds″ using AxiomCardinalChoiceGen˙def
nat˙into˙Card
    by auto
  moreover {
    fix x
    assume as: x∈nat –the axiom of″ x –choice holds″
    {
      fix M N assume ass: M≲succ(x) ∀t∈M. Nt≠0
      {
        assume M≲x
        from as(2) ass(2) have
          (M ≲ x ∧ (∀t∈M. N  t ≠ 0)) ⟶ (∃f. f ∈ Pi(M,λt. N  t) ∧ (∀t∈M. f  t ∈
N  t))
            unfolding AxiomCardinalChoiceGen˙def by auto
        with ‹M≲x› ass(2) have (∃f. f ∈ Pi(M,λt. N  t) ∧ (∀t∈M. f  t ∈ N  t))
          by auto
      }
      moreover
      {
        assume M≈succ(x)
        then obtain f where f:f∈bij(succ(x),M) using eqpoll˙sym eqpoll˙def by blast
        moreover
        have x∈succ(x) unfolding succ˙def by auto
     ultimately have restrict(f,succ(x)-–x″)∈bij(succ(x)-–x″,M-–fx″) using bij˙restrict˙rem

          by auto
        moreover
        have x∉x using mem˙not˙refl by auto
        then have succ(x)-–x″=x unfolding succ˙def by auto
        ultimately have restrict(f,x)∈bij(x,M-–fx″) by auto
        then have x≈M-–fx″ unfolding eqpoll˙def by auto
        then have M-–fx″≈x using eqpoll˙sym by auto
        then have M-–fx″≲x using eqpoll˙imp˙lepoll by auto
       with as(2) ass(2) have (∃g. g ∈ Pi(M-–fx″,λt. N  t) ∧ (∀t∈M-–fx″. g  t ∈ N
t))
          unfolding AxiomCardinalChoiceGen˙def by auto
        then obtain g where g: g∈ Pi(M-–fx″,λt. N  t) ∀t∈M-–fx″. g  t ∈ N  t
```

```
      by auto
    from f have ff: fx∈M using bij˙def inj˙def apply˙funtype by auto
    with ass(2) have N(fx)≠0 by auto
    then obtain y where y: y∈N(fx) by auto
    from g(1) have gg: g⊆Sigma(M-–fx″,()(N)) unfolding Pi˙def by auto
    with y ff have g ∪–⟨fx, y⟩″⊆Sigma(M, ()(N)) unfolding Sigma˙def by auto
    moreover
    from g(1) have dom: M-–fx″⊆domain(g) unfolding Pi˙def by auto
    then have M⊆domain(g ∪–⟨fx, y⟩″) unfolding domain˙def by auto
    moreover
    from gg g(1) have noe: ˜(∃t. ⟨fx,t⟩∈g) and function(g)
      unfolding domain˙def Pi˙def Sigma˙def by auto
    with dom have fg: function(g ∪–⟨fx, y⟩″) unfolding function˙def by blast
    ultimately have PP: g ∪–⟨fx, y⟩″∈Pi(M,λt. N  t) unfolding Pi˙def by auto
    have ⟨fx, y⟩ ∈ g ∪–⟨fx, y⟩″ by auto
    from this fg have (g ∪–⟨fx, y⟩″)(fx)=y by (rule function˙apply˙equality)
    with y have (g ∪–⟨fx, y⟩″)(fx)∈N(fx) by auto
    moreover
    {
      fix t assume A:t∈M-–fx″
      with g(1) have ⟨t,gt⟩∈g using apply˙Pair by auto
      then have ⟨t,gt⟩∈(g ∪–⟨fx, y⟩″) by auto
      then have (g ∪–⟨fx, y⟩″)t=gt using apply˙equality PP by auto
      with A have (g ∪–⟨fx, y⟩″)t∈Nt using g(2) by auto
    }
    ultimately have ∀t∈M. (g ∪–⟨fx, y⟩″)t∈Nt by auto
    with PP have ∃g. g∈Pi(M,λt. N  t) ∧ (∀t∈M. gt∈Nt) by auto
  }
 ultimately have ∃g. g ∈ Pi(M, λt. Nt) ∧ (∀t∈M. g  t ∈ N  t) using as(1) ass(1)
  lepoll˙succ˙disj by auto
 }
 then have ∀M N. M ≲ succ(x)∧(∀t∈M. Nt≠0)⟶(∃g. g ∈ Pi(M,λt. N  t) ∧
(∀t∈M. g  t ∈ N  t))
  by auto
 then have –the axiom of″succ(x)–choice holds″
  using AxiomCardinalChoiceGen˙def nat˙into˙Card as(1) nat˙succI by auto
}
ultimately show thesis by (rule nat˙induct)
qed
```

The axiom of choice holds if and only if the AxiomCardinalChoice holds for every couple of a cardinal Q and a set K.

```
lemma choice˙subset˙imp˙choice:
 shows –the axiom of″ Q –choice holds″ ⟷ (∀ K. –the axiom of″ Q –choice holds
for subsets″K)
 unfolding AxiomCardinalChoice˙def AxiomCardinalChoiceGen˙def by blast
```

A choice axiom for greater cardinality implies one for smaller cardinality

```
lemma greater˙choice˙imp˙smaller˙choice:
```

```
  assumes Q≲Q1 Card(Q)
  shows –the axiom of″ Q1 –choice holds″ ⟶ (–the axiom of″ Q –choice holds″)
using assms
  AxiomCardinalChoiceGen˙def lepoll˙trans by auto
```

If we have a surjective function from a set which is injective to a set of ordinals, then we can find an injection which goes the other way.

```
lemma surj˙fun˙inv:
  assumes f ∈ surj(A,B) A⊆Q Ord(Q)
  shows B≲A
proof-
  let g = –⟨m,μ j. j∈A ∧ f(j)=m⟩. m∈B″
  have g:B→range(g) using lam˙is˙fun˙range by simp
  then have fun: g:B→g(B) using range˙image˙domain by simp
  from assms(2,3) have OA: ∀j∈A. Ord(j) using lt˙def Ord˙in˙Ord by auto
  {
    fix x
    assume x∈g(B)
    then have x∈range(g) and ∃y∈B. ⟨y,x⟩∈g by auto
    then obtain y where T: x=(μ j. j∈A∧ f(j)=y) and y∈B by auto
    with assms(1) OA obtain z where P: z∈A ∧ f(z)=y Ord(z) unfolding surj˙def
      by auto
    with T have x∈A ∧ f(x)=y using LeastI by simp
    hence x∈A by simp
  }
  then have g(B) ⊆ A by auto
  with fun have fun2: g:B→A using fun˙weaken˙type by auto
  then have g∈inj(B,A)
  proof -
    {
      fix w x
      assume AS: gw=gx w∈B x∈B
      from assms(1) OA AS(2,3) obtain wz xz where
        P1: wz∈A ∧ f(wz)=w  Ord(wz) and P2: xz∈A ∧ f(xz)=x  Ord(xz)
        unfolding surj˙def by blast
      from P1 have (μ j. j∈A∧ fj=w) ∈ A ∧ f(μ j. j∈A∧ fj=w)=w
        by (rule LeastI)
      moreover from P2 have (μ j. j∈A∧ fj=x) ∈ A ∧ f(μ j. j∈A∧ fj=x)=x
        by (rule LeastI)
      ultimately have R: f(μ j. j∈A∧ fj=w)=w f(μ j. j∈A∧ fj=x)=x
        by auto
      from AS have (μ j. j∈A∧ f(j)=w)=(μ j. j∈A ∧ f(j)=x)
        using apply˙equality fun2 by auto
      hence f(μ j. j∈A ∧ f(j)=w) = f(μ j. j∈A ∧ f(j)=x) by auto
      with R(1) have w = f(μ j. j∈A∧ fj=x) by auto
      with R(2) have w=x by auto
    }
    hence ∀w∈B. ∀x∈B. g(w) = g(x) ⟶ w = x
      by auto
```

```
    with fun2 show g∈inj(B,A) unfolding inj˙def by auto
  qed
  then show thesis unfolding lepoll˙def by auto
qed
```

The difference with the previous result is that in this one A is not a subset of an ordinal, it is only injective with one.

```
theorem surj˙fun˙inv˙2:
  assumes f:surj(A,B) A≲Q Ord(Q)
  shows B≲A
proof-
  from assms(2) obtain h where h˙def: h∈inj(A,Q) using lepoll˙def by auto
  then have bij: h∈bij(A,range(h)) using inj˙bij˙range by auto
  then obtain h1 where h1∈bij(range(h),A) using bij˙converse˙bij by auto
  then have h1 ∈ surj(range(h),A) using bij˙def by auto
  with assms(1) have (f O h1)∈surj(range(h),B) using comp˙surj by auto
  moreover
  {
    fix x
    assume p: x∈range(h)
    from bij have h∈surj(A,range(h)) using bij˙def by auto
    with p obtain q where q∈A and h(q)=x using surj˙def by auto
    then have x∈Q using h˙def inj˙def by auto
  }
  then have range(h)⊆Q by auto
  ultimately have B≲range(h) using surj˙fun˙inv assms(3) by auto
  moreover have range(h)≈A using bij eqpoll˙def eqpoll˙sym by blast
  ultimately show B≲A using lepoll˙eq˙trans by auto
qed


end
```

39 Groups 4

```
theory Group˙ZF˙4 imports Group˙ZF˙1 Group˙ZF˙2 Finite˙ZF Ring˙ZF
  Cardinal˙ZF Semigroup˙ZF

begin
```

This theory file deals with normal subgroup test and some finite group theory. Then we define group homomorphisms and prove that the set of endomorphisms forms a ring with unity and we also prove the first isomorphism theorem.

39.1 Conjugation of subgroups

The conjugate of a subgroup is a subgroup.

```
theorem(in group0) semigr0:
  shows semigr0(G,P)
  unfolding semigr0˙def using groupAssum IsAgroup˙def IsAmonoid˙def by auto

theorem (in group0) conj˙group˙is˙group:
  assumes IsAsubgroup(H,P) g∈G
  shows IsAsubgroup(–g·(h·g⁻¹). h∈H″,P)
proof-
  have sub:H⊆G using assms(1) group0˙3˙L2 by auto
  from assms(2) have g⁻¹∈G using inverse˙in˙group by auto
  {
    fix r assume r∈–g·(h·g⁻¹). h∈H″
    then  obtain h where h:h∈H r=g·(h·(g⁻¹)) by auto
    from h(1) have h⁻¹∈H using group0˙3˙T3A assms(1) by auto
    from h(1) sub have h∈G by auto
    then have h⁻¹∈G using inverse˙in˙group by auto
    with ‹g⁻¹∈G› have ((h⁻¹)·(g)⁻¹)∈G using group˙op˙closed by auto
    from h(2) have r⁻¹=(g·(h·(g⁻¹)))⁻¹ by auto moreover
    from ‹h∈G› ‹g⁻¹∈G› have s:h·(g⁻¹)∈G using group˙op˙closed by blast
    ultimately have r⁻¹=(h·(g⁻¹))⁻¹·(g)⁻¹ using group˙inv˙of˙two[OF assms(2)] by
auto
    moreover
    from s assms(2) h(2) have r:r∈G using group˙op˙closed by auto
      have (h·(g⁻¹))⁻¹=(g⁻¹)⁻¹·h⁻¹ using group˙inv˙of˙two[OF ‹h∈G›‹g⁻¹∈G›] by
auto
    moreover have (g⁻¹)⁻¹=g using group˙inv˙of˙inv[OF assms(2)] by auto
    ultimately have r⁻¹=(g·(h⁻¹))·(g)⁻¹ by auto
   then have r⁻¹=g·((h⁻¹)·(g)⁻¹) using group˙oper˙assoc[OF assms(2) ‹h⁻¹∈G›‹g⁻¹∈G›]
by auto
    with ‹h⁻¹∈H› r have r⁻¹∈–g·(h·g⁻¹). h∈H″ r∈G by auto
  }
  then have ∀r∈–g·(h·g⁻¹). h∈H″. r⁻¹∈–g·(h·g⁻¹). h∈H″ and –g·(h·g⁻¹). h∈H″⊆G
by auto moreover
  {
    fix s t assume s:s∈–g·(h·g⁻¹). h∈H″ and t:t∈–g·(h·g⁻¹). h∈H″
    then obtain hs ht where hs:hs∈H s=g·(hs·(g⁻¹)) and ht:ht∈H t=g·(ht·(g⁻¹)) by
auto
    from hs(1) have hs∈G using sub by auto
    then have g·hs∈G using group˙op˙closed assms(2) by auto
    then have (g·hs)⁻¹∈G using inverse˙in˙group by auto
    from ht(1) have ht∈G using sub by auto
    with ‹g⁻¹:G› have ht·(g⁻¹)∈G using group˙op˙closed by auto
    from hs(2) ht(2) have s·t=(g·(hs·(g⁻¹)))·(g·(ht·(g⁻¹))) by auto moreover
   have g·(hs·(g⁻¹))=g·hs·(g⁻¹) using group˙oper˙assoc[OF assms(2) ‹hs∈G› ‹g⁻¹∈G›]
by auto
    then have (g·(hs·(g⁻¹)))·(g·(ht·(g⁻¹)))=(g·hs·(g⁻¹))·(g·(ht·(g⁻¹))) by auto
      then have (g·(hs·(g⁻¹)))·(g·(ht·(g⁻¹)))=(g·hs·(g⁻¹))·(g⁻¹⁻¹·(ht·(g⁻¹))) using
group˙inv˙of˙inv[OF assms(2)] by auto
   also have ...=g·hs·(ht·(g⁻¹)) using group0˙2˙L14A(2)[OF ‹(g·hs)⁻¹∈G› ‹g⁻¹∈G›‹ht·(g⁻¹)∈G›]
```

group˙inv˙of˙inv[OF ‹(g·hs)∈G›]
by auto
ultimately have s·t=g·hs·(ht·(g^{-1})) by auto moreover
have hs·(ht·(g^{-1}))=(hs·ht)·(g^{-1}) using group˙oper˙assoc[OF ‹hs∈G›‹ht∈G›‹g^{-1}∈G›]
by auto moreover
have g·hs·(ht·(g^{-1}))=g·(hs·(ht·(g^{-1}))) using group˙oper˙assoc[OF ‹g∈G›‹hs∈G›‹(ht·g^{-1})∈G›]
by auto
ultimately have s·t=g·((hs·ht)·(g^{-1})) by auto moreover
from hs(1) ht(1) have hs·ht∈H using assms(1) group0˙3˙L6 by auto
ultimately have s·t∈–g·(h·g^{-1}). h∈H″ by auto
}
then have –g·(h·g^{-1}). h∈H″ –is closed under″P unfolding IsOpClosed˙def by auto
moreover
from assms(1) have **1**∈H using group0˙3˙L5 by auto
then have g·(**1**·g^{-1})∈–g·(h·g^{-1}). h∈H″ by auto
then have –g·(h·g^{-1}). h∈H″≠0 by auto ultimately
show thesis using group0˙3˙T3 by auto
qed

Every set is equipollent with its conjugates.

theorem (in group0) conj˙set˙is˙eqpoll:
assumes H⊆G g∈G
shows H≈–g·(h·g^{-1}). h∈H″
proof-
have fun:–⟨h,g·(h·g^{-1})⟩. h∈H″:H→–g·(h·g^{-1}). h∈H″ unfolding Pi˙def function˙def
domain˙def by auto
{
fix h1 h2 assume h1∈Hh2∈H–⟨h,g·(h·g^{-1})⟩. h∈H″h1=–⟨h,g·(h·g^{-1})⟩. h∈H″h2
with fun have g·(h1·g^{-1})=g·(h2·g^{-1})h1·g^{-1}∈Gh2·g^{-1}∈Gh1∈Gh2∈G using ap-
ply˙equality assms(1)
group˙op˙closed[OF ˙ inverse˙in˙group[OF assms(2)]] by auto
then have h1·g^{-1}=h2·g^{-1} using group0˙2˙L19(2)[OF ‹h1·g^{-1}∈G› ‹h2·g^{-1}∈G›
assms(2)] by auto
then have h1=h2 using group0˙2˙L19(1)[OF ‹h1∈G›‹h2∈G› inverse˙in˙group[OF
assms(2)]] by auto
}
then have ∀h1∈H. ∀h2∈H. –⟨h,g·(h·g^{-1})⟩. h∈H″h1=–⟨h,g·(h·g^{-1})⟩. h∈H″h2 ⟶
h1=h2 by auto
with fun have –⟨h,g·(h·g^{-1})⟩. h∈H″∈inj(H,–g·(h·g^{-1}). h∈H″) unfolding inj˙def by
auto moreover
{
fix ghg assume ghg∈–g·(h·g^{-1}). h∈H″
then obtain h where h∈H ghg=g·(h·g^{-1}) by auto
then have ⟨h,ghg⟩∈–⟨h,g·(h·g^{-1})⟩. h∈H″ by auto
then have –⟨h,g·(h·g^{-1})⟩. h∈H″h=ghg using apply˙equality fun by auto
with ‹h∈H› have ∃h∈H. –⟨h,g·(h·g^{-1})⟩. h∈H″h=ghg by auto
}
with fun have –⟨h,g·(h·g^{-1})⟩. h∈H″∈surj(H,–g·(h·g^{-1}). h∈H″) unfolding surj˙def
by auto

ultimately have $-\langle h,g\cdot(h\cdot g^{-1})\rangle$. $h\in H''\in bij(H,-g\cdot(h\cdot g^{-1}).\ h\in H'')$ unfolding bij˙def by auto
then show thesis unfolding eqpoll˙def by auto
qed

Every normal subgroup contains its conjugate subgroups.

theorem (in group0) norm˙group˙cont˙conj:
assumes IsAnormalSubgroup(G,P,H) $g\in G$
shows $-g\cdot(h\cdot g^{-1}).\ h\in H''\subseteq H$
proof-
{
fix r assume $r\in -g\cdot(h\cdot g^{-1}).\ h\in H''$
then obtain h where $r=g\cdot(h\cdot g^{-1})$ $h\in H$ by auto moreover
then have $h\in G$ using group0˙3˙L2 assms(1) unfolding IsAnormalSubgroup˙def by auto moreover
from assms(2) have $g^{-1}\in G$ using inverse˙in˙group by auto
ultimately have $r=g\cdot h\cdot g^{-1}$ $h\in H$ using group˙oper˙assoc assms(2) by auto
then have $r\in H$ using assms unfolding IsAnormalSubgroup˙def by auto
}
then show $-g\cdot(h\cdot g^{-1}).\ h\in H''\subseteq H$ by auto
qed

If a subgroup contains all its conjugate subgroups, then it is normal.

theorem (in group0) cont˙conj˙is˙normal:
assumes IsAsubgroup(H,P) $\forall g\in G.\ -g\cdot(h\cdot g^{-1}).\ h\in H''\subseteq H$
shows IsAnormalSubgroup(G,P,H)
proof-
{
fix h g assume $h\in H$ $g\in G$
with assms(2) have $g\cdot(h\cdot g^{-1})\in H$ by auto
moreover have $h\in Gg^{-1}\in G$ using group0˙3˙L2 assms(1) ‹g∈G› ‹h∈H› inverse˙in˙group by auto
ultimately have $g\cdot h\cdot g^{-1}\in H$ using group˙oper˙assoc ‹g∈G› by auto
}
then show thesis using assms(1) unfolding IsAnormalSubgroup˙def by auto
qed

If a group has only one subgroup of a given order, then this subgroup is normal.

corollary(in group0) only˙one˙equipoll˙sub:
assumes IsAsubgroup(H,P) $\forall M.$ IsAsubgroup(M,P)$\wedge$ $H\approx M \longrightarrow M=H$
shows IsAnormalSubgroup(G,P,H)
proof-
{
fix g assume g:$g\in G$
with assms(1) have IsAsubgroup($-g\cdot(h\cdot g^{-1}).\ h\in H''$,P) using conj˙group˙is˙group by auto
moreover

```
    from assms(1) g have H≈–g·(h·g⁻¹). h∈H″ using conj˙set˙is˙eqpoll group0˙3˙L2
by auto
    ultimately have –g·(h·g⁻¹). h∈H″=H using assms(2) by auto
    then have –g·(h·g⁻¹). h∈H″⊆H by auto
  }
  then show thesis using cont˙conj˙is˙normal assms(1) by auto
qed
```

The trivial subgroup is then a normal subgroup.

```
corollary(in group0) trivial˙normal˙subgroup:
  shows IsAnormalSubgroup(G,P,–1″)
proof-
  have –1″⊆G using group0˙2˙L2 by auto
  moreover have –1″≠0 by auto moreover
  {
    fix a b assume a∈–1″b∈–1″
    then have a=1b=1 by auto
    then have P⟨a,b⟩=1·1 by auto
    then have P⟨a,b⟩=1 using group0˙2˙L2 by auto
    then have P⟨a,b⟩∈–1″ by auto
  }
  then have –1″–is closed under″P unfolding IsOpClosed˙def by auto
  moreover
  {
    fix a assume a∈–1″
    then have a=1 by auto
    then have a⁻¹=1⁻¹ by auto
    then have a⁻¹=1 using group˙inv˙of˙one by auto
    then have a⁻¹∈–1″ by auto
  }
  then have ∀a∈–1″. a⁻¹∈–1″ by auto ultimately
  have IsAsubgroup(–1″,P) using group0˙3˙T3 by auto moreover
  {
    fix M assume M:IsAsubgroup(M,P) –1″≈M
    then have 1∈M M≈–1″ using eqpoll˙sym group0˙3˙L5 by auto
    then obtain f where f∈bij(M,–1″) unfolding eqpoll˙def by auto
    then have inj:f∈inj(M,–1″) unfolding bij˙def by auto
    then have fun:f:M→–1″ unfolding inj˙def by auto
    {
      fix b assume b∈Mb≠1
      then have fb≠f1 using inj ‹1∈M› unfolding inj˙def by auto
      then have False using ‹b∈M› ‹1∈M› apply˙type[OF fun] by auto
    }
    then have M=–1″ using ‹1∈M› by auto
  }
  ultimately show thesis using only˙one˙equipoll˙sub by auto
qed

lemma(in group0) whole˙normal˙subgroup:
```

```
  shows IsAnormalSubgroup(G,P,G)
  unfolding IsAnormalSubgroup˙def
  using group˙op˙closed inverse˙in˙group
  using group0˙2˙L2 group0˙3˙T3[of G] unfolding IsOpClosed˙def
    by auto
```

Since the whole group and the trivial subgroup are normal, it is natural to define simplicity of groups in the following way:

```
definition
  IsSimple ([˙,˙]–is a simple group″ 89)
 where [G,f]–is a simple group″ ≡ IsAgroup(G,f)∧(∀M. IsAnormalSubgroup(G,f,M)
⟶ M=G∨M=–TheNeutralElement(G,f)″)
```

From the definition follows that if a group has no subgroups, then it is simple.

```
corollary (in group0) noSubgroup˙imp˙simple:
  assumes ∀H. IsAsubgroup(H,P)⟶ H=G∨H=–1″
  shows [G,P]–is a simple group″
proof-
  have IsAgroup(G,P) using groupAssum. moreover
  {
    fix M assume IsAnormalSubgroup(G,P,M)
    then have IsAsubgroup(M,P) unfolding IsAnormalSubgroup˙def by auto
    with assms have M=G∨M=–1″ by auto
  }
  ultimately show thesis unfolding IsSimple˙def by auto
qed
```

Since every subgroup is normal in abelian groups, it follows that commutative simple groups do not have subgroups.

```
corollary (in group0) abelian˙simple˙noSubgroups:
  assumes [G,P]–is a simple group″ P–is commutative on″G
  shows ∀H. IsAsubgroup(H,P)⟶ H=G∨H=–1″
proof(safe)
  fix H assume A:IsAsubgroup(H,P)H ≠ –1″
   then have IsAnormalSubgroup(G,P,H) using Group˙ZF˙2˙4˙L6(1) groupAssum
assms(2)
    by auto
  with assms(1) A show H=G unfolding IsSimple˙def by auto
qed
```

39.2 Finite groups

The subgroup of a finite group is finite.

```
lemma(in group0) finite˙subgroup:
  assumes Finite(G) IsAsubgroup(H,P)
  shows Finite(H)
  using group0˙3˙L2 subset˙Finite assms by force
```

The space of cosets is also finite. In particular, quotient groups.

```
lemma(in group0) finite˙cosets:
  assumes Finite(G) IsAsubgroup(H,P) r=QuotientGroupRel(G,P,H)
  shows Finite(G//r)
proof-
  have fun:–⟨g,r–g″⟩. g∈G″:G→(G//r) unfolding Pi˙def function˙def domain˙def by
auto
  {
    fix C assume C:C∈G//r
      then obtain c where c:c∈C using EquivClass˙1˙L5[OF Group˙ZF˙2˙4˙L1[OF
assms(2)]] assms(3) by auto
    with C have r–c″=C using EquivClass˙1˙L2[OF Group˙ZF˙2˙4˙L3] assms(2,3) by
auto
   with c C have ⟨c,C⟩∈–⟨g,r–g″⟩. g∈G″ using EquivClass˙1˙L1[OF Group˙ZF˙2˙4˙L3]
assms(2,3)
      by auto
    then have –⟨g,r–g″⟩. g∈G″c=C c∈G using apply˙equality fun by auto
    then have ∃c∈G. –⟨g,r–g″⟩. g∈G″c=C by auto
  }
  with fun have surj:–⟨g,r–g″⟩. g∈G″∈surj(G,G//r) unfolding surj˙def by auto more-
over
  from assms(1) obtain n where n∈nat G≈n unfolding Finite˙def by auto
  then have G:G≲n Ord(n) using eqpoll˙imp˙lepoll by auto
  then have G//r≲G using surj˙fun˙inv˙2 surj by auto
  with G(1) have G//r≲n using lepoll˙trans by blast
  then show Finite(G//r) using lepoll˙nat˙imp˙Finite ‹n∈nat› by auto
qed
```

All the cosets are equipollent.

```
lemma(in group0) cosets˙equipoll:
  assumes IsAsubgroup(H,P) r=QuotientGroupRel(G,P,H) g1∈Gg2∈G
  shows r–g1″≈r–g2″
proof-
  from assms(3,4) have GG:(g1⁻¹)·g2∈G using inverse˙in˙group group˙op˙closed by
auto
   then have RightTranslation(G,P,(g1⁻¹)·g2)∈bij(G,G) using trans˙bij(1) by auto
moreover
   have sub2:r–g2″⊆G using EquivClass˙1˙L1[OF Group˙ZF˙2˙4˙L3[OF assms(1)]]
assms(2,4) unfolding quotient˙def by auto
 have sub:r–g1″⊆G using EquivClass˙1˙L1[OF Group˙ZF˙2˙4˙L3[OF assms(1)]] assms(2,3)
unfolding quotient˙def by auto
 ultimately have restrict(RightTranslation(G,P,(g1⁻¹)·g2),r–g1″)∈bij(r–g1″,RightTranslation(G,P,(g1⁻¹)·g2)(r
    using restrict˙bij unfolding bij˙def by auto
  then have r–g1″≈RightTranslation(G,P,(g1⁻¹)·g2)(r–g1″) unfolding eqpoll˙def by
auto
  then have A0:r–g1″≈–RightTranslation(G,P,(g1⁻¹)·g2)t. t∈r–g1″″
    using func˙imagedef[OF group0˙5˙L1(1)[OF GG] sub] by auto
  {
    fix t assume t∈–RightTranslation(G,P,(g1⁻¹)·g2)t. t∈r–g1″″
```

then obtain q where q:t=RightTranslation(G,P,$(g1^{-1})\cdot g2$)q q∈r–g1″ by auto
then have ⟨g1,q⟩∈r q∈G using image˙iff sub by auto
then have $g1\cdot(q^{-1})\in H$ $q^{-1}\in G$ using assms(2) inverse˙in˙group unfolding QuotientGroupRel˙def by auto
from q(1) have t:t=q·$((g1^{-1})\cdot g2)$ using group0˙5˙L2(1)[OF GG] q(2) sub by auto
then have $g2\cdot t^{-1}=g2\cdot(q\cdot((g1^{-1})\cdot g2))^{-1}$ by auto
then have $g2\cdot t^{-1}=g2\cdot(((g1^{-1})\cdot g2)^{-1}\cdot q^{-1})$ using group˙inv˙of˙two[OF ‹q∈G› GG] by auto
then have $g2\cdot t^{-1}=g2\cdot(((g2^{-1})\cdot g1^{-1-1})\cdot q^{-1})$ using group˙inv˙of˙two[OF inverse˙in˙group[OF assms(3)]
assms(4)] by auto
then have $g2\cdot t^{-1}=g2\cdot(((g2^{-1})\cdot g1)\cdot q^{-1})$ using group˙inv˙of˙inv assms(3) by auto moreover
have t∈G using t ‹q∈G› ‹g2∈G› inverse˙in˙group[OF assms(3)] group˙op˙closed by auto
have $(g2^{-1})\cdot g1\in G$ using assms(3) inverse˙in˙group[OF assms(4)] group˙op˙closed by auto
with assms(4) ‹$q^{-1}\in G$› have $g2\cdot(((g2^{-1})\cdot g1)\cdot q^{-1})=g2\cdot((g2^{-1})\cdot g1)\cdot q^{-1}$ using group˙oper˙assoc by auto
moreover have $g2\cdot((g2^{-1})\cdot g1)=g2\cdot(g2^{-1})\cdot g1$ using assms(3) inverse˙in˙group[OF assms(4)] assms(4)
group˙oper˙assoc by auto
then have $g2\cdot((g2^{-1})\cdot g1)=g1$ using group0˙2˙L6[OF assms(4)] group0˙2˙L2 assms(3) by auto ultimately
have $g2\cdot t^{-1}=g1\cdot q^{-1}$ by auto
with ‹$g1\cdot(q^{-1})\in H$› have $g2\cdot t^{-1}\in H$ by auto
then have ⟨g2,t⟩∈r using assms(2) unfolding QuotientGroupRel˙def using assms(4) ‹t∈G› by auto
then have t∈r–g2″ using image˙iff assms(4) by auto
}
then have A1:–RightTranslation(G,P,$(g1^{-1})\cdot g2$)t. t∈r–g1″″⊆r–g2″ by auto
{
fix t assume t∈r–g2″
then have ⟨g2,t⟩∈r t∈G using sub2 image˙iff by auto
then have H:$g2\cdot t^{-1}\in H$ using assms(2) unfolding QuotientGroupRel˙def by auto
then have G:$g2\cdot t^{-1}\in G$ using group0˙3˙L2 assms(1) by auto
then have $g1\cdot(g1^{-1}\cdot(g2\cdot t^{-1}))=g1\cdot g1^{-1}\cdot(g2\cdot t^{-1})$ using group˙oper˙assoc[OF assms(3) inverse˙in˙group[OF assms(3)]]
by auto
then have $g1\cdot(g1^{-1}\cdot(g2\cdot t^{-1}))=g2\cdot t^{-1}$ using group0˙2˙L6[OF assms(3)] group0˙2˙L2 G by auto
with H have HH:$g1\cdot(g1^{-1}\cdot(g2\cdot t^{-1}))\in H$ by auto
have GGG:$t\cdot g2^{-1}\in G$ using ‹t∈G› inverse˙in˙group[OF assms(4)] group˙op˙closed by auto
have $(t\cdot g2^{-1})^{-1}=g2^{-1-1}\cdot t^{-1}$ using group˙inv˙of˙two[OF ‹t∈G› inverse˙in˙group[OF assms(4)]] by auto
also have …$=g2\cdot t^{-1}$ using group˙inv˙of˙inv[OF assms(4)] by auto
ultimately have $(t\cdot g2^{-1})^{-1}=g2\cdot t^{-1}$ by auto
then have $g1^{-1}\cdot(t\cdot g2^{-1})^{-1}=g1^{-1}\cdot(g2\cdot t^{-1})$ by auto

then have $((t\cdot g2^{-1})\cdot g1)^{-1}=g1^{-1}\cdot(g2\cdot t^{-1})$ using group˙inv˙of˙two[OF GGG assms(3)] by auto
then have HHH:$g1\cdot((t\cdot g2^{-1})\cdot g1)^{-1}\in H$ using HH by auto
have $(t\cdot g2^{-1})\cdot g1\in G$ using assms(3) ‹t∈G› inverse˙in˙group[OF assms(4)] group˙op˙closed by auto
with HHH have $\langle g1,(t\cdot g2^{-1})\cdot g1\rangle\in r$ using assms(2,3) unfolding QuotientGroupRel˙def by auto
then have rg1:$t\cdot g2^{-1}\cdot g1\in r-g1''$ using image˙iff by auto
have $t\cdot g2^{-1}\cdot g1\cdot((g1^{-1})\cdot g2)=t\cdot(g2^{-1}\cdot g1)\cdot((g1^{-1})\cdot g2)$ using group˙oper˙assoc[OF ‹t∈G› inverse˙in˙group[OF assms(4)] assms(3)]
by auto
also have $\ldots=t\cdot((g2^{-1}\cdot g1)\cdot((g1^{-1})\cdot g2))$ using group˙oper˙assoc[OF ‹t∈G› group˙op˙closed[OF inverse˙in˙group[OF assms(4)] assms(3)] GG]
by auto
also have $\ldots=t\cdot(g2^{-1}\cdot(g1\cdot((g1^{-1})\cdot g2)))$ using group˙oper˙assoc[OF inverse˙in˙group[OF assms(4)] assms(3) GG] by auto
also have $\ldots=t\cdot(g2^{-1}\cdot(g1\cdot(g1^{-1})\cdot g2))$ using group˙oper˙assoc[OF assms(3) inverse˙in˙group[OF assms(3)] assms(4)] by auto
also have $\ldots=t$ using group0˙2˙L6[OF assms(3)]group0˙2˙L6[OF assms(4)] group0˙2˙L2 ‹t∈G› assms(4) by auto
ultimately have $t\cdot g2^{-1}\cdot g1\cdot((g1^{-1})\cdot g2)=t$ by auto
then have RightTranslation(G,P,$(g1^{-1})\cdot g2$)$(t\cdot g2^{-1}\cdot g1)=t$ using group0˙5˙L2(1)[OF GG] ‹$(t\cdot g2^{-1})\cdot g1\in G$› by auto
then have $t\in$–RightTranslation(G,P,$(g1^{-1})\cdot g2$)t. $t\in r-g1'''$ using rg1 by force
}
then have $r-g2''\subseteq$–RightTranslation(G,P,$(g1^{-1})\cdot g2$)t. $t\in r-g1'''$ by blast
with A1 have $r-g2''=$–RightTranslation(G,P,$(g1^{-1})\cdot g2$)t. $t\in r-g1'''$ by auto
with A0 show thesis by auto
qed

The order of a subgroup multiplied by the order of the space of cosets is the order of the group. We only prove the theorem for finite groups.

theorem(in group0) Lagrange:
assumes Finite(G) IsAsubgroup(H,P) r=QuotientGroupRel(G,P,H)
shows —G—=—H— #* —G//r—
proof-
have Finite(G//r) using assms finite˙cosets by auto moreover
have un:$\bigcup$(G//r)=G using Union˙quotient Group˙ZF˙2˙4˙L3 assms(2,3) by auto
then have Finite($\bigcup$(G//r)) using assms(1) by auto moreover
have $\forall$c1∈(G//r). $\forall$c2∈(G//r). c1≠c2 $\longrightarrow$ c1∩c2=0 using quotient˙disj[OF Group˙ZF˙2˙4˙L3[OF assms(2)]]
assms(3) by auto moreover
have $\forall$aa∈G. aa∈H $\longleftrightarrow$ $\langle$aa,**1**$\rangle\in r$ using Group˙ZF˙2˙4˙L5C assms(3) by auto
then have $\forall$aa∈G. aa∈H $\longleftrightarrow$ $\langle$**1**,aa$\rangle\in r$ using Group˙ZF˙2˙4˙L2 assms(2,3) unfolding sym˙def
by auto
then have $\forall$aa∈G. aa∈H $\longleftrightarrow$ aa∈$r-$**1**$''$ using image˙iff by auto
then have H:H=$r-$**1**$''$ using group0˙3˙L2[OF assms(2)] assms(3) unfolding QuotientGroupRel˙def by auto

```
  {
    fix c assume c∈(G//r)
    then obtain g where g∈G c=r–g″ unfolding quotient˙def by auto
    then have c≈r–1″ using cosets˙equipoll[OF assms(2,3)] group0˙2˙L2 by auto
    then have —c—=—H— using H cardinal˙cong by auto
  }
  then have ∀c∈(G//r). —c—=—H— by auto ultimately
  show thesis using card˙partition un by auto
qed
```

39.3 Subgroups generated by sets

Given a subset of a group, we can ask ourselves which is the smallest group that contains that set; if it even exists.

```
lemma(in group0) inter˙subgroups:
  assumes ∀H∈ℌ. IsAsubgroup(H,P) ℌ≠0
  shows IsAsubgroup(⋂ℌ,P)
proof-
  from assms have 1∈⋂ℌ using group0˙3˙L5 by auto
  then have ⋂ℌ≠0 by auto moreover
  {
    fix A B assume A∈⋂ℌB∈⋂ℌ
    then have ∀H∈ℌ. A∈H∧B∈H by auto
    then have ∀H∈ℌ. A·B∈H using assms(1) group0˙3˙L6 by auto
    then have A·B∈⋂ℌ using assms(2) by auto
  }
  then have (⋂ℌ)–is closed under″P using IsOpClosed˙def by auto moreover
  {
    fix A assume A∈⋂ℌ
    then have ∀H∈ℌ. A∈H by auto
    then have ∀H∈ℌ. A⁻¹∈H using assms(1) group0˙3˙T3A by auto
    then have A⁻¹∈⋂ℌ using assms(2) by auto
  }
  then have ∀A∈⋂ℌ. A⁻¹∈⋂ℌ by auto moreover
  have ⋂ℌ⊆G using assms(1,2) group0˙3˙L2 by force
  ultimately show thesis using group0˙3˙T3 by auto
qed
```

As the previous lemma states, the subgroup that contains a subset can be defined as an intersection of subgroups.

```
definition(in group0)
  SubgroupGenerated (⟨˙⟩G 80)
  where ⟨X⟩G ≡ ⋂–H∈Pow(G). X⊆H ∧ IsAsubgroup(H,P)″

theorem(in group0) subgroupGen˙is˙subgroup:
  assumes X⊆G
  shows IsAsubgroup(⟨X⟩G,P)
proof-
```

have restrict(P,G×G)=P using group˙oper˙assocA restrict˙idem unfolding Pi˙def by auto
then have IsAsubgroup(G,P) unfolding IsAsubgroup˙def using groupAssum by auto
with assms have G∈–H∈Pow(G). X⊆H ∧ IsAsubgroup(H,P)″ by auto
then have –H∈Pow(G). X⊆H ∧ IsAsubgroup(H,P)″≠0 by auto
then show thesis using inter˙subgroups unfolding SubgroupGenerated˙def by auto
qed

39.4 Homomorphisms

A homomorphism is a function between groups that preserves group operations.

definition
Homomor (˙–is a homomorphism″–˙,˙″→–˙,˙″ 85)
where IsAgroup(G,P) ⟹ IsAgroup(H,F) ⟹ Homomor(f,G,P,H,F) ≡ ∀g1∈G. ∀g2∈G. f(P⟨g1,g2⟩)=F⟨fg1,fg2⟩

Now a lemma about the definition:

lemma homomor˙eq:
assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F) g1∈G g2∈G
shows f(P⟨g1,g2⟩)=F⟨fg1,fg2⟩
using assms Homomor˙def by auto

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

definition
End
where End(G,P) ≡ –f:G→G. Homomor(f,G,P,G,P)″

The set of endomorphisms forms a submonoid of the monoid of function from a set to that set under composition.

lemma(in group0) end˙composition:
assumes f1∈End(G,P)f2∈End(G,P)
shows Composition(G)⟨f1,f2⟩∈End(G,P)
proof-
from assms have fun:f1:G→Gf2:G→G unfolding End˙def by auto
then have fun2:f1 O f2:G→G using comp˙fun by auto
have comp:Composition(G)⟨f1,f2⟩=f1 O f2 using func˙ZF˙5˙L2 fun by auto
{
fix g1 g2 assume AS2:g1∈Gg2∈G
then have g1g2:g1·g2∈G using group˙op˙closed by auto
from fun2 have (f1 O f2)(g1·g2)=f1(f2(g1·g2)) using comp˙fun˙apply fun(2) g1g2 by auto
also have . . .=f1((f2g1)·(f2g2)) using assms(2) unfolding End˙def Homomor˙def[OF groupAssum groupAssum]
using AS2 by auto moreover
have f2g1∈Gf2g2∈G using fun(2) AS2 apply˙type by auto ultimately

```
    have (f1 O f2)(g1·g2)=(f1(f2g1))·(f1(f2g2)) using assms(1) unfolding End˙def
Homomor˙def[OF groupAssum groupAssum]
     using AS2 by auto
    then have (f1 O f2)(g1·g2)=((f1 O f2)g1)·((f1 O f2)g2) using comp˙fun˙apply
fun(2) AS2 by auto
  }
  then have ∀g1∈G. ∀g2∈G. (f1 O f2)(g1·g2)=((f1 O f2)g1)·((f1 O f2)g2) by auto
  then have (f1 O f2)∈End(G,P) unfolding End˙def Homomor˙def[OF groupAssum
groupAssum] using fun2 by auto
  with comp show Composition(G)⟨f1,f2⟩∈End(G,P) by auto
qed

theorem(in group0) end˙comp˙monoid:
  shows IsAmonoid(End(G,P),restrict(Composition(G),End(G,P)×End(G,P)))
 and TheNeutralElement(End(G,P),restrict(Composition(G),End(G,P)×End(G,P)))=id(G)
proof-
  have fun:id(G):G→G unfolding id˙def by auto
  {
    fix g h assume g∈Gh∈G
    then have id:g·h∈Gid(G)g=gid(G)h=h using group˙op˙closed by auto
    then have id(G)(g·h)=g·h unfolding id˙def by auto
    with id(2,3) have id(G)(g·h)=(id(G)g)·(id(G)h) by auto
  }
  with fun have id(G)∈End(G,P) unfolding End˙def Homomor˙def[OF groupAssum
groupAssum] by auto moreover
  from Group˙ZF˙2˙5˙L2(2) have A0:id(G)=TheNeutralElement(G → G, Composi-
tion(G)) by auto ultimately
  have A1:TheNeutralElement(G → G, Composition(G))∈End(G,P) by auto more-
over
  have A2:End(G,P)⊆G→G unfolding End˙def by auto moreover
  from end˙composition have A3:End(G,P)–is closed under˝Composition(G) unfold-
ing IsOpClosed˙def by auto
 ultimately show IsAmonoid(End(G,P),restrict(Composition(G),End(G,P)×End(G,P)))

    using monoid0.group0˙1˙T1 unfolding monoid0˙def using Group˙ZF˙2˙5˙L2(1)
    by force
  have IsAmonoid(G→G,Composition(G)) using Group˙ZF˙2˙5˙L2(1) by auto
 with A0 A1 A2 A3 show TheNeutralElement(End(G,P),restrict(Composition(G),End(G,P)×End(G,P)))=id(G
    using group0˙1˙L6 by auto
qed
```

The set of endomorphisms is closed under pointwise addition. This is so because the group is abelian.

```
theorem(in group0) end˙pointwise˙addition:
 assumes f∈End(G,P)g∈End(G,P)P–is commutative on˝GF = P –lifted to function
space over˝ G
  shows F⟨f,g⟩∈End(G,P)
proof-
  from assms(1,2) have fun:f∈G→Gg∈G→G unfolding End˙def by auto
```

then have fun2:F⟨f,g⟩:G→G using monoid0.Group˙ZF˙2˙1˙L0 group0˙2˙L1 assms(4) by auto
{
fix g1 g2 assume AS:g1∈Gg2∈G
then have g1·g2∈G using group˙op˙closed by auto
then have (F⟨f,g⟩)(g1·g2)=(f(g1·g2))·(g(g1·g2)) using Group˙ZF˙2˙1˙L3 fun assms(4) by auto
also have ...=(f(g1)·f(g2))·(g(g1)·g(g2)) using assms unfolding End˙def Homomor˙def[OF groupAssum groupAssum]
using AS by auto ultimately
have (F⟨f,g⟩)(g1·g2)=(f(g1)·f(g2))·(g(g1)·g(g2)) by auto moreover
have fg1∈Gfg2∈Ggg1∈Ggg2∈G using fun apply˙type AS by auto ultimately
have (F⟨f,g⟩)(g1·g2)=(f(g1)·g(g1))·(f(g2)·g(g2)) using group0˙4˙L8(3) assms(3)
by auto
with AS have (F⟨f,g⟩)(g1·g2)=((F⟨f,g⟩)g1)·((F⟨f,g⟩)g2)
using Group˙ZF˙2˙1˙L3 fun assms(4) by auto
}
with fun2 show thesis unfolding End˙def Homomor˙def[OF groupAssum groupAssum] by auto
qed

The inverse of an abelian group is an endomorphism.

lemma(in group0) end˙inverse˙group:
assumes P–is commutative on˝G
shows GroupInv(G,P)∈End(G,P)
proof-
{
fix s t assume AS:s∈Gt∈G
then have elinv:s^{-1}∈Gt^{-1}∈G using inverse˙in˙group by auto
have $(s\cdot t)^{-1}=t^{-1}\cdot s^{-1}$ using group˙inv˙of˙two AS by auto
then have $(s\cdot t)^{-1}=s^{-1}\cdot t^{-1}$ using assms(1) elinv unfolding IsCommutative˙def by auto
}
then have ∀s∈G. ∀t∈G. GroupInv(G,P)(s·t)=GroupInv(G,P)(s)·GroupInv(G,P)(t) by auto
with group0˙2˙T2 groupAssum show thesis unfolding End˙def using Homomor˙def by auto
qed

The set of homomorphisms of an abelian group is an abelian subgroup of the group of functions from a set to a group, under pointwise multiplication.

theorem(in group0) end˙addition˙group:
assumes P–is commutative on˝G F = P –lifted to function space over˝ G
shows IsAgroup(End(G,P),restrict(F,End(G,P)×End(G,P))) restrict(F,End(G,P)×End(G,P))–is commutative on˝End(G,P)
proof-
from end˙comp˙monoid(1) monoid0.group0˙1˙L3A have End(G,P)≠0 unfolding monoid0˙def by auto
moreover have End(G,P)⊆G→G unfolding End˙def by auto moreover

```
 have End(G,P)–is closed under″F unfolding IsOpClosed˙def using end˙pointwise˙addition
    assms(1,2) by auto moreover
  {
    fix ff assume AS:ff∈End(G,P)
   then have restrict(Composition(G),End(G,P)×End(G,P))⟨GroupInv(G,P), ff⟩∈End(G,P)
using monoid0.group0˙1˙L1
    unfolding monoid0˙def using end˙composition(1) end˙inverse˙group[OF assms(1)]
      by force
   then have Composition(G)⟨GroupInv(G,P), ff⟩∈End(G,P) using AS end˙inverse˙group[OF
assms(1)]
      by auto
    then have GroupInv(G,P) O ff∈End(G,P) using func˙ZF˙5˙L2 AS group0˙2˙T2
groupAssum unfolding
      End˙def by auto
    then have GroupInv(G→G,F)ff∈End(G,P) using Group˙ZF˙2˙1˙L6 assms(2) AS
unfolding End˙def
      by auto
  }
  then have ∀ff∈End(G,P). GroupInv(G→G,F)ff∈End(G,P) by auto ultimately
 show IsAgroup(End(G,P),restrict(F,End(G,P)×End(G,P))) using group0.group0˙3˙T3
Group˙ZF˙2˙1˙T2[OF assms(2)] unfolding IsAsubgroup˙def group0˙def
    by auto
 show restrict(F,End(G,P)×End(G,P))–is commutative on″End(G,P) using Group˙ZF˙2˙1˙L7[OF
assms(2,1)] unfolding End˙def IsCommutative˙def by auto
qed

lemma(in group0) distributive˙comp˙pointwise:
  assumes P–is commutative on″G F = P –lifted to function space over″ G
 shows IsDistributive(End(G,P),restrict(F,End(G,P)×End(G,P)),restrict(Composition(G),End(G,P)×End(G,
proof-
  {
    fix b c d assume AS:b∈End(G,P)c∈End(G,P)d∈End(G,P)
   have ig1:Composition(G) ⟨b, F ⟨c, d⟩⟩ =b O (F⟨c,d⟩) using monoid0.Group˙ZF˙2˙1˙L0[OF
group0˙2˙L1 assms(2)]
      AS unfolding End˙def using func˙ZF˙5˙L2 by auto
    have ig2:F ⟨Composition(G) ⟨b , c⟩,Composition(G) ⟨b , d⟩⟩=F ⟨b O c,b O d⟩
using AS unfolding End˙def using func˙ZF˙5˙L2 by auto
   have comp1fun:(b O (F⟨c,d⟩)):G→G using monoid0.Group˙ZF˙2˙1˙L0[OF group0˙2˙L1
assms(2)] comp˙fun AS unfolding End˙def by force
     have comp2fun:(F ⟨b O c,b O d⟩):G→G using monoid0.Group˙ZF˙2˙1˙L0[OF
group0˙2˙L1 assms(2)] comp˙fun AS unfolding End˙def by force
    {
      fix g assume gG:g∈G
    then have (b O (F⟨c,d⟩))g=b((F⟨c,d⟩)g) using comp˙fun˙apply monoid0.Group˙ZF˙2˙1˙L0[OF
group0˙2˙L1 assms(2)]
        AS(2,3) unfolding End˙def by force
       also have ...=b(c(g)·d(g)) using Group˙ZF˙2˙1˙L3[OF assms(2)] AS(2,3) gG
unfolding End˙def by auto
      ultimately have (b O (F⟨c,d⟩))g=b(c(g)·d(g)) by auto moreover
```

have cg∈Gdg∈G using AS(2,3) unfolding End˙def using apply˙type gG by auto
ultimately have (b O (F⟨c,d⟩))g=(b(cg))·(b(dg)) using AS(1) unfolding End˙def Homomor˙def[OF groupAssum groupAssum] by auto
then have (b O (F⟨c,d⟩))g=((b O c)g)·((b O d)g) using comp˙fun˙apply gG AS(2,3)
unfolding End˙def by auto
then have (b O (F⟨c,d⟩))g=(F⟨b O c,b O d⟩)g using gG Group˙ZF˙2˙1˙L3[OF assms(2) comp˙fun comp˙fun gG]
AS unfolding End˙def by auto
}
then have ∀g∈G. (b O (F⟨c,d⟩))g=(F⟨b O c,b O d⟩)g by auto
then have b O (F⟨c,d⟩)=F⟨b O c,b O d⟩ using fun˙extension[OF comp1fun comp2fun] by auto
with ig1 ig2 have Composition(G) ⟨b, F ⟨c, d⟩⟩ =F ⟨Composition(G) ⟨b , c⟩,Composition(G) ⟨b , d⟩⟩ by auto moreover
have F ⟨c, d⟩=restrict(F,End(G,P)×End(G,P)) ⟨c, d⟩ using AS(2,3) restrict by auto moreover
have Composition(G) ⟨b , c⟩=restrict(Composition(G),End(G,P)×End(G,P)) ⟨b , c⟩ Composition(G) ⟨b , d⟩=restrict(Composition(G),End(G,P)×End(G,P)) ⟨b , d⟩
using restrict AS by auto moreover
have Composition(G) ⟨b, F ⟨c, d⟩⟩ =restrict(Composition(G),End(G,P)×End(G,P)) ⟨b, F ⟨c, d⟩⟩ using AS(1)
end˙pointwise˙addition[OF AS(2,3) assms] by auto
moreover have F ⟨Composition(G) ⟨b , c⟩,Composition(G) ⟨b , d⟩⟩=restrict(F,End(G,P)×End(G,P)) ⟨Composition(G) ⟨b , c⟩,Composition(G) ⟨b , d⟩⟩
using end˙composition[OF AS(1,2)] end˙composition[OF AS(1,3)] by auto ultimately
have eq1:restrict(Composition(G),End(G,P)×End(G,P)) ⟨b, restrict(F,End(G,P)×End(G,P)) ⟨c, d⟩⟩ =restrict(F,End(G,P)×End(G,P)) ⟨restrict(Composition(G),End(G,P)×End(G,P)) ⟨b , c⟩,restrict(Composition(G),End(G,P)×End(G,P))⟨b , d⟩⟩
by auto
have ig1:Composition(G) ⟨ F ⟨c, d⟩,b⟩ = (F⟨c,d⟩) O b using monoid0.Group˙ZF˙2˙1˙L0[OF group0˙2˙L1 assms(2)]
AS unfolding End˙def using func˙ZF˙5˙L2 by auto
have ig2:F ⟨Composition(G) ⟨c , b⟩,Composition(G) ⟨d , b⟩⟩=F ⟨c O b,d O b⟩ using AS unfolding End˙def using func˙ZF˙5˙L2 by auto
have comp1fun:((F⟨c,d⟩) O b):G→G using monoid0.Group˙ZF˙2˙1˙L0[OF group0˙2˙L1 assms(2)] comp˙fun AS unfolding End˙def by force
have comp2fun:(F ⟨c O b,d O b⟩):G→G using monoid0.Group˙ZF˙2˙1˙L0[OF group0˙2˙L1 assms(2)] comp˙fun AS unfolding End˙def by force
{
fix g assume gG:g∈G
then have bg:bg∈G using AS(1) unfolding End˙def using apply˙type by auto
from gG have ((F⟨c,d⟩) O b)g=(F⟨c,d⟩)(bg) using comp˙fun˙apply AS(1) unfolding End˙def by force
also have ...=(c(bg))·(d(bg)) using Group˙ZF˙2˙1˙L3[OF assms(2)] AS(2,3) bg unfolding End˙def by auto
also have ...=((c O b)g)·((d O b)g) using comp˙fun˙apply gG AS unfolding

```
End˙def by auto
      also have ...=(F⟨c O b,d O b⟩)g using gG Group˙ZF˙2˙1˙L3[OF assms(2)
comp˙fun comp˙fun gG]
      AS unfolding End˙def by auto
    ultimately have((F⟨c,d⟩) O b)g=(F⟨c O b,d O b⟩)g by auto
  }
  then have ∀g∈G. ((F⟨c,d⟩) O b)g=(F⟨c O b,d O b⟩)g by auto
    then have (F⟨c,d⟩) O b=F⟨c O b,d O b⟩ using fun˙extension[OF comp1fun
comp2fun] by auto
    with ig1 ig2 have Composition(G) ⟨F  ⟨c, d⟩,b⟩ =F ⟨Composition(G) ⟨c ,
b⟩,Composition(G) ⟨d , b⟩⟩ by auto moreover
  have F  ⟨c, d⟩=restrict(F,End(G,P)×End(G,P))  ⟨c, d⟩ using AS(2,3) restrict by
auto moreover
  have Composition(G) ⟨c , b⟩=restrict(Composition(G),End(G,P)×End(G,P)) ⟨c
, b⟩ Composition(G) ⟨d , b⟩=restrict(Composition(G),End(G,P)×End(G,P)) ⟨d ,
b⟩
    using restrict AS by auto moreover
  have Composition(G) ⟨F  ⟨c, d⟩,b⟩ =restrict(Composition(G),End(G,P)×End(G,P))
⟨F  ⟨c, d⟩,b⟩ using AS(1)
    end˙pointwise˙addition[OF AS(2,3) assms] by auto
  moreover have F ⟨Composition(G) ⟨c , b⟩,Composition(G) ⟨d , b⟩⟩=restrict(F,End(G,P)×End(G,P))
⟨Composition(G) ⟨c , b⟩,Composition(G) ⟨d , b⟩⟩
      using end˙composition[OF AS(2,1)] end˙composition[OF AS(3,1)] by auto
ultimately
  have eq2:restrict(Composition(G),End(G,P)×End(G,P)) ⟨ restrict(F,End(G,P)×End(G,P))
⟨c, d⟩,b⟩ =restrict(F,End(G,P)×End(G,P)) ⟨restrict(Composition(G),End(G,P)×End(G,P))
⟨c ,b⟩,restrict(Composition(G),End(G,P)×End(G,P))⟨d , b⟩⟩
    by auto
  with eq1 have (restrict(Composition(G),End(G,P)×End(G,P)) ⟨b, restrict(F,End(G,P)×End(G,P))
⟨c, d⟩⟩ =restrict(F,End(G,P)×End(G,P)) ⟨restrict(Composition(G),End(G,P)×End(G,P))
⟨b , c⟩,restrict(Composition(G),End(G,P)×End(G,P))⟨b , d⟩⟩)∧
   (restrict(Composition(G),End(G,P)×End(G,P)) ⟨ restrict(F,End(G,P)×End(G,P))
⟨c, d⟩,b⟩ =restrict(F,End(G,P)×End(G,P)) ⟨restrict(Composition(G),End(G,P)×End(G,P))
⟨c ,b⟩,restrict(Composition(G),End(G,P)×End(G,P))⟨d , b⟩⟩)
    by auto
  }
  then show thesis unfolding IsDistributive˙def by auto
qed
```

The endomorphisms of an abelian group is in fact a ring with the previous operations.

```
theorem(in group0) end˙is˙ring:
  assumes P–is commutative on˝G F = P –lifted to function space over˝ G
 shows IsAring(End(G,P),restrict(F,End(G,P)×End(G,P)),restrict(Composition(G),End(G,P)×End(G,P)))
   unfolding IsAring˙def using end˙addition˙group[OF assms] end˙comp˙monoid(1)
distributive˙comp˙pointwise[OF assms]
  by auto
```

39.5 First isomorphism theorem

Now we will prove that any homomorphism $f : G \to H$ defines a bijective homomorphism between G/H and $f(G)$.

A group homomorphism sends the neutral element to the neutral element and commutes with the inverse.

```
lemma image˙neutral:
  assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F) f:G→H
  shows fTheNeutralElement(G,P)=TheNeutralElement(H,F)
proof-
 have g:TheNeutralElement(G,P)=P⟨TheNeutralElement(G,P),TheNeutralElement(G,P)⟩
TheNeutralElement(G,P)∈G
    using assms(1) group0.group0˙2˙L2 unfolding group0˙def by auto
 from g(1) have fTheNeutralElement(G,P)=f(P⟨TheNeutralElement(G,P),TheNeutralElement(G,P)⟩)
by auto
  also have ...=F⟨fTheNeutralElement(G,P),fTheNeutralElement(G,P)⟩
    using assms(3) unfolding Homomor˙def[OF assms(1,2)] using g(2) by auto
 ultimately have fTheNeutralElement(G,P)=F⟨fTheNeutralElement(G,P),fTheNeutralElement(G,P)⟩
by auto moreover
  have h:fTheNeutralElement(G,P)∈H using g(2) apply˙type[OF assms(4)] by auto
 then have fTheNeutralElement(G,P)=F⟨fTheNeutralElement(G,P),TheNeutralElement(H,F)⟩
    using assms(2) group0.group0˙2˙L2 unfolding group0˙def by auto ultimately
 have F⟨fTheNeutralElement(G,P),TheNeutralElement(H,F)⟩=F⟨fTheNeutralElement(G,P),fTheNeutralElem
by auto
 with h have LeftTranslation(H,F,fTheNeutralElement(G,P))TheNeutralElement(H,F)=LeftTranslation(H,F,f
      using group0.group0˙5˙L2(2)[OF ˙ h] assms(2) group0.group0˙2˙L2 unfolding
group0˙def by auto
   moreover have LeftTranslation(H,F,fTheNeutralElement(G,P))∈bij(H,H) using
group0.trans˙bij(2)
    assms(2) h unfolding group0˙def by auto
   then have LeftTranslation(H,F,fTheNeutralElement(G,P))∈inj(H,H) unfolding
bij˙def by auto ultimately
 show fTheNeutralElement(G,P)=TheNeutralElement(H,F) using h assms(2) group0.group0˙2˙L2
unfolding inj˙def group0˙def
    by force
qed

lemma image˙inv:
  assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F) f:G→H g∈G
  shows f( GroupInv(G,P)g)=GroupInv(H,F) (fg)
proof-
  have im:fg∈H using apply˙type[OF assms(4,5)].
 have inv:GroupInv(G,P)g∈G using group0.inverse˙in˙group[OF ˙ assms(5)] assms(1)
unfolding group0˙def by auto
  then have inv2:f(GroupInv(G,P)g)∈Husing apply˙type[OF assms(4)] by auto
 have fTheNeutralElement(G,P)=f(P⟨g,GroupInv(G,P)g⟩) using assms(1,5) group0.group0˙2˙L6
    unfolding group0˙def by auto
 also have ...=F⟨fg,f(GroupInv(G,P)g)⟩ using assms(3) unfolding Homomor˙def[OF
```

```
assms(1,2)] using
    assms(5) inv by auto
   ultimately have TheNeutralElement(H,F)=F⟨fg,f(GroupInv(G,P)g)⟩ using image˙neutral[OF assms(1-4)]
    by auto
  then show thesis using group0.group0˙2˙L9(2)[OF ˙ im inv2] assms(2) unfolding group0˙def by auto
qed
```

The kernel of an homomorphism is a normal subgroup.

```
theorem kerner˙normal˙sub:
  assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F) f:G→H
  shows IsAnormalSubgroup(G,P,f-–TheNeutralElement(H,F)″)
proof-
  have xy:∀x y. ⟨x, y⟩ ∈ f ⟶ (∀y'. ⟨x, y'⟩ ∈ f ⟶ y = y') using assms(4) unfolding Pi˙def function˙def
    by force
  {
   fix g1 g2 assume g1∈f-–TheNeutralElement(H,F)″g2∈f-–TheNeutralElement(H,F)″
    then have ⟨g1,TheNeutralElement(H,F)⟩∈f⟨g2,TheNeutralElement(H,F)⟩∈f
      using vimage˙iff by auto moreover
    then have G:g1∈Gg2∈G using assms(4) unfolding Pi˙def by auto
    then have ⟨g1,fg1⟩∈f⟨g2,fg2⟩∈f using apply˙Pair[OF assms(4)] by auto moreover
    note xy ultimately
   have fg1=TheNeutralElement(H,F)fg2=TheNeutralElement(H,F) by auto moreover
   have f(P⟨g1,g2⟩)=F⟨fg1,fg2⟩ using assms(3) G unfolding Homomor˙def[OF assms(1,2)] by auto
   ultimately have f(P⟨g1,g2⟩)=F⟨TheNeutralElement(H,F),TheNeutralElement(H,F)⟩ by auto
    also have ...=TheNeutralElement(H,F) using group0.group0˙2˙L2 assms(2) unfolding group0˙def
      by auto
    ultimately have f(P⟨g1,g2⟩)=TheNeutralElement(H,F) by auto moreover
      from G have P⟨g1,g2⟩∈G using group0.group˙op˙closed assms(1) unfolding group0˙def by auto
      ultimately have ⟨P⟨g1,g2⟩,TheNeutralElement(H,F)⟩∈f using apply˙Pair[OF assms(4)] by force
    then have P⟨g1,g2⟩∈f-–TheNeutralElement(H,F)″ using vimage˙iff by auto
  }
 then have f-–TheNeutralElement(H,F)″ –is closed under″P unfolding IsOpClosed˙def by auto
  moreover have A:f-–TheNeutralElement(H,F)″ ⊆ G using func1˙1˙L3 assms(4) by auto
   moreover have fTheNeutralElement(G,P)=TheNeutralElement(H,F) using image˙neutral
    assms by auto
 then have ⟨TheNeutralElement(G,P),TheNeutralElement(H,F)⟩∈f using apply˙Pair[OF assms(4)]
```

group0.group0˙2˙L2 assms(1) unfolding group0˙def by force
then have TheNeutralElement(G,P)∈f-−TheNeutralElement(H,F)″ using vimage˙iff by auto
then have f-−TheNeutralElement(H,F)″≠0 by auto moreover
{
fix x assume x∈f-−TheNeutralElement(H,F)″
then have ⟨x,TheNeutralElement(H,F)⟩∈f and x:x∈G using vimage˙iff A by auto moreover
from x have ⟨x,fx⟩∈f using apply˙Pair[OF assms(4)] by auto ultimately
have fx=TheNeutralElement(H,F) using xy by auto moreover
have f(GroupInv(G,P)x)=GroupInv(H,F)(fx) using x image˙inv assms by auto
ultimately have f(GroupInv(G,P)x)=GroupInv(H,F)TheNeutralElement(H,F) by auto
then have f(GroupInv(G,P)x)=TheNeutralElement(H,F) using group0.group˙inv˙of˙one assms(2) unfolding group0˙def by auto moreover
have ⟨GroupInv(G,P)x,f(GroupInv(G,P)x)⟩∈f using apply˙Pair[OF assms(4)] x group0.inverse˙in˙group assms(1) unfolding group0˙def by auto
ultimately have ⟨GroupInv(G,P)x,TheNeutralElement(H,F)⟩∈f by auto
then have GroupInv(G,P)x∈f-−TheNeutralElement(H,F)″ using vimage˙iff by auto
}
then have ∀x∈f-−TheNeutralElement(H,F)″. GroupInv(G,P)x∈f-−TheNeutralElement(H,F)″ by auto
ultimately have SS:IsAsubgroup(f-−TheNeutralElement(H,F)″,P) using group0.group0˙3˙T3 assms(1) unfolding group0˙def by auto
{
fix g h assume AS:g∈Gh∈f-−TheNeutralElement(H,F)″
from AS(1) have im:fg∈H using assms(4) apply˙type by auto
then have iminv:GroupInv(H,F)(fg)∈H using assms(2) group0.inverse˙in˙group unfolding group0˙def by auto
from AS have h∈G and inv:GroupInv(G,P)g∈G using A group0.inverse˙in˙group assms(1) unfolding group0˙def by auto
then have P:P⟨h,GroupInv(G,P)g⟩∈G using assms(1) group0.group˙op˙closed unfolding group0˙def by auto
with ‹g∈G› have P⟨g,P⟨h,GroupInv(G,P)g⟩ ⟩∈G using assms(1) group0.group˙op˙closed unfolding group0˙def by auto
then have f(P⟨g,P⟨h,GroupInv(G,P)g⟩ ⟩)=F⟨fg,f(P⟨h,GroupInv(G,P)g⟩)⟩
using assms(3) unfolding Homomor˙def[OF assms(1,2)] using ‹g∈G› P by auto
also have . . .=F⟨fg,F(⟨fh,f(GroupInv(G,P)g)⟩)⟩ using assms(3) unfolding Homomor˙def[OF assms(1,2)]
using ‹h∈G› inv by auto
also have . . .=F⟨fg,F(⟨fh,GroupInv(H,F)(fg)⟩)⟩ using image˙inv[OF assms ‹g∈G›] by auto
ultimately have f(P⟨g,P⟨h,GroupInv(G,P)g⟩ ⟩)=F⟨fg,F(⟨fh,GroupInv(H,F)(fg)⟩)⟩ by auto
moreover from AS(2) have fh=TheNeutralElement(H,F) using func1˙1˙L15[OF assms(4)]
by auto ultimately
have f(P⟨g,P⟨h,GroupInv(G,P)g⟩ ⟩)=F⟨fg,F(⟨TheNeutralElement(H,F),GroupInv(H,F)(fg)⟩)⟩

```
by auto
    also have ...=F⟨fg,GroupInv(H,F)(fg)⟩ using assms(2) im group0.group0˙2˙L2
unfolding group0˙def
      using iminv by auto
    also have ...=TheNeutralElement(H,F) using assms(2) group0.group0˙2˙L6 im
      unfolding group0˙def by auto
      ultimately have f(P⟨g,P⟨h,GroupInv(G,P)g⟩ ⟩)=TheNeutralElement(H,F) by
auto moreover
    from P ‹g∈G› have P⟨g,P⟨h,GroupInv(G,P)g⟩⟩∈G using group0.group˙op˙closed
assms(1) unfolding group0˙def by auto
    ultimately have P⟨g,P⟨h,GroupInv(G,P)g⟩ ⟩∈f-–TheNeutralElement(H,F)″ using
func1˙1˙L15[OF assms(4)]
      by auto
  }
 then have ∀g∈G. –P⟨g,P⟨h,GroupInv(G,P)g⟩ ⟩. h∈f-–TheNeutralElement(H,F)″″⊆f-–TheNeutralElement(H,F
    by auto
 then show thesis using group0.cont˙conj˙is˙normal assms(1) SS unfolding group0˙def
by auto
qed
```

The image of a homomorphism is a subgroup.

```
theorem image˙sub:
  assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F) f:G→H
  shows IsAsubgroup(fG,F)
proof-
  have TheNeutralElement(G,P)∈G using group0.group0˙2˙L2 assms(1) unfolding
group0˙def by auto
  then have TheNeutralElement(H,F)∈fG using func˙imagedef[OF assms(4),of G]
image˙neutral[OF assms]
    by force
  then have fG≠0 by auto moreover
  {
    fix h1 h2 assume h1∈fGh2∈fG
   then obtain g1 g2 where h1=fg1 h2=fg2 and p:g1∈Gg2∈G using func˙imagedef[OF
assms(4)] by auto
    then have F⟨h1,h2⟩=F⟨fg1,fg2⟩ by auto
   also have ...=f(P⟨g1,g2⟩) using assms(3) unfolding Homomor˙def[OF assms(1,2)]
using p by auto
    ultimately have F⟨h1,h2⟩=f(P⟨g1,g2⟩) by auto
     moreover have P⟨g1,g2⟩∈G using p group0.group˙op˙closed assms(1) unfolding
group0˙def
      by auto ultimately
    have F⟨h1,h2⟩∈fG using func˙imagedef[OF assms(4)] by auto
  }
  then have fG –is closed under″ F unfolding IsOpClosed˙def by auto
  moreover have fG⊆H using func1˙1˙L6(2) assms(4) by auto moreover
  {
    fix h assume h∈fG
   then obtain g where h=fg and p:g∈G using func˙imagedef[OF assms(4)] by auto
```

then have GroupInv(H,F)h=GroupInv(H,F)(fg) by auto
then have GroupInv(H,F)h=f(GroupInv(G,P)g) using p image˙inv[OF assms] by auto
then have GroupInv(H,F)h∈fG using p group0.inverse˙in˙group assms(1) unfolding group0˙def
using func˙imagedef[OF assms(4)] by auto
}
then have ∀h∈fG. GroupInv(H,F)h∈fG by auto ultimately
show thesis using group0.group0˙3˙T3 assms(2) unfolding group0˙def by auto
qed

Now we are able to prove the first isomorphism theorem. This theorem states that any group homomorphism $f : G \to H$ gives an isomorphism between a quotient group of G and a subgroup of H.

theorem isomorphism˙first˙theorem:
assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F) f:G→H
defines r ≡ QuotientGroupRel(G,P,f-–TheNeutralElement(H,F)″) and
PP ≡ QuotientGroupOp(G,P,f-–TheNeutralElement(H,F)″)
shows ∃ff. Homomor(ff,G//r,PP,fG,restrict(F,(fG)×(fG))) ∧ ff∈bij(G//r,fG)
proof-
let ff=–⟨r–g″,fg⟩. g∈G″
{
fix t assume t∈–⟨r–g″,fg⟩. g∈G″
then obtain g where t=⟨r–g″,fg⟩ g∈G by auto
moreover then have r–g″∈G//r unfolding r˙def quotient˙def by auto
moreover from ‹g∈G› have fg∈fG using func˙imagedef[OF assms(4)] by auto
ultimately have t∈(G//r)×fG by auto
}
then have ff∈Pow((G//r)×fG) by auto
moreover have (G//r)⊆domain(ff) unfolding domain˙def quotient˙def by auto moreover
{
fix x y t assume A:⟨x,y⟩∈ff ⟨x,t⟩∈ff
then obtain gy gr where ⟨x, y⟩=⟨r–gy″,fgy⟩ ⟨x, t⟩=⟨r–gr″,fgr⟩ and p:gr∈Ggy∈G by auto
then have B:r–gy″=r–gr″y=fgyt=fgr by auto
from B(2,3) have q:y∈Ht∈H using apply˙type p assms(4) by auto
have ⟨gy,gr⟩∈r using eq˙equiv˙class[OF B(1) ˙ p(1)] group0.Group˙ZF˙2˙4˙L3 kerner˙normal˙sub[OF assms(1-4)]
assms(1) unfolding group0˙def IsAnormalSubgroup˙def r˙def by auto
then have P⟨gy,GroupInv(G,P)gr⟩∈f-–TheNeutralElement(H,F)″ unfolding r˙def QuotientGroupRel˙def by auto
then have eq:f(P⟨gy,GroupInv(G,P)gr⟩)=TheNeutralElement(H,F) using func1˙1˙L15[OF assms(4)] by auto
from B(2,3) have F⟨y,GroupInv(H,F)t⟩=F⟨fgy,GroupInv(H,F)(fgr)⟩ by auto
also have ...=F⟨fgy,f(GroupInv(G,P)gr)⟩ using image˙inv[OF assms(1-4)] p(1) by auto
also have ...=f(P⟨gy,GroupInv(G,P)gr⟩) using assms(3) unfolding Homomor˙def[OF assms(1,2)] using p(2)

group0.inverse˙in˙group assms(1) p(1) unfolding group0˙def by auto
ultimately have F⟨y,GroupInv(H,F)t⟩=TheNeutralElement(H,F) using eq by auto
then have y=t using assms(2) group0.group0˙2˙L11A q unfolding group0˙def by auto
}
then have ∀x y. ⟨x,y⟩∈ff ⟶ (∀y'. ⟨x,y'⟩∈ff ⟶ y=y') by auto
ultimately have ff˙fun:ff:G//r→fG unfolding Pi˙def function˙def by auto
{
fix a1 a2 assume AS:a1∈G//ra2∈G//r
then obtain g1 g2 where p:g1∈Gg2∈G and a:a1=r–g1″a2=r–g2″ unfolding quotient˙def by auto
have equiv(G,r) using group0.Group˙ZF˙2˙4˙L3 kerner˙normal˙sub[OF assms(1-4)] assms(1) unfolding group0˙def IsAnormalSubgroup˙def r˙def by auto moreover
have Congruent2(r,P) using Group˙ZF˙2˙4˙L5A[OF assms(1) kerner˙normal˙sub[OF assms(1-4)]] unfolding r˙def by auto moreover
have PP=ProjFun2(G,r,P) unfolding PP˙def QuotientGroupOp˙def r˙def by auto moreover
note a p ultimately have PP⟨a1,a2⟩=r–P⟨g1,g2⟩″ using group0.Group˙ZF˙2˙2˙L2 assms(1) unfolding group0˙def by auto
then have ⟨PP⟨a1,a2⟩,f(P⟨g1,g2⟩)⟩∈ff using group0.group˙op˙closed[OF ˙ p] assms(1) unfolding group0˙def by auto
then have eq:ff(PP⟨a1,a2⟩)=f(P⟨g1,g2⟩) using apply˙equality ff˙fun by auto
from p a have ⟨a1,fg1⟩∈ff⟨a2,fg2⟩∈ff by auto
then have ffa1=fg1ffa2=fg2 using apply˙equality ff˙fun by auto
then have F⟨ffa1,ffa2⟩=F⟨fg1,fg2⟩ by auto
also have ...=f(P⟨g1,g2⟩) using assms(3) unfolding Homomor˙def[OF assms(1,2)] using p by auto
ultimately have F⟨ffa1,ffa2⟩=ff(PP⟨a1,a2⟩) using eq by auto moreover
have ffa1∈fGffa2∈fG using ff˙fun apply˙type AS by auto ultimately
have restrict(F,fG×fG)⟨ffa1,ffa2⟩=ff(PP⟨a1,a2⟩) by auto
}
then have r:∀a1∈G//r. ∀a2∈G//r. restrict(F,fG×fG)⟨ffa1,ffa2⟩=ff(PP⟨a1,a2⟩) by auto
have G:IsAgroup(G//r,PP) using Group˙ZF˙2˙4˙T1[OF assms(1) kerner˙normal˙sub[OF assms(1-4)]] unfolding r˙def PP˙def by auto
have H:IsAgroup(fG, restrict(F,fG×fG)) using image˙sub[OF assms(1-4)] unfolding IsAsubgroup˙def .
have HOM:Homomor(ff,G//r,PP,fG,restrict(F,(fG)×(fG))) using r unfolding Homomor˙def[OF G H] by auto
{
fix b1 b2 assume AS:ffb1=ffb2b1∈G//rb2∈G//r
have invb2:GroupInv(G//r,PP)b2∈G//r using group0.inverse˙in˙group[OF ˙ AS(3)] G unfolding group0˙def by auto
with AS(2) have PP⟨b1,GroupInv(G//r,PP)b2⟩∈G//r using group0.group˙op˙closed

```
G unfolding group0˙def by auto moreover
    then obtain gg where gg:gg∈GPP⟨b1,GroupInv(G//r,PP)b2⟩=r–gg″ unfolding
quotient˙def by auto
  ultimately have E:ff(PP⟨b1,GroupInv(G//r,PP)b2⟩)=fgg using apply˙equality[OF
˙ ff˙fun] by auto
  from invb2 have pp:ff(GroupInv(G//r,PP)b2)∈fG using apply˙type ff˙fun by auto
   from AS(2,3) have fff:ffb1∈fGffb2∈fG using apply˙type[OF ff˙fun] by auto
  from fff(1) pp have EE:F⟨ffb1,ff(GroupInv(G//r,PP)b2)⟩=restrict(F,fG×fG)⟨ffb1,ff(GroupInv(G//r,PP)b2)⟩
     by auto
   from fff have fff2:ffb1∈Hffb2∈H using func1˙1˙L6(2)[OF assms(4)] by auto
    with AS(1) have TheNeutralElement(H,F)=F⟨ffb1,GroupInv(H,F)(ffb2)⟩ using
group0.group0˙2˙L6(1)
     assms(2) unfolding group0˙def by auto
    also have ...=F⟨ffb1,restrict(GroupInv(H,F),fG)(ffb2)⟩ using restrict fff(2) by
auto
   also have ...=F⟨ffb1,ff(GroupInv(G//r,PP)b2)⟩ using image˙inv[OF G H HOM
ff˙fun AS(3)]
   group0.group0˙3˙T1[OF ˙ image˙sub[OF assms(1-4)]] assms(2) unfolding group0˙def
by auto
    also have ...=restrict(F,fG×fG)⟨ffb1,ff(GroupInv(G//r,PP)b2)⟩ using EE by
auto
    also have ...=ff(PP⟨b1,GroupInv(G//r,PP)b2⟩) using HOM unfolding Homo-
mor˙def[OF G H] using AS(2)
     group0.inverse˙in˙group[OF ˙ AS(3)] G unfolding group0˙def by auto
   also have ...=fgg using E by auto
   ultimately have fgg=TheNeutralElement(H,F) by auto
     then have gg∈f-–TheNeutralElement(H,F)″ using func1˙1˙L15[OF assms(4)]
‹gg∈G› by auto
  then have r–gg″=TheNeutralElement(G//r,PP) using group0.Group˙ZF˙2˙4˙L5E[OF
˙ kerner˙normal˙sub[OF assms(1-4)]
     ‹gg∈G› ] using assms(1) unfolding group0˙def r˙def PP˙def by auto
    with gg(2) have PP⟨b1,GroupInv(G//r,PP)b2⟩=TheNeutralElement(G//r,PP)
by auto
  then have b1=b2 using group0.group0˙2˙L11A[OF ˙ AS(2,3)] G unfolding group0˙def
by auto
 }
 then have ff∈inj(G//r,fG) unfolding inj˙def using ff˙fun by auto moreover
 {
   fix m assume m∈fG
   then obtain g where g∈Gm=fg using func˙imagedef[OF assms(4)] by auto
   then have ⟨r–g″,m⟩∈ff by auto
   then have ff(r–g″)=m using apply˙equality ff˙fun by auto
   then have ∃A∈G//r. ffA=m unfolding quotient˙def using ‹g∈G› by auto
 }
 ultimately have ff∈bij(G//r,fG) unfolding bij˙def surj˙def using ff˙fun by auto
 with HOM show thesis by auto
qed
```

As a last result, the inverse of a bijective homomorphism is an homomor-

phism. Meaning that in the previous result, the homomorphism we found is an isomorphism.

```
theorem bij˙homomor:
  assumes f∈bij(G,H)IsAgroup(G,P)IsAgroup(H,F)Homomor(f,G,P,H,F)
  shows Homomor(converse(f),H,F,G,P)
proof-
  {
    fix h1 h2 assume A:h1∈H h2∈H
     from A(1) obtain g1 where g1:g1∈G fg1=h1 using assms(1) unfolding bij˙def
surj˙def by auto moreover
     from A(2) obtain g2 where g2:g2∈G fg2=h2 using assms(1) unfolding bij˙def
surj˙def by auto ultimately
    have F⟨fg1,fg2⟩=F⟨h1,h2⟩ by auto
    then have f(P⟨g1,g2⟩)=F⟨h1,h2⟩ using assms(2,3,4) homomor˙eq g1(1) g2(1) by
auto
    then have converse(f)(f(P⟨g1,g2⟩))=converse(f)(F⟨h1,h2⟩) by auto
   then have P⟨g1,g2⟩=converse(f)(F⟨h1,h2⟩) using left˙inverse assms(1) group0.group˙op˙closed
      assms(2) g1(1) g2(1) unfolding group0˙def bij˙def by auto moreover
    from g1(2) have converse(f)(fg1)=converse(f)h1 by auto
     then have g1=converse(f)h1 using left˙inverse assms(1) unfolding bij˙def using
g1(1) by auto moreover
    from g2(2) have converse(f)(fg2)=converse(f)h2 by auto
     then have g2=converse(f)h2 using left˙inverse assms(1) unfolding bij˙def using
g2(1) by auto ultimately
    have P⟨converse(f)h1,converse(f)h2⟩=converse(f)(F⟨h1,h2⟩) by auto
  }
  then show thesis using assms(2,3) Homomor˙def by auto
qed

end
```

40 Fields - introduction

```
theory Field˙ZF imports Ring˙ZF

begin
```

This theory covers basic facts about fields.

40.1 Definition and basic properties

In this section we define what is a field and list the basic properties of fields.

Field is a notrivial commutative ring such that all non-zero elements have an inverse. We define the notion of being a field as a statement about three sets. The first set, denoted K is the carrier of the field. The second set, denoted A represents the additive operation on K (recall that in ZF set theory functions are sets). The third set M represents the multiplicative operation on K.

definition
IsAfield(K,A,M) ≡
(IsAring(K,A,M) ∧ (M –is commutative on″ K) ∧
TheNeutralElement(K,A) ≠ TheNeutralElement(K,M) ∧
(∀ a∈K. a≠TheNeutralElement(K,A)⟶
(∃ b∈K. M⟨a,b⟩ = TheNeutralElement(K,M))))

The field0 context extends the ring0 context adding field-related assumptions and notation related to the multiplicative inverse.

locale field0 = ring0 K A M for K A M +
assumes mult˙commute: M –is commutative on″ K

assumes not˙triv: **0** ≠ **1**

assumes inv˙exists: ∀ a∈K. a≠**0** ⟶ (∃ b∈K. a·b = **1**)

fixes non˙zero (K_0)
defines non˙zero˙def[simp]: K_0 ≡ K-–**0**″

fixes inv (˙$^{-1}$ [96] 97)
defines inv˙def[simp]: a^{-1} ≡ GroupInv(K_0,restrict(M,K_0×K_0))(a)

The next lemma assures us that we are talking fields in the field0 context.

lemma (in field0) Field˙ZF˙1˙L1: shows IsAfield(K,A,M)
using ringAssum mult˙commute not˙triv inv˙exists IsAfield˙def
by simp

We can use theorems proven in the field0 context whenever we talk about a field.

lemma field˙field0: assumes IsAfield(K,A,M)
shows field0(K,A,M)
using assms IsAfield˙def field0˙axioms.intro ring0˙def field0˙def
by simp

Let's have an explicit statement that the multiplication in fields is commutative.

lemma (in field0) field˙mult˙comm: assumes a∈K b∈K
shows a·b = b·a
using mult˙commute assms IsCommutative˙def by simp

Fields do not have zero divisors.

lemma (in field0) field˙has˙no˙zero˙divs: shows HasNoZeroDivs(K,A,M)
proof -
{ fix a b assume A1: a∈K b∈K and A2: a·b = **0** and A3: b≠**0**
from inv˙exists A1 A3 obtain c where I: c∈K and II: b·c = **1**
by auto
from A2 have a·b·c = **0**·c by simp
with A1 I have a·(b·c) = **0**

```
    using Ring˙ZF˙1˙L11 Ring˙ZF˙1˙L6 by simp
  with A1 II have a=0 using Ring˙ZF˙1˙L3 by simp }
 then have ∀a∈K.∀b∈K. a·b = 0 ⟶ a=0 ∨ b=0 by auto
  then show thesis using HasNoZeroDivs˙def by auto
qed
```

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

```
lemma (in field0) Field˙ZF˙1˙L2:
 shows K0 –is closed under″ M
 using Ring˙ZF˙1˙L4 field˙has˙no˙zero˙divs Ring˙ZF˙1˙L12
  IsOpClosed˙def by auto
```

Any nonzero element has a right inverse that is nonzero.

```
lemma (in field0) Field˙ZF˙1˙L3: assumes A1: a∈K0
 shows ∃b∈K0. a·b = 1
proof -
 from inv˙exists A1 obtain b where b∈K and a·b = 1
  by auto
 with not˙triv A1 show ∃b∈K0. a·b = 1
  using Ring˙ZF˙1˙L6 by auto
qed
```

If we remove zero, the field with multiplication becomes a group and we can use all theorems proven in group0 context.

```
theorem (in field0) Field˙ZF˙1˙L4: shows
 IsAgroup(K0,restrict(M,K0×K0))
 group0(K0,restrict(M,K0×K0))
 1 = TheNeutralElement(K0,restrict(M,K0×K0))
proof-
 let f = restrict(M,K0×K0)
 have
  M –is associative on″ K
  K0 ⊆ K  K0 –is closed under″ M
  using Field˙ZF˙1˙L1 IsAfield˙def IsAring˙def IsAgroup˙def
   IsAmonoid˙def Field˙ZF˙1˙L2 by auto
 then have f –is associative on″ K0
  using func˙ZF˙4˙L3 by simp
 moreover
 from not˙triv have
  I: 1∈K0 ∧ (∀a∈K0. f⟨1,a⟩ = a ∧  f⟨a,1⟩ = a)
  using Ring˙ZF˙1˙L2 Ring˙ZF˙1˙L3 by auto
 then have ∃n∈K0. ∀a∈K0. f⟨n,a⟩ = a ∧  f⟨a,n⟩ = a
  by blast
 ultimately have II: IsAmonoid(K0,f) using IsAmonoid˙def
  by simp
 then have monoid0(K0,f) using monoid0˙def by simp
 moreover note I
 ultimately show 1 = TheNeutralElement(K0,f)
  by (rule monoid0.group0˙1˙L4)
```

```
  then have ∀a∈K0.∃b∈K0. f⟨a,b⟩ =  TheNeutralElement(K0,f)
    using Field˙ZF˙1˙L3 by auto
  with II show IsAgroup(K0,f) by (rule definition˙of˙group)
  then show group0(K0,f) using group0˙def by simp
qed
```

The inverse of a nonzero field element is nonzero.

```
lemma (in field0) Field˙ZF˙1˙L5: assumes A1: a∈K  a≠0
  shows a^-1 ∈ K0  (a^-1)^2 ∈ K0   a^-1 ∈ K  a^-1 ≠ 0
proof -
  from A1 have a ∈ K0 by simp
  then show a^-1 ∈ K0 using Field˙ZF˙1˙L4 group0.inverse˙in˙group
    by auto
  then show  (a^-1)^2 ∈ K0  a^-1 ∈ K  a^-1 ≠ 0
    using Field˙ZF˙1˙L2 IsOpClosed˙def by auto
qed
```

The inverse is really the inverse.

```
lemma (in field0) Field˙ZF˙1˙L6: assumes A1: a∈K  a≠0
  shows a·a^-1 = 1  a^-1·a = 1
proof -
  let f = restrict(M,K0×K0)
  from A1 have
    group0(K0,f)
    a ∈ K0
    using Field˙ZF˙1˙L4 by auto
  then have
    f⟨a,GroupInv(K0, f)(a)⟩ = TheNeutralElement(K0,f) ∧
    f⟨GroupInv(K0,f)(a),a⟩ = TheNeutralElement(K0, f)
    by (rule group0.group0˙2˙L6)
  with A1 show a·a^-1 = 1  a^-1·a = 1
    using Field˙ZF˙1˙L5 Field˙ZF˙1˙L4 by auto
qed
```

A lemma with two field elements and cancelling.

```
lemma (in field0) Field˙ZF˙1˙L7: assumes a∈K b∈K b≠0
  shows
  a·b·b^-1 = a
  a·b^-1·b = a
  using assms Field˙ZF˙1˙L5 Ring˙ZF˙1˙L11 Field˙ZF˙1˙L6 Ring˙ZF˙1˙L3
  by auto
```

40.2 Equations and identities

This section deals with more specialized identities that are true in fields.

$a/(a^2) = 1/a$.

```
lemma (in field0) Field˙ZF˙2˙L1: assumes A1: a∈K  a≠0
```

```
  shows a·(a⁻¹)² = a⁻¹
proof -
  have a·(a⁻¹)² = a·(a⁻¹·a⁻¹) by simp
  also from A1 have ... =  (a·a⁻¹)·a⁻¹
    using Field˙ZF˙1˙L5 Ring˙ZF˙1˙L11
    by simp
  also from A1 have ... = a⁻¹
    using Field˙ZF˙1˙L6 Field˙ZF˙1˙L5 Ring˙ZF˙1˙L3
    by simp
  finally show a·(a⁻¹)² = a⁻¹ by simp
qed
```

If we multiply two different numbers by a nonzero number, the results will be different.

```
lemma (in field0) Field˙ZF˙2˙L2:
  assumes a∈K  b∈K  c∈K  a≠b  c≠0
  shows a·c⁻¹ ≠ b·c⁻¹
  using assms field˙has˙no˙zero˙divs Field˙ZF˙1˙L5 Ring˙ZF˙1˙L12B
  by simp
```

We can put a nonzero factor on the other side of non-identity (is this the best way to call it?) changing it to the inverse.

```
lemma (in field0) Field˙ZF˙2˙L3:
  assumes A1: a∈K  b∈K  b≠0  c∈K   and A2: a·b ≠ c
  shows a ≠ c·b⁻¹
proof -
  from A1 A2 have a·b·b⁻¹ ≠ c·b⁻¹
    using  Ring˙ZF˙1˙L4 Field˙ZF˙2˙L2 by simp
  with A1 show a ≠ c·b⁻¹ using Field˙ZF˙1˙L7
    by simp
qed
```

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

```
lemma (in field0) Field˙ZF˙2˙L4:
  assumes a∈K  a≠0 and b⁻¹ ≠ a
  shows a⁻¹ ≠ b
  using assms Field˙ZF˙1˙L4 group0.group0˙2˙L11B
  by simp
```

An identity with two field elements, one and an inverse.

```
lemma (in field0) Field˙ZF˙2˙L5:
  assumes a∈K  b∈K b≠0
  shows (1 + a·b)·b⁻¹ = a + b⁻¹
  using assms Ring˙ZF˙1˙L4 Field˙ZF˙1˙L5 Ring˙ZF˙1˙L2 ring˙oper˙distr
    Field˙ZF˙1˙L7 Ring˙ZF˙1˙L3 by simp
```

An identity with three field elements, inverse and cancelling.

```
lemma (in field0) Field˙ZF˙2˙L6: assumes A1: a∈K  b∈K  b≠0  c∈K
  shows a·b·(c·b⁻¹) = a·c
proof -
  from A1 have T: a·b ∈ K  b⁻¹ ∈ K
    using Ring˙ZF˙1˙L4 Field˙ZF˙1˙L5 by auto
  with mult˙commute A1 have a·b·(c·b⁻¹) = a·b·(b⁻¹·c)
    using IsCommutative˙def by simp
  moreover
  from A1 T have a·b ∈ K  b⁻¹ ∈ K  c∈K
    by auto
  then have a·b·b⁻¹·c = a·b·(b⁻¹·c)
    by (rule Ring˙ZF˙1˙L11)
  ultimately have a·b·(c·b⁻¹) = a·b·b⁻¹·c by simp
  with A1 show a·b·(c·b⁻¹) = a·c
    using Field˙ZF˙1˙L7 by simp
qed
```

40.3 1/0=0

In ZF if $f : X \to Y$ and $x \notin X$ we have $f(x) = \emptyset$. Since $\emptyset$ (the empty set) in ZF is the same as zero of natural numbers we can claim that $1/0 = 0$ in certain sense. In this section we prove a theorem that makes makes it explicit.

The next locale extends the field0 locale to introduce notation for division operation.

```
locale fieldd = field0 +
  fixes division
  defines division˙def[simp]: division ≡ −⟨p,fst(p)·snd(p)⁻¹⟩. p∈K×K₀″

  fixes fdiv (infixl / 95)
  defines fdiv˙def[simp]: x/y ≡ division⟨x,y⟩
```

Division is a function on $K \times K_0$ with values in K.

```
lemma (in fieldd) div˙fun: shows division: K×K₀ → K
proof -
  have ∀p ∈ K×K₀. fst(p)·snd(p)⁻¹ ∈ K
  proof
    fix p assume p ∈ K×K₀
    hence fst(p) ∈ K and snd(p) ∈ K₀ by auto
    then show  fst(p)·snd(p)⁻¹ ∈ K using Ring˙ZF˙1˙L4 Field˙ZF˙1˙L5 by auto
  qed
  then have −⟨p,fst(p)·snd(p)⁻¹⟩. p∈K×K₀″: K×K₀ → K
    by (rule ZF˙fun˙from˙total)
  thus thesis by simp
qed
```

So, really $1/0 = 0$. The essential lemma is apply˙0 from standard Isabelle's func.thy.

theorem (in fieldd) one˙over˙zero: shows $\mathbf{1}/\mathbf{0} = 0$
proof-
  have domain(division) = K×K_0 using div˙fun func1˙1˙L1
    by simp
  hence ⟨**1**,**0**⟩ ∉ domain(division) by auto
  then show thesis using apply˙0 by simp
qed

end

41 Ordered fields

theory OrderedField˙ZF imports OrderedRing˙ZF Field˙ZF

begin

This theory covers basic facts about ordered fiels.

41.1 Definition and basic properties

Here we define ordered fields and proove their basic properties.

Ordered field is a notrivial ordered ring such that all non-zero elements have an inverse. We define the notion of being a ordered field as a statement about four sets. The first set, denoted K is the carrier of the field. The second set, denoted A represents the additive operation on K (recall that in ZF set theory functions are sets). The third set M represents the multiplicative operation on K. The fourth set r is the order relation on K.

definition
  IsAnOrdField(K,A,M,r) ≡ (IsAnOrdRing(K,A,M,r) ∧
  (M –is commutative on″ K) ∧
  TheNeutralElement(K,A) ≠ TheNeutralElement(K,M) ∧
  (∀ a∈K. a≠TheNeutralElement(K,A)⟶
  (∃ b∈K. M⟨a,b⟩ = TheNeutralElement(K,M))))

The next context (locale) defines notation used for ordered fields. We do that by extending the notation defined in the ring1 context that is used for 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: $\mathbf{0} \neq \mathbf{1}$

assumes inv˙exists: $\forall$ a∈R. a≠$\mathbf{0}$ $\longrightarrow$ ($\exists$ b∈R. a·b = $\mathbf{1}$)

fixes non˙zero (R_0)
defines non˙zero˙def[simp]: R_0 ≡ R-–$\mathbf{0}$″

fixes inv (˙$^{-1}$ [96] 97)
defines inv˙def[simp]: a^{-1} ≡ GroupInv(R_0,restrict(M,R_0×R_0))(a)

The next lemma assures us that we are talking fields in the field1 context.

lemma (in field1) OrdField˙ZF˙1˙L1: shows IsAnOrdField(R,A,M,r)
using OrdRing˙ZF˙1˙L1 mult˙commute not˙triv inv˙exists IsAnOrdField˙def
by simp

Ordered field is a field, of course.

lemma OrdField˙ZF˙1˙L1A: assumes IsAnOrdField(K,A,M,r)
shows IsAfield(K,A,M)
using assms IsAnOrdField˙def IsAnOrdRing˙def IsAfield˙def
by simp

Theorems proven in field0 (about fields) context are valid in the field1 context (about ordered fields).

lemma (in field1) OrdField˙ZF˙1˙L1B: shows field0(R,A,M)
using OrdField˙ZF˙1˙L1 OrdField˙ZF˙1˙L1A field˙field0
by simp

We can use theorems proven in the field1 context whenever we talk about an ordered field.

lemma OrdField˙ZF˙1˙L2: assumes IsAnOrdField(K,A,M,r)
shows field1(K,A,M,r)
using assms IsAnOrdField˙def OrdRing˙ZF˙1˙L2 ring1˙def
IsAnOrdField˙def field1˙axioms˙def field1˙def
by auto

In ordered rings the existence of a right inverse for all positive elements implies the existence of an inverse for all non zero elements.

lemma (in ring1) OrdField˙ZF˙1˙L3:
assumes A1: $\forall$ a∈R_+. $\exists$ b∈R. a·b = $\mathbf{1}$ and A2: c∈R c≠$\mathbf{0}$
shows $\exists$ b∈R. c·b = $\mathbf{1}$
proof -
{ assume c∈R_+
with A1 have $\exists$ b∈R. c·b = $\mathbf{1}$ by simp }
moreover
{ assume c∉R_+
with A2 have (-c) ∈ R_+
using OrdRing˙ZF˙3˙L2A by simp
with A1 obtain b where b∈R and (-c)·b = $\mathbf{1}$

```
    by auto
  with A2 have (-b) ∈ R  c·(-b) = 1
    using Ring˙ZF˙1˙L3 Ring˙ZF˙1˙L7 by auto
  then have ∃b∈R. c·b = 1 by auto }
 ultimately show thesis by blast
qed
```

Ordered fields are easier to deal with, because it is sufficient to show the existence of an inverse for the set of positive elements.

```
lemma (in ring1) OrdField˙ZF˙1˙L4:
 assumes 0 ≠ 1 and M –is commutative on˝ R
 and ∀a∈R+. ∃b∈R. a·b = 1
 shows IsAnOrdField(R,A,M,r)
 using assms OrdRing˙ZF˙1˙L1 OrdField˙ZF˙1˙L3 IsAnOrdField˙def
 by simp
```

The set of positive field elements is closed under multiplication.

```
lemma (in field1) OrdField˙ZF˙1˙L5: shows R+ –is closed under˝ M
 using OrdField˙ZF˙1˙L1B field0.field˙has˙no˙zero˙divs OrdRing˙ZF˙3˙L3
 by simp
```

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

```
lemma (in field1) pos˙mul˙closed:
 assumes A1: 0 ¡ a  0 ¡ b
 shows 0 ¡ a·b
proof -
 from A1 have a ∈ R+ and  b ∈ R+
   using OrdRing˙ZF˙3˙L14 by auto
 then show 0 ¡ a·b
   using OrdField˙ZF˙1˙L5 IsOpClosed˙def PositiveSet˙def
   by simp
qed
```

In fields square of a nonzero element is positive.

```
lemma (in field1) OrdField˙ZF˙1˙L6: assumes a∈R  a≠0
 shows a^2 ∈ R+
 using assms OrdField˙ZF˙1˙L1B field0.field˙has˙no˙zero˙divs
   OrdRing˙ZF˙3˙L15 by simp
```

The next lemma restates the fact Field˙ZF that out notation for the field inverse means what it is supposed to mean.

```
lemma (in field1) OrdField˙ZF˙1˙L7: assumes a∈R  a≠0
 shows a·(a^-1) = 1  (a^-1)·a = 1
 using assms OrdField˙ZF˙1˙L1B field0.Field˙ZF˙1˙L6
 by auto
```

A simple lemma about multiplication and cancelling of a positive field element.

```
lemma (in field1) OrdField˙ZF˙1˙L7A:
  assumes A1: a∈R  b ∈ R+
  shows
  a·b·b^-1 = a
  a·b^-1·b = a
proof -
  from A1 have b∈R  b≠0 using PositiveSet˙def
    by auto
  with A1 show  a·b·b^-1 = a and a·b^-1·b = a
    using OrdField˙ZF˙1˙L1B field0.Field˙ZF˙1˙L7
    by auto
qed
```

Some properties of the inverse of a positive element.

```
lemma (in field1) OrdField˙ZF˙1˙L8: assumes A1: a ∈ R+
  shows a^-1 ∈ R+  a·(a^-1) = 1  (a^-1)·a = 1
proof -
  from A1 have I: a∈R  a≠0 using PositiveSet˙def
    by auto
  with A1 have a·(a^-1)^2 ∈ R+
    using OrdField˙ZF˙1˙L1B field0.Field˙ZF˙1˙L5 OrdField˙ZF˙1˙L6
      OrdField˙ZF˙1˙L5 IsOpClosed˙def by simp
  with I show a^-1 ∈ R+
    using OrdField˙ZF˙1˙L1B field0.Field˙ZF˙2˙L1
    by simp
  from I show  a·(a^-1) = 1  (a^-1)·a = 1
    using OrdField˙ZF˙1˙L7 by auto
qed
```

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 assms OrdRing˙ZF˙1˙L14 OrdField˙ZF˙1˙L8
  by simp
```

In ordered fields if at least one of a, b is not zero, then $a^2 + b^2 > 0$, in particular $a^2 + b^2 \neq 0$ and 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 ¡ a^2 + b^2  and ∃c∈R. (a^2 + b^2)·c = 1
proof -
  from A1 A2 show 0 ¡ a^2 + b^2
    using OrdField˙ZF˙1˙L1B field0.field˙has˙no˙zero˙divs
      OrdRing˙ZF˙3˙L19 by simp
  then have
    (a^2 + b^2)^-1 ∈ R and (a^2 + b^2)·(a^2 + b^2)^-1 = 1
    using OrdRing˙ZF˙1˙L3 PositiveSet˙def OrdField˙ZF˙1˙L8
    by auto
  then show ∃c∈R. (a^2 + b^2)·c = 1 by auto
```

qed

41.2 Inequalities

In this section we develop tools to deal inequalities in fields.

We can multiply strict inequality by a positive element.

lemma (in field1) OrdField˙ZF˙2˙L1:
  assumes a¡b and c∈R_+
  shows a·c ¡ b·c
  using assms OrdField˙ZF˙1˙L1B field0.field˙has˙no˙zero˙divs
    OrdRing˙ZF˙3˙L13
  by simp

A special case of OrdField˙ZF˙2˙L1 when we multiply an inverse by an element.

lemma (in field1) OrdField˙ZF˙2˙L2:
  assumes A1: a∈R_+ and A2: a^{-1} ¡ b
  shows **1** ¡ b·a
proof -
  from A1 A2 have (a^{-1})·a ¡ b·a
    using OrdField˙ZF˙2˙L1 by simp
  with A1 show **1** ¡ b·a
    using OrdField˙ZF˙1˙L8 by simp
qed

We can multiply an inequality by the inverse of a positive element.

lemma (in field1) OrdField˙ZF˙2˙L3:
  assumes a≤b and c∈R_+ shows a·(c^{-1}) ≤ b·(c^{-1})
  using assms OrdField˙ZF˙1˙L8 OrdRing˙ZF˙1˙L9A
  by simp

We can multiply a strict inequality by a positive element or its inverse.

lemma (in field1) OrdField˙ZF˙2˙L4:
  assumes a¡b and c∈R_+
  shows
  a·c ¡ b·c
  c·a ¡ c·b
  a·c^{-1} ¡ b·c^{-1}
   using assms OrdField˙ZF˙1˙L1B field0.field˙has˙no˙zero˙divs
    OrdField˙ZF˙1˙L8 OrdRing˙ZF˙3˙L13 by auto

We can put a positive factor on the other side of an inequality, changing it to its inverse.

lemma (in field1) OrdField˙ZF˙2˙L5:
  assumes A1: a∈R b∈R_+ and A2: a·b ≤ c
  shows a ≤ c·b^{-1}
proof -
  from A1 A2 have a·b·b^{-1} ≤ c·b^{-1}

using OrdField˙ZF˙2˙L3 by simp
with A1 show a $\leq$ 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 $\leq$ b·c
shows a·c^{-1} $\leq$ b
proof -
from A1 A2 have a·c^{-1} $\leq$ b·c·c^{-1}
using OrdField˙ZF˙2˙L3 by simp
with A1 show a·c^{-1} $\leq$ 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} $\leq$ b
shows b^{-1} $\leq$ a
proof -
from A1 have a^{-1} $\in$ 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
```

41.3 Definition of real numbers

The only purpose of this section is to define what does it mean to be a model of real numbers.

We define model of real numbers as any quadruple of sets (K, A, M, r) such that (K, A, M, r) is an ordered field and the order relation r is complete, that is every set that is nonempty and bounded above in this relation has a supremum.

```
definition
  IsAmodelOfReals(K,A,M,r) ≡ IsAnOrdField(K,A,M,r) ∧ (r –is complete")

end
```

42 Integers - introduction

```
theory Int˙ZF˙IML imports OrderedGroup˙ZF˙1 Finite˙ZF˙1 ZF.Int Nat˙ZF˙IML

begin
```

This theory file is an interface between the old-style Isabelle (ZF logic) material on integers and the IsarMathLib project. Here we redefine the meta-level operations on integers (addition and multiplication) to convert them to ZF-functions and show that integers form a commutative group with respect to addition and commutative monoid with respect to multiplication. Similarly, we redefine the order on integers as a relation, that is a subset of $Z \times Z$. We show that a subset of intergers is bounded iff it is finite. As we are forced to use standard Isabelle notation with all these dollar signs, sharps etc. to denote "type coercions" (?) the notation is often ugly and difficult to read.

42.1 Addition and multiplication as ZF-functions.

In this section we provide definitions of addition and multiplication as subsets of $(Z \times Z) \times Z$. We use the (higher order) relation defined in the standard Int theory to define a subset of $Z \times Z$ that constitutes the ZF order relation corresponding to it. We define the set of positive integers using the notion of positive set from the OrderedGroup˙ZF theory.

Definition of addition of integers as a binary operation on int. Recall that in standard Isabelle/ZF int is the set of integers and the sum of integers is denoted by prependig + with a dollar sign.

```
definition
  IntegerAddition ≡ − ⟨ x,c⟩ ∈ (int×int)×int. fst(x) $+ snd(x) = c″
```

Definition of multiplication of integers as a binary operation on int. In standard Isabelle/ZF product of integers is denoted by prepending the dollar sign to *.

```
definition
  IntegerMultiplication ≡
    − ⟨ x,c⟩ ∈ (int×int)×int. fst(x) $* snd(x) = c″
```

Definition of natural order on integers as a relation on int. In the standard Isabelle/ZF the inequality relation on integers is denoted ≤ prepended with the dollar sign.

```
definition
  IntegerOrder ≡ −p ∈ int×int. fst(p) $≤ snd(p)″
```

This defines the set of positive integers.

```
definition
  PositiveIntegers ≡ PositiveSet(int,IntegerAddition,IntegerOrder)
```

IntegerAddition and IntegerMultiplication are functions on int × int.

```
lemma Int˙ZF˙1˙L1: shows
  IntegerAddition : int×int → int
  IntegerMultiplication : int×int → int
proof -
  have
    −⟨ x,c⟩ ∈ (int×int)×int. fst(x) $+ snd(x) = c″ ∈ int×int→int
    −⟨ x,c⟩ ∈ (int×int)×int. fst(x) $* snd(x) = c″ ∈ int×int→int
    using func1˙1˙L11A by auto
  then show IntegerAddition : int×int → int
    IntegerMultiplication : int×int → int
    using IntegerAddition˙def IntegerMultiplication˙def by auto
qed
```

The next context (locale) defines notation used for integers. We define **0** to denote the neutral element of addition, **1** as the unit of the multiplicative monoid. We introduce notation m≤n for integers and write m..n to denote the integer interval with endpoints in m and n. abs(m) means the absolute value of m. This is a function defined in OrderedGroup that assigns x to itself if x is positive and assigns the opposite of x if $x \leq 0$. Unforunately we cannot use the $|\cdot|$ notation as in the OrderedGroup theory as this notation has been hogged by the standard Isabelle's Int theory. The notation -A where A is a subset of integers means the set $\{-m : m \in A\}$. The symbol maxf(f,M)

denotes tha maximum of function f over the set A. We also introduce a similar notation for the minimum.

locale int0 =

fixes ints ($\mathbb{Z}$)
defines ints˙def [simp]: $\mathbb{Z} \equiv$ int

fixes ia (infixl + 69)
defines ia˙def [simp]: a+b $\equiv$ IntegerAddition⟨ a,b⟩

fixes iminus (- ˙ 72)
defines rminus˙def [simp]: -a $\equiv$ GroupInv($\mathbb{Z}$,IntegerAddition)(a)

fixes isub (infixl - 69)
defines isub˙def [simp]: a-b $\equiv$ a+ (- b)

fixes imult (infixl · 70)
defines imult˙def [simp]: a·b $\equiv$ IntegerMultiplication⟨ a,b⟩

fixes setneg (- ˙ 72)
defines setneg˙def [simp]: -A $\equiv$ GroupInv($\mathbb{Z}$,IntegerAddition)(A)

fixes izero (**0**)
defines izero˙def [simp]: **0** $\equiv$ TheNeutralElement($\mathbb{Z}$,IntegerAddition)

fixes ione (**1**)
defines ione˙def [simp]: **1** $\equiv$ TheNeutralElement($\mathbb{Z}$,IntegerMultiplication)

fixes itwo (**2**)
defines itwo˙def [simp]: **2** $\equiv$ **1**+**1**

fixes ithree (**3**)
defines ithree˙def [simp]: **3** $\equiv$ **2**+**1**

fixes nonnegative ($\mathbb{Z}^+$)
defines nonnegative˙def [simp]:
$\mathbb{Z}^+ \equiv$ Nonnegative($\mathbb{Z}$,IntegerAddition,IntegerOrder)

fixes positive ($\mathbb{Z}_+$)
defines positive˙def [simp]:
$\mathbb{Z}_+ \equiv$ PositiveSet($\mathbb{Z}$,IntegerAddition,IntegerOrder)

fixes abs
defines abs˙def [simp]:
abs(m) $\equiv$ AbsoluteValue($\mathbb{Z}$,IntegerAddition,IntegerOrder)(m)

fixes lesseq (infix $\leq$ 60)
defines lesseq˙def [simp]: m $\leq$ n $\equiv$ ⟨m,n⟩ $\in$ IntegerOrder

```
  fixes interval (infix .. 70)
  defines interval˙def [simp]: m..n ≡ Interval(IntegerOrder,m,n)

  fixes maxf
  defines maxf˙def [simp]: maxf(f,A) ≡ Maximum(IntegerOrder,f(A))

  fixes minf
  defines minf˙def [simp]: minf(f,A) ≡ Minimum(IntegerOrder,f(A))
```

IntegerAddition adds integers and IntegerMultiplication multiplies integers. This states that the ZF functions IntegerAddition and IntegerMultiplication give the same results as the higher-order equivalents defined in the standard Int theory.

```
lemma (in int0) Int˙ZF˙1˙L2: assumes A1: a ∈ ℤ  b ∈ ℤ
  shows
  a+b = a $+ b
  a·b = a $* b
proof -
  let x = ⟨ a,b⟩
  let c = a $+ b
  let d = a $* b
  from A1 have
    ⟨ x,c⟩ ∈ −⟨ x,c⟩ ∈ (ℤ×ℤ)×ℤ. fst(x) $+ snd(x) = c″
    ⟨ x,d⟩ ∈ −⟨ x,d⟩ ∈ (ℤ×ℤ)×ℤ. fst(x) $* snd(x) = d″
    by auto
  then show a+b = a $+ b  a·b = a $* b
    using IntegerAddition˙def IntegerMultiplication˙def
      Int˙ZF˙1˙L1 apply˙iff by auto
qed
```

Integer addition and multiplication are associative.

```
lemma (in int0) Int˙ZF˙1˙L3:
  assumes x∈ℤ  y∈ℤ  z∈ℤ
  shows x+y+z = x+(y+z)  x·y·z = x·(y·z)
  using assms Int˙ZF˙1˙L2 zadd˙assoc zmult˙assoc by auto
```

Integer addition and multiplication are commutative.

```
lemma (in int0) Int˙ZF˙1˙L4:
  assumes x∈ℤ  y∈ℤ
  shows x+y = y+x  x·y = y·x
  using assms Int˙ZF˙1˙L2 zadd˙commute zmult˙commute
  by auto
```

Zero is neutral for addition and one for multiplication.

```
lemma (in int0) Int˙ZF˙1˙L5: assumes A1:x∈ℤ
  shows ($# 0) + x = x ∧ x + ($# 0) = x
  ($# 1)·x = x ∧ x·($# 1) = x
proof -
```

```
  from A1 show ($# 0) + x = x ∧ x + ($# 0) = x
    using  Int˙ZF˙1˙L2 zadd˙int0 Int˙ZF˙1˙L4 by simp
  from A1 have ($# 1)·x = x
    using Int˙ZF˙1˙L2 zmult˙int1 by simp
  with A1 show ($# 1)·x = x ∧ x·($# 1) = x
    using Int˙ZF˙1˙L4 by simp
qed
```

Zero is neutral for addition and one for multiplication.

```
lemma (in int0) Int˙ZF˙1˙L6: shows ($# 0)∈ℤ ∧
  (∀x∈ℤ. ($# 0)+x = x ∧ x+($# 0) = x)
  ($# 1)∈ℤ ∧
  (∀x∈ℤ. ($# 1)·x = x ∧ x·($# 1) = x)
  using Int˙ZF˙1˙L5 by auto
```

Integers with addition and integers with multiplication form monoids.

```
theorem (in int0) Int˙ZF˙1˙T1: shows
  IsAmonoid(ℤ,IntegerAddition)
  IsAmonoid(ℤ,IntegerMultiplication)
proof -
   have
    ∃e∈ℤ. ∀x∈ℤ. e+x = x ∧ x+e = x
     ∃e∈ℤ. ∀x∈ℤ. e·x = x ∧ x·e = x
     using int0.Int˙ZF˙1˙L6 by auto
   then show IsAmonoid(ℤ,IntegerAddition)
     IsAmonoid(ℤ,IntegerMultiplication) using
     IsAmonoid˙def IsAssociative˙def Int˙ZF˙1˙L1 Int˙ZF˙1˙L3
     by auto
qed
```

Zero is the neutral element of the integers with addition and one is the neutral element of the integers with multiplication.

```
lemma (in int0) Int˙ZF˙1˙L8: shows ($# 0) = 0  ($# 1) = 1
proof -
  have monoid0(ℤ,IntegerAddition)
    using Int˙ZF˙1˙T1 monoid0˙def by simp
  moreover have
    ($# 0)∈ℤ ∧
    (∀x∈ℤ. IntegerAddition⟨$# 0,x⟩ = x ∧
    IntegerAddition⟨x ,$# 0⟩ = x)
    using Int˙ZF˙1˙L6 by auto
  ultimately have ($# 0) = TheNeutralElement(ℤ,IntegerAddition)
    by (rule monoid0.group0˙1˙L4)
  then show ($# 0) = 0 by simp
  have monoid0(int,IntegerMultiplication)
    using Int˙ZF˙1˙T1 monoid0˙def by simp
  moreover have ($# 1) ∈ int ∧
    (∀x∈int. IntegerMultiplication⟨$# 1, x⟩ = x ∧
    IntegerMultiplication⟨x ,$# 1⟩ = x)
```

```
    using Int˙ZF˙1˙L6 by auto
  ultimately have
    ($# 1) = TheNeutralElement(int,IntegerMultiplication)
    by (rule monoid0.group0˙1˙L4)
  then show  ($# 1) = 1 by simp
qed
```

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

```
lemma (in int0) Int˙ZF˙1˙L8A: shows 0 ∈ ℤ  1 ∈ ℤ
proof -
  have ($# 0) ∈ ℤ  ($# 1) ∈ ℤ by auto
  then show 0 ∈ ℤ  1 ∈ ℤ using Int˙ZF˙1˙L8 by auto
qed
```

Zero is not one.

```
lemma (in int0) int˙zero˙not˙one: shows 0 ≠ 1
proof -
  have ($# 0) ≠ ($# 1) by simp
  then show 0 ≠ 1 using Int˙ZF˙1˙L8 by simp
qed
```

The set of integers is not empty, of course.

```
lemma (in int0) int˙not˙empty: shows ℤ ≠ 0
  using Int˙ZF˙1˙L8A by auto
```

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

```
lemma (in int0) int˙not˙trivial: shows ℤ ≠ –0″
  using Int˙ZF˙1˙L8A int˙zero˙not˙one by blast
```

Each integer has an inverse (in the addition sense).

```
lemma (in int0) Int˙ZF˙1˙L9: assumes A1: g ∈ ℤ
  shows ∃ b∈ℤ. g+b = 0
proof -
  from A1 have g+ $-g = 0
    using Int˙ZF˙1˙L2 Int˙ZF˙1˙L8 by simp
  thus thesis by auto
qed
```

Integers with addition form an abelian group. This also shows that we can apply all theorems proven in the proof contexts (locales) that require the assumpion that some pair of sets form a group like locale group0.

```
theorem Int˙ZF˙1˙T2: shows
  IsAgroup(int,IntegerAddition)
  IntegerAddition –is commutative on″ int
  group0(int,IntegerAddition)
  using int0.Int˙ZF˙1˙T1 int0.Int˙ZF˙1˙L9 IsAgroup˙def
  group0˙def int0.Int˙ZF˙1˙L4 IsCommutative˙def by auto
```

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 assms 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 assms eq˙zdiff˙iff by auto
```

A trivial calculation lemma that allows to subtract and add one, version with ZF-operation.

```
lemma (in int0) Int˙ZF˙1˙L13: assumes m∈ℤ
  shows (m $- $#1) + 1 = m
  using assms Int˙ZF˙1˙L8A Int˙ZF˙1˙L2 Int˙ZF˙1˙L8 Int˙ZF˙1˙L12
  by simp
```

Adding or subtracing one changes integers.

```
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.inv˙cancel˙two
    by simp
  moreover assume m-1 = m
  ultimately have m + 1 = m by simp
  with I have False by simp
 } then show m-1 ≠ m by auto
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
```

42.2 Integers as an ordered group

In this section we define order on integers as a relation, that is a subset of $Z \times Z$ and show that integers form an ordered group.

The next lemma interprets the order definition one way.

```
lemma (in int0) Int˙ZF˙2˙L1:
 assumes A1: m∈ℤ n∈ℤ and A2: m $≤ n
 shows m ≤ n
proof -
 from A1 A2 have ⟨ m,n⟩ ∈ –x∈ℤ×ℤ. fst(x) $≤ snd(x)″
  by simp
 then show thesis using IntegerOrder˙def by simp
qed
```

The next lemma interprets the definition the other way.

```
lemma (in int0) Int˙ZF˙2˙L1A: assumes A1: m ≤ n
 shows m $≤ n m∈ℤ n∈ℤ
proof -
 from A1 have ⟨ m,n⟩ ∈ –p∈ℤ×ℤ. fst(p) $≤ snd(p)″
  using IntegerOrder˙def by simp
```

```
  thus m $≤ n  m∈ℤ  n∈ℤ by auto
qed
```

Integer order is a relation on integers.

```
lemma Int˙ZF˙2˙L1B: shows IntegerOrder ⊆ int×int
proof
  fix x assume x∈IntegerOrder
  then have x ∈ –p∈int×int. fst(p) $≤ snd(p)″
    using IntegerOrder˙def by simp
  then show x∈int×int by simp
qed
```

The way we define the notion of being bounded below, its sufficient for the relation to be on integers for all bounded below sets to be subsets of integers.

```
lemma (in int0) Int˙ZF˙2˙L1C:
  assumes A1: IsBoundedBelow(A,IntegerOrder)
  shows A⊆ℤ
proof -
  from A1 have
    IntegerOrder ⊆ ℤ×ℤ
    IsBoundedBelow(A,IntegerOrder)
    using Int˙ZF˙2˙L1B by auto
  then show A⊆ℤ by (rule Order˙ZF˙3˙L1B)
qed
```

The order on integers is reflexive.

```
lemma (in int0) int˙ord˙is˙refl: shows refl(ℤ,IntegerOrder)
  using Int˙ZF˙2˙L1 zle˙refl refl˙def by auto
```

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

```
lemma (in int0) Int˙ZF˙2˙L3:
  assumes A1: m ≤ n  n ≤ m
  shows m=n
proof -
  from A1 have m $≤ n  n $≤ m  m∈ℤ  n∈ℤ
    using Int˙ZF˙2˙L1A by auto
  then show m=n using zle˙anti˙sym by auto
qed
```

The order on integers is antisymmetric.

```
lemma (in int0) Int˙ZF˙2˙L4: shows antisym(IntegerOrder)
proof -
  have ∀m n. m ≤ n  ∧ n ≤ m ⟶ m=n
    using Int˙ZF˙2˙L3 by auto
  then show thesis using imp˙conj antisym˙def by simp
qed
```

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

```
lemma Int˙ZF˙2˙L5:
  assumes A1: ⟨m,n⟩ ∈ IntegerOrder  ⟨n,k⟩ ∈ IntegerOrder
  shows ⟨m,k⟩ ∈ IntegerOrder
proof -
  from A1 have T1: m $≤ n n $≤ k and T2: m∈int k∈int
    using int0.Int˙ZF˙2˙L1A by auto
  from T1 have m $≤ k by (rule zle˙trans)
  with T2 show thesis using int0.Int˙ZF˙2˙L1 by simp
qed
```

The order on integers is transitive. This version is stated in the int0 context using notation for integers.

```
lemma (in int0) Int˙order˙transitive:
  assumes A1: m≤n  n≤k
  shows m≤k
proof -
  from A1 have ⟨ m,n⟩ ∈ IntegerOrder  ⟨ n,k⟩ ∈ IntegerOrder
    by auto
  then have ⟨ m,k⟩ ∈ IntegerOrder by (rule Int˙ZF˙2˙L5)
  then show m≤k by simp
qed
```

The order on integers is transitive.

```
lemma Int˙ZF˙2˙L6: shows trans(IntegerOrder)
proof -
  have ∀ m n k.
    ⟨m, n⟩ ∈ IntegerOrder ∧ ⟨n, k⟩ ∈ IntegerOrder ⟶
    ⟨m, k⟩ ∈ IntegerOrder
    using Int˙ZF˙2˙L5 by blast
  then show thesis by (rule Fol1˙L2)
qed
```

The order on integers is a partial order.

```
lemma Int˙ZF˙2˙L7: shows IsPartOrder(int,IntegerOrder)
  using int0.int˙ord˙is˙refl int0.Int˙ZF˙2˙L4
    Int˙ZF˙2˙L6 IsPartOrder˙def by simp
```

The essential condition to show that the order on integers is preserved by translations.

```
lemma (in int0) int˙ord˙transl˙inv:
  assumes A1: k ∈ ℤ and A2: m ≤ n
  shows m+k ≤ n+k   k+m≤ k+n
proof -
  from A2 have m $≤ n and m∈ℤ n∈ℤ
    using Int˙ZF˙2˙L1A by auto
  with A1 show m+k ≤ n+k   k+m≤ k+n
    using zadd˙right˙cancel˙zle zadd˙left˙cancel˙zle
    Int˙ZF˙1˙L2 Int˙ZF˙1˙L1 apply˙funtype
```

```
  Int˙ZF˙1˙L2 Int˙ZF˙2˙L1 Int˙ZF˙1˙L2 by auto
qed
```

Integers form a linearly ordered group. We can apply all theorems proven in group3 context to integers.

```
theorem (in int0) Int˙ZF˙2˙T1: shows
  IsAnOrdGroup(ℤ,IntegerAddition,IntegerOrder)
  IntegerOrder –is total on″ ℤ
  group3(ℤ,IntegerAddition,IntegerOrder)
  IsLinOrder(ℤ,IntegerOrder)
proof -
  have ∀k∈ℤ. ∀m n. m ≤ n  ⟶
    m+k ≤ n+k ∧ k+m≤ k+n
    using int˙ord˙transl˙inv by simp
  then show T1: IsAnOrdGroup(ℤ,IntegerAddition,IntegerOrder) using
    Int˙ZF˙1˙T2 Int˙ZF˙2˙L1B Int˙ZF˙2˙L7 IsAnOrdGroup˙def
    by simp
  then show group3(ℤ,IntegerAddition,IntegerOrder)
    using group3˙def by simp
  have ∀n∈ℤ. ∀m∈ℤ. n≤m ∨ m≤n
    using zle˙linear Int˙ZF˙2˙L1 by auto
  then show IntegerOrder –is total on″ ℤ
    using IsTotal˙def by simp
  with T1 show IsLinOrder(ℤ,IntegerOrder)
    using IsAnOrdGroup˙def IsPartOrder˙def IsLinOrder˙def by simp
qed
```

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 assms zle˙def Int˙ZF˙2˙L1 by auto
```

A small technical lemma about putting one on the other side of an inequality.

```
lemma (in int0) Int˙ZF˙2˙L9A:
  assumes A1: k∈ℤ and A2: m ≤ k $- ($# 1)
  shows m+1 ≤ k
proof -
```

```
  from A2 have m+1 ≤ (k $- ($# 1)) + 1
    using Int˙ZF˙1˙L8A int˙ord˙transl˙inv by simp
  with A1 show m+1 ≤ k
    using Int˙ZF˙1˙L13 by simp
qed
```

We can put any integer on the other side of an inequality reversing its sign.

```
lemma (in int0) Int˙ZF˙2˙L9B: assumes i∈ℤ  m∈ℤ  k∈ℤ
  shows i+m ≤ k  ⟷ i ≤ k-m
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L9A
  by simp
```

A special case of Int˙ZF˙2˙L9B with weaker assumptions.

```
lemma (in int0) Int˙ZF˙2˙L9C:
  assumes i∈ℤ  m∈ℤ and i-m ≤ k
  shows i ≤ k+m
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L9B
  by simp
```

Taking (higher order) minus on both sides of inequality reverses it.

```
lemma (in int0) Int˙ZF˙2˙L10: assumes k ≤ i
  shows
  (-i) ≤ (-k)
  $-i ≤ $-k
  using assms Int˙ZF˙2˙L1A Int˙ZF˙1˙L9A Int˙ZF˙2˙T1
    group3.OrderedGroup˙ZF˙1˙L5 by auto
```

Taking minus on both sides of inequality reverses it, version with a negative on one side.

```
lemma (in int0) Int˙ZF˙2˙L10AA: assumes n∈ℤ  m≤(-n)
  shows n≤(-m)
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L5AD
  by simp
```

We can cancel the same element on on both sides of an inequality, a version with minus on both sides.

```
lemma (in int0) Int˙ZF˙2˙L10AB:
  assumes m∈ℤ  n∈ℤ  k∈ℤ and m-n ≤ m-k
  shows k≤n
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L5AF
  by simp
```

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

```
lemma (in int0) Int˙ZF˙2˙L10A: assumes k ≤ 0
  shows 0≤(-k)
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L5A by simp
```

If the opposite of an integers is nonnegative, then the integer is nonpositive.

```
lemma (in int0) Int˙ZF˙2˙L10B:
  assumes k∈ℤ and 0≤(-k)
  shows k≤0
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L5AA by simp
```

Adding one to an integer corresponds to taking a successor for a natural number.

```
lemma (in int0) Int˙ZF˙2˙L11:
  shows i $+ $# n $+ ($# 1)  =  i $+ $# succ(n)
proof -
  have $# succ(n) = $#1 $+ $# n using int˙succ˙int˙1 by blast
  then have i $+ $# succ(n) = i $+ ($# n  $+ $#1)
    using zadd˙commute by simp
  then show thesis using zadd˙assoc by simp
qed
```

Adding a natural number increases integers.

```
lemma (in int0) Int˙ZF˙2˙L12: assumes A1: i∈ℤ and A2: n∈nat
  shows i ≤ i $+ $#n
proof -
  { assume n = 0
    with A1 have i ≤ i $+ $#n using zadd˙int0 int˙ord˙is˙refl refl˙def
      by simp }
  moreover
  { assume n≠0
    with A2 obtain k where k∈nat n = succ(k)
      using Nat˙ZF˙1˙L3 by auto
    with A1 have i ≤ i $+ $#n
      using zless˙succ˙zadd zless˙imp˙zle Int˙ZF˙2˙L1 by simp }
  ultimately show thesis by blast
qed
```

Adding one increases integers.

```
lemma (in int0) Int˙ZF˙2˙L12A: assumes A1: j≤k
  shows j ≤ k $+ $#1  j ≤ k+1
proof -
  from A1 have T1:j∈ℤ k∈ℤ j $≤ k
    using Int˙ZF˙2˙L1A by auto
  moreover from T1 have k $≤ k $+ $#1 using Int˙ZF˙2˙L12 Int˙ZF˙2˙L1A
    by simp
  ultimately have j $≤ k $+ $#1 using zle˙trans by fast
  with T1 show j ≤ k $+ $#1 using Int˙ZF˙2˙L1 by simp
  with T1 have j≤ k+$#1
    using Int˙ZF˙1˙L2 by simp
  then show j ≤ k+1 using Int˙ZF˙1˙L8 by simp
qed
```

Adding one increases integers, yet one more version.

```
lemma (in int0) Int˙ZF˙2˙L12B: assumes A1: m∈ℤ shows m ≤ m+1
```

```
  using assms int˙ord˙is˙refl refl˙def Int˙ZF˙2˙L12A by simp
```

If $k + 1 = m + n$, where n is a non-zero natural number, then $m \leq k$.

```
lemma (in int0) Int˙ZF˙2˙L13:
  assumes A1: k∈ℤ m∈ℤ and A2: n∈nat
  and A3: k $+ ($# 1) = m $+ $# succ(n)
  shows m ≤ k
proof -
  from A1 have k∈ℤ m $+ $# n ∈ ℤ by auto
  moreover from assms have k $+ $# 1 = m $+ $# n $+ $#1
    using Int˙ZF˙2˙L11 by simp
  ultimately have k = m $+ $# n using zadd˙right˙cancel by simp
  with A1 A2 show thesis using Int˙ZF˙2˙L12 by simp
qed
```

The absolute value of an integer is an integer.

```
lemma (in int0) Int˙ZF˙2˙L14: assumes A1: m∈ℤ
  shows abs(m) ∈ ℤ
proof -
  have AbsoluteValue(ℤ,IntegerAddition,IntegerOrder) : ℤ→ℤ
    using Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L1 by simp
  with A1 show thesis using apply˙funtype by simp
qed
```

If two integers are nonnegative, then the opposite of one is less or equal than the other and the sum is also nonnegative.

```
lemma (in int0) Int˙ZF˙2˙L14A:
  assumes 0≤m  0≤n
  shows
  (-m) ≤ n
  0 ≤ m + n
  using assms Int˙ZF˙2˙T1
    group3.OrderedGroup˙ZF˙1˙L5AC group3.OrderedGroup˙ZF˙1˙L12
  by auto
```

We can increase components in an estimate.

```
lemma (in int0) Int˙ZF˙2˙L15:
  assumes b≤b1 c≤c1 and a≤b+c
  shows a≤b1+c1
proof -
  from assms have group3(ℤ,IntegerAddition,IntegerOrder)
    ⟨a,IntegerAddition⟨ b,c⟩⟩ ∈ IntegerOrder
    ⟨b,b1⟩ ∈ IntegerOrder ⟨c,c1⟩ ∈ IntegerOrder
    using Int˙ZF˙2˙T1 by auto
  then have ⟨a,IntegerAddition⟨ b1,c1⟩⟩ ∈ IntegerOrder
    by (rule group3.OrderedGroup˙ZF˙1˙L5E)
  thus thesis by simp
qed
```

We can add or subtract the sides of two inequalities.

```
lemma (in int0) int˙ineq˙add˙sides:
  assumes a≤b and c≤d
  shows
  a+c ≤ b+d
  a-d ≤ b-c
  using assms Int˙ZF˙2˙T1
    group3.OrderedGroup˙ZF˙1˙L5B group3.OrderedGroup˙ZF˙1˙L5I
  by auto
```

We can increase the second component in an estimate.

```
lemma (in int0) Int˙ZF˙2˙L15A:
  assumes b∈ℤ and a≤b+c and A3: c≤c1
  shows a≤b+c1
proof -
  from assms have
    group3(ℤ,IntegerAddition,IntegerOrder)
    b ∈ ℤ
    ⟨a,IntegerAddition⟨ b,c⟩⟩ ∈ IntegerOrder
    ⟨c,c1⟩ ∈ IntegerOrder
    using Int˙ZF˙2˙T1 by auto
  then have ⟨a,IntegerAddition⟨ b,c1⟩⟩ ∈ IntegerOrder
     by (rule group3.OrderedGroup˙ZF˙1˙L5D)
   thus thesis by simp
qed
```

If we increase the second component in a sum of three integers, the whole sum inceases.

```
lemma (in int0) Int˙ZF˙2˙L15C:
  assumes A1: m∈ℤ  n∈ℤ and A2: k ≤ L
  shows m+k+n ≤ m+L+n
proof -
  let P = IntegerAddition
  from assms have
    group3(int,P,IntegerOrder)
    m ∈ int  n ∈ int
    ⟨k,L⟩ ∈ IntegerOrder
    using Int˙ZF˙2˙T1 by auto
  then have ⟨P⟨P⟨ m,k⟩,n⟩, P⟨P⟨ m,L⟩,n⟩ ⟩ ∈ IntegerOrder
    by (rule group3.OrderedGroup˙ZF˙1˙L10)
  then show m+k+n ≤ m+L+n by simp
qed
```

We don't decrease an integer by adding a nonnegative one.

```
lemma (in int0) Int˙ZF˙2˙L15D:
  assumes 0≤n  m∈ℤ
  shows m ≤ n+m
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L5F
```

```
  by simp
```

Some inequalities about the sum of two integers and its absolute value.

```
lemma (in int0) Int˙ZF˙2˙L15E:
  assumes m∈ℤ  n∈ℤ
  shows
  m+n ≤ abs(m)+abs(n)
  m-n ≤ abs(m)+abs(n)
  (-m)+n ≤ abs(m)+abs(n)
  (-m)-n ≤ abs(m)+abs(n)
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L6A
  by auto
```

We can add a nonnegative integer to the right hand side of an inequality.

```
lemma (in int0) Int˙ZF˙2˙L15F:  assumes m≤k  and 0≤n
  shows m ≤ k+n  m ≤ n+k
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L5G
  by auto
```

Triangle inequality for integers.

```
lemma (in int0) Int˙triangle˙ineq:
  assumes m∈ℤ  n∈ℤ
  shows abs(m+n)≤abs(m)+abs(n)
  using assms Int˙ZF˙1˙T2 Int˙ZF˙2˙T1 group3.OrdGroup˙triangle˙ineq
  by simp
```

Taking absolute value does not change nonnegative integers.

```
lemma (in int0) Int˙ZF˙2˙L16:
  assumes 0≤m shows  m∈ℤ⁺ and abs(m) = m
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L2
    group3.OrderedGroup˙ZF˙3˙L2 by auto
```

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

```
lemma (in int0) Int˙ZF˙2˙L16A: shows 0≤1 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 assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L7A by simp
```

The absolute value of zero is zero.

```
lemma (in int0) Int˙ZF˙2˙L18: shows abs(0) = 0
  using Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L2A by simp
```

A different version of the triangle inequality.

```
lemma (in int0) Int˙triangle˙ineq1:
  assumes A1: m∈ℤ  n∈ℤ
  shows
  abs(m-n) ≤ abs(n)+abs(m)
  abs(m-n) ≤ abs(m)+abs(n)
proof -
  have $-n ∈ ℤ by simp
  with A1 have abs(m-n) ≤ abs(m)+abs(-n)
    using Int˙ZF˙1˙L9A Int˙triangle˙ineq by simp
  with A1 show
    abs(m-n) ≤ abs(n)+abs(m)
    abs(m-n) ≤ abs(m)+abs(n)
```

```
    using Int˙ZF˙2˙L17 Int˙ZF˙2˙L14 Int˙ZF˙1˙T2 IsCommutative˙def
    by auto
qed
```

Another version of the triangle inequality.

```
lemma (in int0) Int˙triangle˙ineq2:
  assumes m∈ℤ  n∈ℤ
  and abs(m-n) ≤ k
  shows
  abs(m) ≤ abs(n)+k
  m-k ≤ n
  m ≤ n+k
  n-k ≤ m
  using assms Int˙ZF˙1˙T2 Int˙ZF˙2˙T1
    group3.OrderedGroup˙ZF˙3˙L7D group3.OrderedGroup˙ZF˙3˙L7E
  by auto
```

Triangle inequality with three integers. We could use OrdGroup˙triangle˙ineq3, but since simp cannot translate the notation directly, it is simpler to reprove it for integers.

```
lemma (in int0) Int˙triangle˙ineq3:
  assumes A1: m∈ℤ  n∈ℤ  k∈ℤ
  shows abs(m+n+k) ≤ abs(m)+abs(n)+abs(k)
proof -
  from A1 have T: m+n ∈ ℤ  abs(k) ∈ ℤ
    using Int˙ZF˙1˙T2 group0.group˙op˙closed  Int˙ZF˙2˙L14
    by auto
  with A1 have abs(m+n+k) ≤ abs(m+n) + abs(k)
    using Int˙triangle˙ineq by simp
  moreover from A1 T have
    abs(m+n) + abs(k) ≤ abs(m) + abs(n) + abs(k)
    using Int˙triangle˙ineq int˙ord˙transl˙inv by simp
  ultimately show thesis by (rule Int˙order˙transitive)
qed
```

The next lemma shows what happens when one integers is not greater or equal than another.

```
lemma (in int0) Int˙ZF˙2˙L19:
  assumes A1: m∈ℤ  n∈ℤ and A2: ¬(n≤m)
  shows m≤n  (-n) ≤ (-m)  m≠n
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 T: 𝟬 ∈ ℤ
    using Int˙ZF˙1˙T2 group0.group0˙2˙L2 by auto
  with A1 A2 show m≤𝟬 using Int˙ZF˙2˙L19 by blast
  from A1 T A2 show m≠𝟬  by (rule Int˙ZF˙2˙L19)
  from A1 T A2 have (-𝟬)≤(-m) by (rule Int˙ZF˙2˙L19)
  then show 𝟬 ≤ (-m)
    using Int˙ZF˙1˙T2 group0.group˙inv˙of˙one by simp
qed
```

We can prove a theorem about integers by proving that it holds for $m = 0$, $m \in \mathbb{Z}_+$ and $-m \in \mathbb{Z}_+$.

```
lemma (in int0) Int˙ZF˙2˙L19B:
  assumes m∈ℤ and Q(𝟬) and ∀n∈ℤ+. Q(n) and ∀n∈ℤ+. Q(-n)
  shows Q(m)
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  let b = m
  from assms have
    group3(G, P, r)
    r –is total on″ G
    b ∈ G
    Q(TheNeutralElement(G, P))
    ∀a∈PositiveSet(G, P, r). Q(a)
    ∀a∈PositiveSet(G, P, r). Q(GroupInv(G, P)(a))
```

```
    using Int˙ZF˙2˙T1 by auto
  then show Q(b) by (rule group3.OrderedGroup˙ZF˙1˙L18)
qed
```

An integer is not greater than its absolute value.

```
lemma (in int0) Int˙ZF˙2˙L19C: assumes A1: m∈ℤ
  shows
  m ≤ abs(m)
  (-m) ≤ abs(m)
  using assms Int˙ZF˙2˙T1
    group3.OrderedGroup˙ZF˙3˙L5 group3.OrderedGroup˙ZF˙3˙L6
  by auto
```

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

```
lemma (in int0) Int˙ZF˙2˙L20: assumes m∈ℤ  n∈ℤ
  shows abs(m-n) = abs(n-m)
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L7B by simp
```

We can add the sides of inequalities with absolute values.

```
lemma (in int0) Int˙ZF˙2˙L21:
  assumes A1: m∈ℤ n∈ℤ
  and A2: abs(m) ≤ k  abs(n) ≤ l
  shows
  abs(m+n) ≤ k + l
  abs(m-n) ≤ k + l
  using assms Int˙ZF˙1˙T2 Int˙ZF˙2˙T1
    group3.OrderedGroup˙ZF˙3˙L7C group3.OrderedGroup˙ZF˙3˙L7CA
  by auto
```

Absolute value is nonnegative.

```
lemma (in int0) int˙abs˙nonneg: assumes A1: m∈ℤ
  shows abs(m) ∈ ℤ⁺  𝟬 ≤ abs(m)
proof -
  have AbsoluteValue(ℤ,IntegerAddition,IntegerOrder) : ℤ→ℤ⁺
    using Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L3C by simp
  with A1 show abs(m) ∈ ℤ⁺ using apply˙funtype
    by simp
  then show 𝟬 ≤ abs(m)
    using Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L2 by simp
qed
```

If an nonnegative integer is less or equal than another, then so is its absolute value.

```
lemma (in int0) Int˙ZF˙2˙L23:
  assumes 𝟬≤m   m≤k
  shows abs(m) ≤ k
  using assms Int˙ZF˙2˙L16 by simp
```

42.3 Induction on integers.

In this section we show some induction lemmas for integers. The basic tools are the induction on natural numbers and the fact that integers can be written as a sum of a smaller integer and a natural number.

An integer can be written a a sum of a smaller integer and a natural number.

```
lemma (in int0) Int˙ZF˙3˙L2: assumes A1: i ≤ m
  shows ∃n∈nat. m = i $+ $# n
proof -
  let n = 0
  { assume A2: i=m
    from A1 A2 have n ∈ nat m = i $+ $# n
      using Int˙ZF˙2˙L1A zadd˙int0˙right by auto
    hence ∃n∈nat. m = i $+ $# n by blast }
  moreover
  { assume A3: i≠m
    with A1 have i $¡ m i∈ℤ m∈ℤ
      using Int˙ZF˙2˙L9 Int˙ZF˙2˙L1A by auto
    then obtain k where D1: k∈nat m = i $+ $# succ(k)
      using zless˙imp˙succ˙zadd˙lemma by auto
    let n = succ(k)
    from D1 have n∈nat m = i $+ $# n by auto
    hence ∃n∈nat. m = i $+ $# n by simp }
  ultimately show thesis by blast
qed
```

Induction for integers, the induction step.

```
lemma (in int0) Int˙ZF˙3˙L6: assumes A1: i∈ℤ
  and A2: ∀m. i≤m ∧ Q(m) ⟶ Q(m $+ ($# 1))
  shows ∀k∈nat. Q(i $+ ($# k)) ⟶ Q(i $+ ($# succ(k)))
proof
  fix k assume A3: k∈nat show Q(i $+ $# k) ⟶ Q(i $+ $# succ(k))
  proof
    assume A4: Q(i $+ $# k)
    from A1 A3 have i≤ i $+ ($# k) using Int˙ZF˙2˙L12
      by simp
    with A4 A2 have Q(i $+ ($# k) $+ ($# 1)) by simp
    then show Q(i $+ ($# succ(k))) using Int˙ZF˙2˙L11 by simp
  qed
qed
```

Induction on integers, version with higher-order increment function.

```
lemma (in int0) Int˙ZF˙3˙L7:
  assumes A1: i≤k and A2: Q(i)
  and A3: ∀m. i≤m ∧ Q(m) ⟶ Q(m $+ ($# 1))
  shows Q(k)
proof -
  from A1 obtain n where D1: n∈nat and D2: k = i $+ $# n
```

```
    using Int˙ZF˙3˙L2 by auto
  from A1 have T1: i∈ℤ using Int˙ZF˙2˙L1A by simp
  note ‹n∈nat›
  moreover from A1 A2 have Q(i $+ $#0)
    using Int˙ZF˙2˙L1A zadd˙int0 by simp
  moreover from T1 A3 have
    ∀k∈nat. Q(i $+ ($# k)) ⟶ Q(i $+ ($# succ(k)))
    by (rule Int˙ZF˙3˙L6)
  ultimately have Q(i $+ ($# n)) by (rule ind˙on˙nat)
  with D2 show Q(k) by simp
qed
```

Induction on integer, implication between two forms of the induction step.

```
lemma (in int0) Int˙ZF˙3˙L7A: assumes
  A1: ∀m. i≤m ∧ Q(m) ⟶ Q(m+𝟭)
  shows ∀m. i≤m ∧ Q(m) ⟶ Q(m $+ ($# 1))
proof -
  { fix m assume i≤m ∧ Q(m)
    with A1 have T1: m∈ℤ Q(m+𝟭) using Int˙ZF˙2˙L1A by auto
    then have m+𝟭 = m+($# 1) using Int˙ZF˙1˙L8 by simp
    with T1 have Q(m $+ ($# 1)) using Int˙ZF˙1˙L2
      by simp
  } then show thesis by simp
qed
```

Induction on integers, version with ZF increment function.

```
theorem (in int0) Induction˙on˙int:
  assumes A1: i≤k and A2: Q(i)
  and A3: ∀m. i≤m ∧ Q(m) ⟶ Q(m+𝟭)
  shows Q(k)
proof -
  from A3 have ∀m. i≤m ∧ Q(m) ⟶ Q(m $+ ($# 1))
    by (rule Int˙ZF˙3˙L7A)
  with A1 A2 show thesis by (rule Int˙ZF˙3˙L7)
qed
```

Another form of induction on integers. This rewrites the basic theorem Int˙ZF˙3˙L7 substituting $P(-k)$ for $Q(k)$.

```
lemma (in int0) Int˙ZF˙3˙L7B: assumes A1: i≤k and A2: P($-i)
  and A3: ∀m. i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
  shows P($-k)
proof -
  from A1 A2 A3 show P($-k) by (rule Int˙ZF˙3˙L7)
qed
```

Another induction on integers. This rewrites Int_ZF_3_L7 substituting $-k$ for k and $-i$ for i.

```
lemma (in int0) Int˙ZF˙3˙L8: assumes A1: k≤i and A2: P(i)
```

```
  and A3: ∀m. $-i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
  shows P(k)
proof -
  from A1 have T1: $-i≤$-k using Int˙ZF˙2˙L10 by simp
  from A1 A2 have T2: P($- $- i) using Int˙ZF˙2˙L1A zminus˙zminus
    by simp
  from T1 T2 A3 have P($-($-k)) by (rule Int˙ZF˙3˙L7)
  with A1 show P(k) using Int˙ZF˙2˙L1A zminus˙zminus by simp
qed
```

An implication between two forms of induction steps.

```
lemma (in int0) Int˙ZF˙3˙L9: assumes A1: i∈ℤ
  and A2: ∀n. n≤i ∧ P(n) ⟶ P(n $+ $-($#1))
  shows ∀m. $-i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
proof
  fix m show $-i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
  proof
    assume A3: $- i ≤ m ∧ P($- m)
    then have $- i ≤ m by simp
    then have $-m ≤ $- ($- i) by (rule Int˙ZF˙2˙L10)
    with A1 A2 A3 show P($-(m $+ ($# 1)))
      using zminus˙zminus zminus˙zadd˙distrib by simp
  qed
qed
```

Backwards induction on integers, version with higher-order decrement function.

```
lemma (in int0) Int˙ZF˙3˙L9A: assumes A1: k≤i and A2: P(i)
  and A3: ∀n. n≤i ∧ P(n) ⟶P(n $+ $-($#1))
  shows P(k)
proof -
  from A1 have T1: i∈ℤ using Int˙ZF˙2˙L1A by simp
  from T1 A3 have T2: ∀m. $-i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
    by (rule Int˙ZF˙3˙L9)
  from A1 A2 T2 show P(k) by (rule Int˙ZF˙3˙L8)
qed
```

Induction on integers, implication between two forms of the induction step.

```
lemma (in int0) Int˙ZF˙3˙L10: assumes
  A1: ∀n. n≤i ∧ P(n) ⟶ P(n-1)
  shows ∀n. n≤i ∧ P(n) ⟶ P(n $+ $-($#1))
proof -
  { fix n assume n≤i ∧ P(n)
    with A1 have T1: n∈ℤ P(n-1) using Int˙ZF˙2˙L1A by auto
    then have n-1 = n-($# 1) using Int˙ZF˙1˙L8 by simp
    with T1 have P(n $+ $-($#1)) using Int˙ZF˙1˙L10 by simp
  } then show thesis by simp
qed
```

Backwards induction on integers.

```
theorem (in int0) Back˙induct˙on˙int:
  assumes A1: k≤i and A2: P(i)
  and A3: ∀n. n≤i ∧ P(n) ⟶ P(n-1)
  shows P(k)
proof -
  from A3 have ∀n. n≤i ∧ P(n) ⟶ P(n $+ $-($#1))
    by (rule Int˙ZF˙3˙L10)
  with A1 A2 show P(k) by (rule Int˙ZF˙3˙L9A)
qed
```

42.4 Bounded vs. finite subsets of integers

The goal of this section is to establish that a subset of integers is bounded is and only is it is finite. The fact that all finite sets are bounded is already shown for all linearly ordered groups in OrderedGroups˙ZF.thy. To show the other implication we show that all intervals starting at 0 are finite and then use a result from OrderedGroups˙ZF.thy.

There are no integers between k and $k+1$.

```
lemma (in int0) Int˙ZF˙4˙L1:
  assumes A1: k∈ℤ m∈ℤ n∈nat and A2: k $+ $#1 = m $+ $#n
  shows m =  k $+ $#1 ∨ m ≤ k
proof -
  { assume n=0
    with A1 A2 have m =  k $+ $#1 ∨ m ≤ k
      using zadd˙int0 by simp }
  moreover
  { assume n≠0
    with A1 obtain j where D1: j∈nat n = succ(j)
      using Nat˙ZF˙1˙L3 by auto
    with A1 A2 D1 have m =  k $+ $#1 ∨ m ≤ k
      using Int˙ZF˙2˙L13 by simp }
  ultimately show thesis by blast
qed
```

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

```
lemma Int˙ZF˙4˙L1A:
  assumes m∈int shows m $- $#1 $+ $#1 = m
  using assms eq˙zdiff˙iff by auto
```

There are no integers between k and $k+1$, another formulation.

```
lemma (in int0) Int˙ZF˙4˙L1B: assumes A1: m ≤ L
  shows
  m = L ∨ m+1 ≤ L
  m = L ∨ m ≤ L-1
proof -
```

```
  let k = L $- $#1
  from A1 have T1: m∈ℤ  L∈ℤ  L = k $+ $#1
    using Int˙ZF˙2˙L1A Int˙ZF˙4˙L1A by auto
  moreover from A1 obtain n where D1: n∈nat  L = m $+ $# n
    using Int˙ZF˙3˙L2 by auto
  ultimately have m = L ∨ m ≤ k
    using Int˙ZF˙4˙L1 by simp
  with T1 show m = L   ∨  m+1 ≤ L
    using Int˙ZF˙2˙L9A by auto
  with T1 show m = L ∨ m ≤ L-1
    using Int˙ZF˙1˙L8A Int˙ZF˙2˙L9B by simp
qed
```

If $j \in m..k+1$, then $j \in m..n$ or $j = k+1$.

```
lemma (in int0) Int˙ZF˙4˙L2: assumes A1: k∈ℤ
  and A2: j ∈ m..(k $+ $#1)
  shows j ∈ m..k ∨ j ∈ –k $+ $#1″
proof -
  from A2 have T1: m≤j j≤(k $+ $#1) using Order˙ZF˙2˙L1A
    by auto
  then have T2: m∈ℤ j∈ℤ using Int˙ZF˙2˙L1A by auto
  from T1 obtain n where n∈nat k $+ $#1 = j $+ $# n
    using Int˙ZF˙3˙L2 by auto
  with A1 T1 T2 have (m≤j ∧ j ≤ k) ∨ j ∈ –k $+ $#1″
    using Int˙ZF˙4˙L1 by auto
  then show thesis using Order˙ZF˙2˙L1B by auto
qed
```

Extending an integer interval by one is the same as adding the new endpoint.

```
lemma (in int0) Int˙ZF˙4˙L3: assumes A1: m≤ k
  shows m..(k $+ $#1) = m..k ∪ –k $+ $#1″
proof
  from A1 have T1: m∈ℤ k∈ℤ using Int˙ZF˙2˙L1A by auto
  then show m .. (k $+ $# 1) ⊆ m .. k ∪ –k $+ $# 1″
    using Int˙ZF˙4˙L2 by auto
  from T1 have m≤ m using Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L3
    by simp
  with T1 A1 have m .. k ⊆ m .. (k $+ $# 1)
    using Int˙ZF˙2˙L12 Int˙ZF˙2˙L6 Order˙ZF˙2˙L3 by simp
  with T1 A1 show m..k ∪ –k $+ $#1″ ⊆ m..(k $+ $#1)
    using Int˙ZF˙2˙L12A int˙ord˙is˙refl Order˙ZF˙2˙L2 by auto
qed
```

Integer intervals are finite - induction step.

```
lemma (in int0) Int˙ZF˙4˙L4:
  assumes A1: i≤m and A2: i..m ∈ Fin(ℤ)
  shows i..(m $+ $#1) ∈ Fin(ℤ)
  using assms Int˙ZF˙4˙L3 by simp
```

Integer intervals are finite.

```
lemma (in int0) Int˙ZF˙4˙L5: assumes A1: i∈ℤ k∈ℤ
  shows i..k ∈ Fin(ℤ)
proof -
  { assume A2: i≤k
    moreover from A1 have i..i ∈ Fin(ℤ)
      using int˙ord˙is˙refl Int˙ZF˙2˙L4 Order˙ZF˙2˙L4 by simp
    moreover from A2 have
      ∀m. i≤m ∧ i..m ∈ Fin(ℤ) ⟶ i..(m $+ $#1) ∈ Fin(ℤ)
      using Int˙ZF˙4˙L4 by simp
    ultimately have i..k ∈ Fin(ℤ) by (rule Int˙ZF˙3˙L7) }
  moreover
  { assume ¬ i ≤ k
    then have i..k ∈ Fin(ℤ) using Int˙ZF˙2˙L6 Order˙ZF˙2˙L5
      by simp }
  ultimately show thesis by blast
qed
```

Bounded integer sets are finite.

```
lemma (in int0) Int˙ZF˙4˙L6: assumes A1: IsBounded(A,IntegerOrder)
  shows A ∈ Fin(ℤ)
proof -
  have T1: ∀m ∈ Nonnegative(ℤ,IntegerAddition,IntegerOrder).
    $#0..m ∈ Fin(ℤ)
  proof
    fix m assume m ∈ Nonnegative(ℤ,IntegerAddition,IntegerOrder)
    then have m∈ℤ using Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L4E
      by auto
    then show $#0..m ∈ Fin(ℤ) using Int˙ZF˙4˙L5 by simp
  qed
  have group3(ℤ,IntegerAddition,IntegerOrder)
    using Int˙ZF˙2˙T1 by simp
  moreover from T1 have ∀m ∈ Nonnegative(ℤ,IntegerAddition,IntegerOrder).
    Interval(IntegerOrder,TheNeutralElement(ℤ,IntegerAddition),m)
    ∈ Fin(ℤ) using Int˙ZF˙1˙L8 by simp
  moreover note A1
  ultimately show A ∈ Fin(ℤ) by (rule group3.OrderedGroup˙ZF˙2˙T1)
qed
```

A subset of integers is bounded iff it is finite.

```
theorem (in int0) Int˙bounded˙iff˙fin:
  shows IsBounded(A,IntegerOrder)⟷ A∈Fin(ℤ)
  using Int˙ZF˙4˙L6 Int˙ZF˙2˙T1 group3.ord˙group˙fin˙bounded
  by blast
```

The image of an interval by any integer function is finite, hence bounded.

```
lemma (in int0) Int˙ZF˙4˙L8:
  assumes A1: i∈ℤ  k∈ℤ and A2: f:ℤ→ℤ
```

shows
f(i..k) ∈ Fin(ℤ)
IsBounded(f(i..k),IntegerOrder)
using assms Int˙ZF˙4˙L5 Finite1˙L6A Int˙bounded˙iff˙fin
by auto

If for every integer we can find one in A that is greater or equal, then A is is not bounded above, hence infinite.

lemma (in int0) Int˙ZF˙4˙L9: assumes A1: ∀m∈ℤ. ∃k∈A. m≤k
shows
¬IsBoundedAbove(A,IntegerOrder)
A ∉ Fin(ℤ)
proof -
have ℤ ≠ −**0**″
using Int˙ZF˙1˙L8A int˙zero˙not˙one by blast
with A1 show
¬IsBoundedAbove(A,IntegerOrder)
A ∉ Fin(ℤ)
using Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙2˙L2A
by auto
qed

end

43 Integers 1

theory Int˙ZF˙1 imports Int˙ZF˙IML OrderedRing˙ZF

begin

This theory file considers the set of integers as an ordered ring.

43.1 Integers as a ring

In this section we show that integers form a commutative ring.

The next lemma provides the condition to show that addition is distributive with respect to multiplication.

lemma (in int0) Int˙ZF˙1˙1˙L1: assumes A1: a∈ℤ b∈ℤ c∈ℤ
shows
a·(b+c) = a·b + a·c
(b+c)·a = b·a + c·a
using assms Int˙ZF˙1˙L2 zadd˙zmult˙distrib zadd˙zmult˙distrib2
by auto

Integers form a commutative ring, hence we can use theorems proven in ring0 context (locale).

```
lemma (in int0) Int˙ZF˙1˙1˙L2: shows
  IsAring(ℤ,IntegerAddition,IntegerMultiplication)
  IntegerMultiplication –is commutative on″ ℤ
  ring0(ℤ,IntegerAddition,IntegerMultiplication)
proof -
  have ∀a∈ℤ.∀b∈ℤ.∀c∈ℤ.
    a·(b+c) = a·b + a·c ∧ (b+c)·a = b·a + c·a
    using Int˙ZF˙1˙1˙L1 by simp
  then have IsDistributive(ℤ,IntegerAddition,IntegerMultiplication)
    using IsDistributive˙def by simp
  then show IsAring(ℤ,IntegerAddition,IntegerMultiplication)
    ring0(ℤ,IntegerAddition,IntegerMultiplication)
    using Int˙ZF˙1˙T1 Int˙ZF˙1˙T2 IsAring˙def ring0˙def
    by auto
  have ∀a∈ℤ.∀b∈ℤ. a·b = b·a using Int˙ZF˙1˙L4 by simp
  then show IntegerMultiplication –is commutative on″ ℤ
    using IsCommutative˙def by simp
qed
```

Zero and one are integers.

```
lemma (in int0) int˙zero˙one˙are˙int: shows 0∈ℤ  1∈ℤ
  using Int˙ZF˙1˙1˙L2 ring0.Ring˙ZF˙1˙L2 by auto
```

Negative of zero is zero.

```
lemma (in int0) int˙zero˙one˙are˙intA: shows (-0) = 0
  using Int˙ZF˙1˙T2 group0.group˙inv˙of˙one by simp
```

Properties with one integer.

```
lemma (in int0) Int˙ZF˙1˙1˙L4: assumes A1: a ∈ ℤ
  shows
  a+0 = a
  0+a = a
  a·1 = a   1·a = a
  0·a = 0   a·0 = 0
  (-a) ∈ ℤ  (-(-a)) = a
  a-a = 0   a-0 = a  2·a = a+a
proof -
  from A1 show
    a+0 = a   0+a = a   a·1 = a
    1·a = a   a-a = 0   a-0 = a
    (-a) ∈ ℤ  2·a = a+a   (-(-a)) = a
    using Int˙ZF˙1˙1˙L2 ring0.Ring˙ZF˙1˙L3 by auto
  from A1 show 0·a = 0   a·0 = 0
    using Int˙ZF˙1˙1˙L2 ring0.Ring˙ZF˙1˙L6 by auto
qed
```

Properties that require two integers.

```
lemma (in int0) Int˙ZF˙1˙1˙L5: assumes a∈ℤ  b∈ℤ
```

shows
a+b ∈ ℤ
a-b ∈ ℤ
a·b ∈ ℤ
a+b = b+a
a·b = b·a
(-b)-a = (-a)-b
(-(a+b)) = (-a)-b
(-(a-b)) = ((-a)+b)
(-a)·b = -(a·b)
a·(-b) = -(a·b)
(-a)·(-b) = a·b
using assms Int˙ZF˙1˙1˙L2 ring0.Ring˙ZF˙1˙L4 ring0.Ring˙ZF˙1˙L9
ring0.Ring˙ZF˙1˙L7 ring0.Ring˙ZF˙1˙L7A Int˙ZF˙1˙L4 by auto

2 and 3 are integers.

lemma (in int0) int˙two˙three˙are˙int: shows **2** ∈ ℤ **3** ∈ ℤ
using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L5 by auto

Another property with two integers.

lemma (in int0) Int˙ZF˙1˙1˙L5B:
assumes a∈ℤ b∈ℤ
shows a-(-b) = a+b
using assms Int˙ZF˙1˙1˙L2 ring0.Ring˙ZF˙1˙L9
by simp

Properties that require three integers.

lemma (in int0) Int˙ZF˙1˙1˙L6: assumes a∈ℤ b∈ℤ c∈ℤ
shows
a-(b+c) = a-b-c
a-(b-c) = a-b+c
a·(b-c) = a·b - a·c
(b-c)·a = b·a - c·a
using assms Int˙ZF˙1˙1˙L2 ring0.Ring˙ZF˙1˙L10 ring0.Ring˙ZF˙1˙L8
by auto

One more property with three integers.

lemma (in int0) Int˙ZF˙1˙1˙L6A: assumes a∈ℤ b∈ℤ c∈ℤ
shows a+(b-c) = a+b-c
using assms Int˙ZF˙1˙1˙L2 ring0.Ring˙ZF˙1˙L10A by simp

Associativity of addition and multiplication.

lemma (in int0) Int˙ZF˙1˙1˙L7: assumes a∈ℤ b∈ℤ c∈ℤ
shows
a+b+c = a+(b+c)
a·b·c = a·(b·c)
using assms Int˙ZF˙1˙1˙L2 ring0.Ring˙ZF˙1˙L11 by auto

43.2 Rearrangement lemmas

In this section we collect lemmas about identities related to rearranging the terms in expresssions

A formula with a positive integer.

```
lemma (in int0) Int˙ZF˙1˙2˙L1: assumes 0≤a
  shows abs(a)+1 = abs(a+1)
  using assms Int˙ZF˙2˙L16 Int˙ZF˙2˙L12A by simp
```

A formula with two integers, one positive.

```
lemma (in int0) Int˙ZF˙1˙2˙L2: assumes A1: a∈ℤ and A2: 0≤b
  shows a+(abs(b)+1)·a = (abs(b+1)+1)·a
proof -
  from A2 have abs(b+1) ∈ ℤ
    using Int˙ZF˙2˙L12A Int˙ZF˙2˙L1A Int˙ZF˙2˙L14 by blast
  with A1 A2 show thesis
    using Int˙ZF˙1˙2˙L1 Int˙ZF˙1˙1˙L2 ring0.Ring˙ZF˙2˙L1
    by simp
qed
```

A couple of formulae about canceling opposite integers.

```
lemma (in int0) Int˙ZF˙1˙2˙L3: assumes A1: a∈ℤ  b∈ℤ
  shows
  a+b-a = b
  a+(b-a) = b
  a+b-b = a
  a-b+b = a
  (-a)+(a+b) = b
  a+(b-a) = b
  (-b)+(a+b) = a
  a-(b+a) = -b
  a-(a+b) = -b
  a-(a-b) = b
  a-b-a =  -b
  a-b - (a+b) = (-b)-b
  using assms Int˙ZF˙1˙T2 group0.group0˙4˙L6A group0.inv˙cancel˙two
    group0.group0˙2˙L16A group0.group0˙4˙L6AA group0.group0˙4˙L6AB
    group0.group0˙4˙L6F group0.group0˙4˙L6AC by auto
```

Subtracting one does not increase integers. This may be moved to a theory about ordered rings one day.

```
lemma (in int0) Int˙ZF˙1˙2˙L3A: assumes A1: a≤b
  shows a-1 ≤ b
proof -
  from A1 have b+1-1 = b
    using Int˙ZF˙2˙L1A int˙zero˙one˙are˙int Int˙ZF˙1˙2˙L3 by simp
  moreover from A1 have a-1 ≤ b+1-1
```

```
    using Int˙ZF˙2˙L12A int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L4 int˙ord˙transl˙inv
    by simp
  ultimately show a-1 ≤ b by simp
qed
```

Subtracting one does not increase integers, special case.

```
lemma (in int0) Int˙ZF˙1˙2˙L3AA:
  assumes A1: a∈ℤ shows
  a-1 ≤a
  a-1 ≠ a
  ¬(a≤a-1)
  ¬(a+1 ≤a)
  ¬(1+a ≤a)
proof -
  from A1 have a≤a using int˙ord˙is˙refl refl˙def
    by simp
  then show a-1 ≤a using Int˙ZF˙1˙2˙L3A
    by simp
  moreover from A1 show a-1 ≠ a using Int˙ZF˙1˙L14 by simp
  ultimately show I: ¬(a≤a-1) using Int˙ZF˙2˙L19AA
    by blast
  with A1 show ¬(a+1 ≤a)
    using int˙zero˙one˙are˙int Int˙ZF˙2˙L9B by simp
  with A1 show ¬(1+a ≤a)
    using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L5 by simp
qed
```

A formula with a nonpositive integer.

```
lemma (in int0) Int˙ZF˙1˙2˙L4: assumes a≤0
  shows abs(a)+1 = abs(a-1)
  using assms int˙zero˙one˙are˙int Int˙ZF˙1˙2˙L3A Int˙ZF˙2˙T1
      group3.OrderedGroup˙ZF˙3˙L3A Int˙ZF˙2˙L1A
      int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L5 by simp
```

A formula with two integers, one negative.

```
lemma (in int0) Int˙ZF˙1˙2˙L5: assumes A1: a∈ℤ and A2: b≤0
  shows a+(abs(b)+1)·a = (abs(b-1)+1)·a
proof -
  from A2 have abs(b-1) ∈ ℤ
    using int˙zero˙one˙are˙int Int˙ZF˙1˙2˙L3A Int˙ZF˙2˙L1A Int˙ZF˙2˙L14
    by blast
  with A1 A2 show thesis
    using Int˙ZF˙1˙2˙L4 Int˙ZF˙1˙1˙L2 ring0.Ring˙ZF˙2˙L1
    by simp
qed
```

A rearrangement with four integers.

```
lemma (in int0) Int˙ZF˙1˙2˙L6:
```

```
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ  d∈ℤ
  shows
  a-(b-1)·c = (d-b·c)-(d-a-c)
proof -
  from A1 have T1:
    (d-b·c) ∈ ℤ d-a ∈ ℤ (-(b·c)) ∈ ℤ
    using Int˙ZF˙1˙1˙L5 Int˙ZF˙1˙1˙L4 by auto
  with A1 have
    (d-b·c)-(d-a-c) = (-(b·c))+a+c
    using Int˙ZF˙1˙1˙L6 Int˙ZF˙1˙2˙L3 by simp
  also from A1 T1 have (-(b·c))+a+c = a-(b-1)·c
    using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L6 Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙1˙L5
    by simp
  finally show thesis by simp
qed
```

Some other rearrangements with two integers.

```
lemma (in int0) Int˙ZF˙1˙2˙L7: assumes a∈ℤ  b∈ℤ
  shows
  a·b = (a-1)·b+b
  a·(b+1) = a·b+a
  (b+1)·a = b·a+a
  (b+1)·a = a+b·a
  using assms Int˙ZF˙1˙1˙L1 Int˙ZF˙1˙1˙L5 int˙zero˙one˙are˙int
    Int˙ZF˙1˙1˙L6 Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙T2 group0.inv˙cancel˙two
  by auto
```

Another rearrangement with two integers.

```
lemma (in int0) Int˙ZF˙1˙2˙L8:
  assumes A1: a∈ℤ b∈ℤ
  shows a+1+(b+1) = b+a+2
  using assms int˙zero˙one˙are˙int Int˙ZF˙1˙T2 group0.group0˙4˙L8
  by simp
```

A couple of rearrangement with three integers.

```
lemma (in int0) Int˙ZF˙1˙2˙L9:
  assumes a∈ℤ  b∈ℤ  c∈ℤ
  shows
  (a-b)+(b-c) = a-c
  (a-b)-(a-c) = c-b
  a+(b+(c-a-b)) = c
  (-a)-b+c = c-a-b
  (-b)-a+c = c-a-b
  (-((-a)+b+c)) = a-b-c
  a+b+c-a = b+c
  a+b-(a+c) = b-c
  using assms Int˙ZF˙1˙T2
    group0.group0˙4˙L4B group0.group0˙4˙L6D group0.group0˙4˙L4D
    group0.group0˙4˙L6B group0.group0˙4˙L6E
```

```
  by auto
```

Another couple of rearrangements with three integers.

```
lemma (in int0) Int˙ZF˙1˙2˙L9A:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows (-(a-b-c)) = c+b-a
proof -
  from A1 have T:
    a-b ∈ ℤ  (-(a-b)) ∈ ℤ  (-b) ∈ ℤ using
    Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙1˙L5 by auto
  with A1 have (-(a-b-c)) = c - ((-b)+a)
     using Int˙ZF˙1˙1˙L5 by simp
   also from A1 T have ... = c+b-a
     using Int˙ZF˙1˙1˙L6 Int˙ZF˙1˙1˙L5B
     by simp
  finally show (-(a-b-c)) = c+b-a
    by simp
qed
```

Another rearrangement with three integers.

```
lemma (in int0) Int˙ZF˙1˙2˙L10:
  assumes A1: a∈ℤ b∈ℤ c∈ℤ
  shows (a+1)·b + (c+1)·b = (c+a+2)·b
proof -
  from A1 have a+1 ∈ ℤ c+1 ∈ ℤ
    using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L5 by auto
  with A1 have
    (a+1)·b + (c+1)·b = (a+1+(c+1))·b
    using Int˙ZF˙1˙1˙L1 by simp
  also from A1 have ... = (c+a+2)·b
    using Int˙ZF˙1˙2˙L8 by simp
  finally show thesis by simp
qed
```

A technical rearrangement involing inequalities with absolute value.

```
lemma (in int0) Int˙ZF˙1˙2˙L10A:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ  e∈ℤ
  and A2: abs(a·b-c) ≤ d  abs(b·a-e) ≤ f
  shows abs(c-e) ≤  f+d
proof -
  from A1 A2 have T1:
    d∈ℤ  f∈ℤ  a·b ∈ ℤ  a·b-c ∈ ℤ  b·a-e ∈ ℤ
    using Int˙ZF˙2˙L1A Int˙ZF˙1˙1˙L5 by auto
  with A2 have
    abs((b·a-e)-(a·b-c)) ≤ f +d
    using Int˙ZF˙2˙L21 by simp
  with A1 T1 show abs(c-e) ≤ f+d
    using Int˙ZF˙1˙1˙L5 Int˙ZF˙1˙2˙L9 by simp
qed
```

Some arithmetics.

```
lemma (in int0) Int˙ZF˙1˙2˙L11: assumes A1: a∈ℤ
  shows
  a+1+2 = a+3
  a = 2·a - a
proof -
  from A1 show a+1+2 = a+3
    using int˙zero˙one˙are˙int int˙two˙three˙are˙int Int˙ZF˙1˙T2 group0.group0˙4˙L4C
    by simp
  from A1 show a = 2·a - a
   using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L1 Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙T2 group0.inv˙cancel˙two
    by simp
qed
```

A simple rearrangement with three integers.

```
lemma (in int0) Int˙ZF˙1˙2˙L12:
  assumes a∈ℤ  b∈ℤ  c∈ℤ
  shows
  (b-c)·a = a·b - a·c
  using assms Int˙ZF˙1˙1˙L6 Int˙ZF˙1˙1˙L5 by simp
```

A big rearrangement with five integers.

```
lemma (in int0) Int˙ZF˙1˙2˙L13:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ d∈ℤ  x∈ℤ
  shows (x+(a·x+b)+c)·d = d·(a+1)·x + (b·d+c·d)
proof -
  from A1 have T1:
    a·x ∈ ℤ   (a+1)·x ∈ ℤ
    (a+1)·x + b ∈ ℤ
    using Int˙ZF˙1˙1˙L5 int˙zero˙one˙are˙int by auto
  with A1 have (x+(a·x+b)+c)·d = ((a+1)·x + b)·d + c·d
    using Int˙ZF˙1˙1˙L7 Int˙ZF˙1˙2˙L7 Int˙ZF˙1˙1˙L1
    by simp
  also from A1 T1 have ... = (a+1)·x·d + b · d + c·d
    using Int˙ZF˙1˙1˙L1 by simp
  finally have (x+(a·x+b)+c)·d = (a+1)·x·d + b·d + c·d
    by simp
  moreover from A1 T1 have (a+1)·x·d = d·(a+1)·x
    using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L5 Int˙ZF˙1˙1˙L7 by simp
  ultimately have (x+(a·x+b)+c)·d = d·(a+1)·x + b·d + c·d
    by simp
  moreover from A1 T1 have
    d·(a+1)·x ∈ ℤ  b·d ∈ ℤ  c·d ∈ ℤ
    using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L5 by auto
  ultimately show thesis using Int˙ZF˙1˙1˙L7 by simp
qed
```

Rerrangement about adding linear functions.

```
lemma (in int0) Int˙ZF˙1˙2˙L14:
```

```
  assumes a∈ℤ  b∈ℤ  c∈ℤ d∈ℤ  x∈ℤ
  shows (a·x + b) + (c·x + d) = (a+c)·x + (b+d)
  using assms Int˙ZF˙1˙1˙L2 ring0.Ring˙ZF˙2˙L3 by simp
```

A rearrangement with four integers. Again we have to use the generic set notation to use a theorem proven in different context.

```
lemma (in int0) Int˙ZF˙1˙2˙L15: assumes A1: a∈ℤ  b∈ℤ  c∈ℤ d∈ℤ
  and A2: a = b-c-d
  shows
  d = b-a-c
  d = (-a)+b-c
  b = a+d+c
proof -
  let G = int
  let f = IntegerAddition
  from A1 A2 have I:
    group0(G, f)   f –is commutative on″ G
    a ∈ G  b ∈ G c ∈ G  d ∈ G
    a = f⟨f⟨b,GroupInv(G, f)(c)⟩,GroupInv(G, f)(d)⟩
    using Int˙ZF˙1˙T2 by auto
  then have
    d = f⟨f⟨b,GroupInv(G, f)(a)⟩,GroupInv(G,f)(c)⟩
    by (rule group0.group0˙4˙L9)
  then show d = b-a-c by simp
  from I have d = f⟨f⟨GroupInv(G, f)(a),b⟩, GroupInv(G, f)(c)⟩
    by (rule group0.group0˙4˙L9)
  thus d = (-a)+b-c
    by simp
  from I have b = f⟨f⟨a, d⟩,c⟩
    by (rule group0.group0˙4˙L9)
  thus b = a+d+c by simp
qed
```

A rearrangement with four integers. Property of groups.

```
lemma (in int0) Int˙ZF˙1˙2˙L16:
  assumes a∈ℤ  b∈ℤ  c∈ℤ d∈ℤ
  shows a+(b-c)+d = a+b+d-c
  using assms Int˙ZF˙1˙T2 group0.group0˙4˙L8 by simp
```

Some rearrangements with three integers. Properties of groups.

```
lemma (in int0) Int˙ZF˙1˙2˙L17:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows
  a+b-c+(c-b) = a
  a+(b+c)-c = a+b
proof -
  let G = int
  let f = IntegerAddition
  from A1 have I:
```

```
    group0(G, f)
    a ∈ G  b ∈ G c ∈ G
    using Int˙ZF˙1˙T2 by auto
  then have
    f⟨f⟨f⟨a,b⟩,GroupInv(G, f)(c)⟩,f⟨c,GroupInv(G, f)(b)⟩⟩ = a
    by (rule group0.group0˙2˙L14A)
  thus a+b-c+(c-b) = a by simp
  from I have
    f⟨f⟨a,f⟨b,c⟩⟩,GroupInv(G, f)(c)⟩ = f⟨a,b⟩
    by (rule group0.group0˙2˙L14A)
  thus a+(b+c)-c = a+b by simp
qed
```

Another rearrangement with three integers. Property of abelian groups.

```
lemma (in int0) Int˙ZF˙1˙2˙L18:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows a+b-c+(c-a) = b
proof -
  let G = int
  let f = IntegerAddition
  from A1 have
    group0(G, f)   f –is commutative on″ G
    a ∈ G  b ∈ G c ∈ G
    using Int˙ZF˙1˙T2 by auto
  then have
    f⟨f⟨f⟨a,b⟩,GroupInv(G, f)(c)⟩,f⟨c,GroupInv(G, f)(a)⟩⟩ = b
    by (rule group0.group0˙4˙L6D)
  thus a+b-c+(c-a) = b by simp
qed
```

43.3 Integers as an ordered ring

We already know from Int˙ZF that integers with addition form a linearly ordered group. To show that integers form an ordered ring we need the fact that the set of nonnegative integers is closed under multiplication.

We start with the property that a product of nonnegative integers is nonnegative. The proof is by induction and the next lemma is the induction step.

```
lemma (in int0) Int˙ZF˙1˙3˙L1: assumes A1: 0≤a  0≤b
  and A3: 0 ≤ a·b
  shows 0 ≤ a·(b+1)
proof -
  from A1 A3 have 0+0 ≤ a·b+a
    using int˙ineq˙add˙sides by simp
  with A1 show 0 ≤ a·(b+1)
    using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L4 Int˙ZF˙2˙L1A Int˙ZF˙1˙2˙L7
    by simp
```

qed

Product of nonnegative integers is nonnegative.

```
lemma (in int0) Int˙ZF˙1˙3˙L2: assumes A1: 0≤a  0≤b
  shows 0≤a·b
proof -
  from A1 have 0≤b by simp
  moreover from A1 have 0 ≤ a·0 using
    Int˙ZF˙2˙L1A Int˙ZF˙1˙1˙L4 int˙zero˙one˙are˙int int˙ord˙is˙refl refl˙def
    by simp
  moreover from A1 have
    ∀m. 0≤m ∧ 0≤a·m ⟶ 0 ≤ a·(m+1)
    using Int˙ZF˙1˙3˙L1 by simp
  ultimately show 0≤a·b by (rule Induction˙on˙int)
qed
```

The set of nonnegative integers is closed under multiplication.

```
lemma (in int0) Int˙ZF˙1˙3˙L2A: shows
  ℤ⁺ –is closed under˝ IntegerMultiplication
proof -
  { fix a b assume a∈ℤ⁺  b∈ℤ⁺
    then have a·b ∈ℤ⁺
      using Int˙ZF˙1˙3˙L2 Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L2
      by simp
  } then have ∀a∈ℤ⁺.∀b∈ℤ⁺.a·b ∈ℤ⁺ by simp
  then show thesis using IsOpClosed˙def by simp
qed
```

Integers form an ordered ring. All theorems proven in the ring1 context are valid in int0 context.

```
theorem (in int0) Int˙ZF˙1˙3˙T1: shows
  IsAnOrdRing(ℤ,IntegerAddition,IntegerMultiplication,IntegerOrder)
  ring1(ℤ,IntegerAddition,IntegerMultiplication,IntegerOrder)
  using Int˙ZF˙1˙1˙L2 Int˙ZF˙2˙L1B Int˙ZF˙1˙3˙L2A Int˙ZF˙2˙T1
    OrdRing˙ZF˙1˙L6 OrdRing˙ZF˙1˙L2 by auto
```

Product of integers that are greater that one is greater than one. The proof is by induction and the next step is the induction step.

```
lemma (in int0) Int˙ZF˙1˙3˙L3˙indstep:
  assumes A1: 1≤a  1≤b
  and A2: 1 ≤ a·b
  shows 1 ≤ a·(b+1)
proof -
   from A1 A2 have 1≤2 and 2 ≤ a·(b+1)
    using Int˙ZF˙2˙L1A int˙ineq˙add˙sides Int˙ZF˙2˙L16B Int˙ZF˙1˙2˙L7
    by auto
  then show 1 ≤ a·(b+1) by (rule Int˙order˙transitive)
qed
```

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

```
lemma (in int0) Int˙ZF˙1˙3˙L3:
  assumes A1: 1≤a 1≤b
  shows 1 ≤ a·b
proof -
  from A1 have 1≤b  1≤a·1
    using Int˙ZF˙2˙L1A Int˙ZF˙1˙1˙L4 by auto
  moreover from A1 have
    ∀m. 1≤m ∧ 1 ≤ a·m ⟶ 1 ≤ a·(m+1)
    using Int˙ZF˙1˙3˙L3˙indstep by simp
  ultimately show 1 ≤ a·b by (rule Induction˙on˙int)
qed
```

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

```
lemma (in int0) Int˙ZF˙1˙3˙L4: assumes a∈ℤ  b∈ℤ
  shows
  abs((-a)·b) = abs(a·b)
  abs(a·(-b)) = abs(a·b)
  abs((-a)·(-b)) = abs(a·b)
  using assms Int˙ZF˙1˙1˙L5 Int˙ZF˙2˙L17 by auto
```

Absolute value of a product is the product of absolute values. Property of ordered rings.

```
lemma (in int0) Int˙ZF˙1˙3˙L5:
  assumes A1: a∈ℤ  b∈ℤ
  shows abs(a·b) = abs(a)·abs(b)
  using assms Int˙ZF˙1˙3˙T1 ring1.OrdRing˙ZF˙2˙L5 by simp
```

Double nonnegative is nonnegative. Property of ordered rings.

```
lemma (in int0) Int˙ZF˙1˙3˙L5A: assumes 0≤a
  shows 0≤2·a
  using assms Int˙ZF˙1˙3˙T1 ring1.OrdRing˙ZF˙1˙L5A by simp
```

The next lemma shows what happens when one integer is not greater or equal than another.

```
lemma (in int0) Int˙ZF˙1˙3˙L6:
  assumes A1: a∈ℤ  b∈ℤ
  shows ¬(b≤a) ⟷ a+1 ≤ b
proof
  assume A3: ¬(b≤a)
  with A1 have  a≤b by (rule Int˙ZF˙2˙L19)
  then have a = b   ∨  a+1 ≤ b
    using Int˙ZF˙4˙L1B by simp
  moreover from A1 A3 have a≠b by (rule Int˙ZF˙2˙L19)
  ultimately show a+1 ≤ b by simp
next assume A4: a+1 ≤ b
  { assume b≤a
```

```
    with A4 have a+1 ≤ a by (rule Int˙order˙transitive)
    moreover from A1 have a ≤ a+1
      using Int˙ZF˙2˙L12B by simp
    ultimately have a+1 = a
      by (rule Int˙ZF˙2˙L3)
    with A1 have False using Int˙ZF˙1˙L14 by simp
  } then show ¬(b≤a) by auto
qed
```

Another form of stating that there are no integers between integers m and $m+1$.

```
corollary (in int0) no˙int˙between: assumes A1: a∈ℤ  b∈ℤ
  shows b≤a ∨ a+1 ≤ b
  using A1 Int˙ZF˙1˙3˙L6 by auto
```

Another way of saying what it means that one integer is not greater or equal than another.

```
corollary (in int0) Int˙ZF˙1˙3˙L6A:
  assumes A1: a∈ℤ  b∈ℤ and A2: ¬(b≤a)
  shows a ≤ b-1
proof -
  from A1 A2 have a+1 - 1 ≤ b - 1
    using Int˙ZF˙1˙3˙L6 int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L4
      int˙ord˙transl˙inv by simp
  with A1 show a ≤ b-1
    using int˙zero˙one˙are˙int Int˙ZF˙1˙2˙L3
    by simp
qed
```

Yet another form of stating that there are nointegers between m and $m+1$.

```
lemma (in int0) no˙int˙between1:
  assumes A1: a≤b  and A2: a≠b
  shows
  a+1 ≤ b
  a ≤ b-1
proof -
  from A1 have T: a∈ℤ  b∈ℤ using Int˙ZF˙2˙L1A
    by auto
  { assume b≤a
    with A1 have a=b by (rule Int˙ZF˙2˙L3)
    with A2 have False by simp }
  then have ¬(b≤a) by auto
  with T show
    a+1 ≤ b
    a ≤ b-1
    using no˙int˙between Int˙ZF˙1˙3˙L6A by auto
qed
```

We can decompose proofs into three cases: $a = b$, $a \leq b - 1b$ or $a \geq b + 1b$.

```
lemma (in int0) Int˙ZF˙1˙3˙L6B: assumes A1: a∈ℤ  b∈ℤ
  shows a=b ∨ (a ≤ b-1) ∨ (b+1 ≤a)
proof -
  from A1 have a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
    using Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L31
    by simp
  then show thesis using no˙int˙between1
    by auto
qed
```

A special case of Int˙ZF˙1˙3˙L6B when $b = 0$. This allows to split the proofs in cases $a \leq -1$, $a = 0$ and $a \geq 1$.

```
corollary (in int0) Int˙ZF˙1˙3˙L6C: assumes A1: a∈ℤ
  shows a=0 ∨ (a ≤ -1) ∨ (1≤a)
proof -
  from A1 have a=0 ∨ (a ≤ 0 -1) ∨ (0 +1 ≤a)
    using int˙zero˙one˙are˙int Int˙ZF˙1˙3˙L6B by simp
  then show thesis using Int˙ZF˙1˙1˙L4 int˙zero˙one˙are˙int
    by simp
qed
```

An integer is not less or equal zero iff it is greater or equal one.

```
lemma (in int0) Int˙ZF˙1˙3˙L7: assumes a∈ℤ
  shows ¬(a≤0) ⟷ 1 ≤ a
  using assms int˙zero˙one˙are˙int Int˙ZF˙1˙3˙L6 Int˙ZF˙1˙1˙L4
  by simp
```

Product of positive integers is positive.

```
lemma (in int0) Int˙ZF˙1˙3˙L8:
  assumes a∈ℤ  b∈ℤ
  and ¬(a≤0)  ¬(b≤0)
  shows ¬((a·b) ≤ 0)
  using assms Int˙ZF˙1˙3˙L7 Int˙ZF˙1˙3˙L3 Int˙ZF˙1˙1˙L5 Int˙ZF˙1˙3˙L7
  by simp
```

If $a \cdot b$ is nonnegative and b is positive, then a is nonnegative. Proof by contradiction.

```
lemma (in int0) Int˙ZF˙1˙3˙L9:
  assumes A1: a∈ℤ  b∈ℤ
  and A2:  ¬(b≤0) and A3: a·b ≤ 0
  shows a≤0
proof -
  { assume ¬(a≤0)
    with A1 A2 have ¬((a·b) ≤ 0) using Int˙ZF˙1˙3˙L8
      by simp
  } with A3 show a≤0 by auto
qed
```

One integer is less or equal another iff the difference is nonpositive.

```
lemma (in int0) Int˙ZF˙1˙3˙L10:
  assumes a∈ℤ  b∈ℤ
  shows a≤b ⟷ a-b ≤ 𝟬
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L9
  by simp
```

Some conclusions from the fact that one integer is less or equal than another.

```
lemma (in int0) Int˙ZF˙1˙3˙L10A: assumes a≤b
  shows 𝟬 ≤ b-a
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L12A
  by simp
```

We can simplify out a positive element on both sides of an inequality.

```
lemma (in int0) Int˙ineq˙simpl˙positive:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  and A2: a·c ≤ b·c and A4: ¬(c≤𝟬)
  shows a ≤ b
proof -
  from A1 A4 have a-b ∈  ℤ  c∈ℤ  ¬(c≤𝟬)
    using Int˙ZF˙1˙1˙L5 by auto
  moreover from A1 A2 have (a-b)·c ≤ 𝟬
    using Int˙ZF˙1˙1˙L5 Int˙ZF˙1˙3˙L10 Int˙ZF˙1˙1˙L6
    by simp
  ultimately have a-b ≤ 𝟬 by (rule Int˙ZF˙1˙3˙L9)
  with A1 show a ≤ b using Int˙ZF˙1˙3˙L10 by simp
qed
```

A technical lemma about conclusion from an inequality between absolute values. This is a property of ordered rings.

```
lemma (in int0) Int˙ZF˙1˙3˙L11:
  assumes A1: a∈ℤ  b∈ℤ
  and A2: ¬(abs(a) ≤ abs(b))
  shows ¬(abs(a) ≤ 𝟬)
proof -
  { assume abs(a) ≤ 𝟬
    moreover from A1 have 𝟬 ≤ abs(a) using int˙abs˙nonneg
      by simp
    ultimately have abs(a) = 𝟬 by (rule Int˙ZF˙2˙L3)
    with A1 A2 have False using int˙abs˙nonneg by simp
  } then show  ¬(abs(a) ≤ 𝟬) by auto
qed
```

Negative times positive is negative. This a property of ordered rings.

```
lemma (in int0) Int˙ZF˙1˙3˙L12:
  assumes a≤𝟬  and 𝟬≤b
  shows a·b ≤ 𝟬
  using assms Int˙ZF˙1˙3˙T1 ring1.OrdRing˙ZF˙1˙L8
  by simp
```

We can multiply an inequality by a nonnegative number. This is a property of ordered rings.

```
lemma (in int0) Int˙ZF˙1˙3˙L13:
  assumes A1: a≤b and A2: 0≤c
  shows
  a·c ≤ b·c
  c·a ≤ c·b
  using assms Int˙ZF˙1˙3˙T1 ring1.OrdRing˙ZF˙1˙L9 by auto
```

A technical lemma about decreasing a factor in an inequality.

```
lemma (in int0) Int˙ZF˙1˙3˙L13A:
  assumes 1≤a and b≤c and (a+1)·c ≤ d
  shows (a+1)·b ≤ d
proof -
  from assms have
    (a+1)·b ≤ (a+1)·c
    (a+1)·c ≤ d
    using Int˙ZF˙2˙L16C Int˙ZF˙1˙3˙L13 by auto
  then show (a+1)·b ≤ d by (rule Int˙order˙transitive)
qed
```

We can multiply an inequality by a positive number. This is a property of ordered rings.

```
lemma (in int0) Int˙ZF˙1˙3˙L13B:
  assumes A1: a≤b and A2: c∈ℤ₊
  shows
  a·c ≤ b·c
  c·a ≤ c·b
proof -
  let R = ℤ
  let A = IntegerAddition
  let M = IntegerMultiplication
  let r = IntegerOrder
  from A1 A2 have
    ring1(R, A, M, r)
    ⟨a,b⟩ ∈ r
    c ∈ PositiveSet(R, A, r)
    using Int˙ZF˙1˙3˙T1 by auto
  then show
    a·c ≤ b·c
    c·a ≤ c·b
    using ring1.OrdRing˙ZF˙1˙L9A by auto
qed
```

A rearrangement with four integers and absolute value.

```
lemma (in int0) Int˙ZF˙1˙3˙L14:
  assumes  A1: a∈ℤ  b∈ℤ  c∈ℤ  d∈ℤ
  shows abs(a·b)+(abs(a)+c)·d = (d+abs(b))·abs(a)+c·d
```

```
proof -
  from A1 have T1:
    abs(a) ∈ ℤ  abs(b) ∈ ℤ
    abs(a)·abs(b) ∈ ℤ
    abs(a)·d ∈ ℤ
    c·d ∈ ℤ
    abs(b)+d ∈ ℤ
    using Int˙ZF˙2˙L14 Int˙ZF˙1˙1˙L5 by auto
  with A1 have abs(a·b)+(abs(a)+c)·d = abs(a)·(abs(b)+d)+c·d
    using Int˙ZF˙1˙3˙L5 Int˙ZF˙1˙1˙L1 Int˙ZF˙1˙1˙L7 by simp
  with A1 T1 show thesis using Int˙ZF˙1˙1˙L5 by simp
qed
```

A technical lemma about what happens when one absolute value is not greater or equal than another.

```
lemma (in int0) Int˙ZF˙1˙3˙L15: assumes A1: m∈ℤ n∈ℤ
  and A2: ¬(abs(m) ≤ abs(n))
  shows n ≤ abs(m)  m≠0
proof -
  from A1 have T1: n ≤ abs(n)
    using Int˙ZF˙2˙L19C by simp
  from A1 have abs(n) ∈ ℤ  abs(m) ∈ ℤ
    using Int˙ZF˙2˙L14 by auto
  moreover note A2
  ultimately have abs(n) ≤ abs(m)
    by (rule Int˙ZF˙2˙L19)
  with T1 show  n ≤ abs(m) by (rule Int˙order˙transitive)
  from A1 A2 show m≠0 using Int˙ZF˙2˙L18 int˙abs˙nonneg by auto
qed
```

Negative of a nonnegative is nonpositive.

```
lemma (in int0) Int˙ZF˙1˙3˙L16: assumes A1: 0 ≤ m
  shows (-m) ≤ 0
proof -
  from A1 have (-m) ≤ (-0)
    using Int˙ZF˙2˙L10 by simp
  then show (-m) ≤ 0 using Int˙ZF˙1˙L11
    by simp
qed
```

Some statements about intervals centered at 0.

```
lemma (in int0) Int˙ZF˙1˙3˙L17: assumes A1: m∈ℤ
  shows
  (-abs(m)) ≤ abs(m)
  (-abs(m))..abs(m) ≠ 0
proof -
  from A1 have (-abs(m)) ≤ 0  0 ≤ abs(m)
    using int˙abs˙nonneg Int˙ZF˙1˙3˙L16 by auto
  then show (-abs(m)) ≤ abs(m) by (rule Int˙order˙transitive)
```

```
  then have abs(m) ∈ (-abs(m))..abs(m)
    using int˙ord˙is˙refl Int˙ZF˙2˙L1A Order˙ZF˙2˙L2 by simp
  thus (-abs(m))..abs(m) ≠ 0 by auto
qed
```

The greater of two integers is indeed greater than both, and the smaller one is smaller that both.

```
lemma (in int0) Int˙ZF˙1˙3˙L18: assumes A1: m∈ℤ  n∈ℤ
  shows
  m ≤ GreaterOf(IntegerOrder,m,n)
  n ≤ GreaterOf(IntegerOrder,m,n)
  SmallerOf(IntegerOrder,m,n) ≤ m
  SmallerOf(IntegerOrder,m,n) ≤ n
  using assms Int˙ZF˙2˙T1 Order˙ZF˙3˙L2 by auto
```

If $|m| \leq n$, then $m \in -n..n$.

```
lemma (in int0) Int˙ZF˙1˙3˙L19:
  assumes A1: m∈ℤ and A2: abs(m) ≤ n
  shows
  (-n) ≤ m  m ≤ n
  m ∈ (-n)..n
  𝟬 ≤ n
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L8
    group3.OrderedGroup˙ZF˙3˙L8A Order˙ZF˙2˙L1
  by auto
```

A slight generalization of the above lemma.

```
lemma (in int0) Int˙ZF˙1˙3˙L19A:
  assumes A1: m∈ℤ and A2: abs(m) ≤ n and A3: 𝟬≤k
  shows (-(n+k)) ≤ m
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L8B
  by simp
```

Sets of integers that have absolute value bounded are bounded.

```
lemma (in int0) Int˙ZF˙1˙3˙L20:
  assumes A1: ∀x∈X. b(x) ∈ ℤ ∧ abs(b(x)) ≤ L
  shows IsBounded(–b(x). x∈X″,IntegerOrder)
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from A1 have
    group3(G, P, r)
    r –is total on″ G
    ∀x∈X. b(x) ∈ G ∧ ⟨AbsoluteValue(G, P, r)  b(x), L⟩ ∈ r
    using Int˙ZF˙2˙T1 by auto
  then show IsBounded(–b(x). x∈X″,IntegerOrder)
    by (rule group3.OrderedGroup˙ZF˙3˙L9A)
```

qed

If a set is bounded, then the absolute values of the elements of that set are bounded.

```
lemma (in int0) Int˙ZF˙1˙3˙L20A: assumes IsBounded(A,IntegerOrder)
  shows ∃L. ∀a∈A. abs(a) ≤ L
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L10A
  by simp
```

Absolute vaues of integers from a finite image of integers are bounded by an integer.

```
lemma (in int0) Int˙ZF˙1˙3˙L20AA:
  assumes A1: –b(x). x∈ℤ″ ∈ Fin(ℤ)
  shows ∃L∈ℤ. ∀x∈ℤ. abs(b(x)) ≤ L
  using assms int˙not˙empty Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L11A
  by simp
```

If absolute values of values of some integer function are bounded, then the image a set from the domain is a bounded set.

```
lemma (in int0) Int˙ZF˙1˙3˙L20B:
  assumes f:X→ℤ and A⊆X and ∀x∈A. abs(f(x)) ≤ L
  shows  IsBounded(f(A),IntegerOrder)
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from assms have
    group3(G, P, r)
    r –is total on″ G
    f:X→G
    A⊆X
    ∀x∈A. ⟨AbsoluteValue(G, P, r)(f(x)), L⟩ ∈ r
    using Int˙ZF˙2˙T1 by auto
  then show IsBounded(f(A), r)
    by (rule group3.OrderedGroup˙ZF˙3˙L9B)
qed
```

A special case of the previous lemma for a function from integers to integers.

```
corollary (in int0) Int˙ZF˙1˙3˙L20C:
  assumes f:ℤ→ℤ and ∀m∈ℤ. abs(f(m)) ≤ L
  shows f(ℤ) ∈ Fin(ℤ)
proof -
  from assms have f:ℤ→ℤ ℤ ⊆ ℤ  ∀m∈ℤ. abs(f(m)) ≤ L
    by auto
  then have IsBounded(f(ℤ),IntegerOrder)
    by (rule Int˙ZF˙1˙3˙L20B)
  then show f(ℤ) ∈ Fin(ℤ) using Int˙bounded˙iff˙fin
    by simp
```

qed

A triangle inequality with three integers. Property of linearly ordered abelian groups.

```
lemma (in int0) int˙triangle˙ineq3:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows abs(a-b-c) ≤ abs(a) + abs(b) + abs(c)
proof -
  from A1 have T: a-b ∈ ℤ  abs(c) ∈ ℤ
    using Int˙ZF˙1˙1˙L5 Int˙ZF˙2˙L14 by auto
  with A1 have abs(a-b-c) ≤ abs(a-b) + abs(c)
    using Int˙triangle˙ineq1 by simp
  moreover from A1 T have
    abs(a-b) + abs(c) ≤  abs(a) + abs(b) + abs(c)
    using Int˙triangle˙ineq1 int˙ord˙transl˙inv by simp
  ultimately show thesis by (rule Int˙order˙transitive)
qed
```

If $a \leq c$ and $b \leq c$, then $a + b \leq 2 \cdot c$. Property of ordered rings.

```
lemma (in int0) Int˙ZF˙1˙3˙L21:
  assumes A1: a≤c  b≤c shows a+b ≤ 2·c
  using assms Int˙ZF˙1˙3˙T1 ring1.OrdRing˙ZF˙2˙L6 by simp
```

If an integer a is between b and $b + c$, then $|b - a| \leq c$. Property of ordered groups.

```
lemma (in int0) Int˙ZF˙1˙3˙L22:
  assumes a≤b and c∈ℤ and b≤ c+a
  shows abs(b-a) ≤ c
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L8C
  by simp
```

An application of the triangle inequality with four integers. Property of linearly ordered abelian groups.

```
lemma (in int0) Int˙ZF˙1˙3˙L22A:
  assumes  a∈ℤ  b∈ℤ  c∈ℤ  d∈ℤ
  shows abs(a-c) ≤ abs(a+b) + abs(c+d) + abs(b-d)
  using assms Int˙ZF˙1˙T2 Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L7F
  by simp
```

If an integer a is between b and $b + c$, then $|b - a| \leq c$. Property of ordered groups. A version of Int˙ZF˙1˙3˙L22 with sligtly different assumptions.

```
lemma (in int0) Int˙ZF˙1˙3˙L23:
  assumes A1: a≤b and A2: c∈ℤ and A3: b≤ a+c
  shows abs(b-a) ≤ c
proof -
  from A1 have a ∈ ℤ
    using Int˙ZF˙2˙L1A by simp
  with A2 A3 have b≤ c+a
```

```
    using Int˙ZF˙1˙1˙L5 by simp
  with A1 A2 show abs(b-a) ≤ c
    using Int˙ZF˙1˙3˙L22 by simp
qed
```

43.4 Maximum and minimum of a set of integers

In this section we provide some sufficient conditions for integer subsets to have extrema (maxima and minima).

Finite nonempty subsets of integers attain maxima and minima.

```
theorem (in int0) Int˙fin˙have˙max˙min:
  assumes A1: A ∈ Fin(ℤ) and A2: A≠0
  shows
  HasAmaximum(IntegerOrder,A)
  HasAminimum(IntegerOrder,A)
  Maximum(IntegerOrder,A) ∈ A
  Minimum(IntegerOrder,A) ∈ A
  ∀x∈A. x ≤ Maximum(IntegerOrder,A)
  ∀x∈A. Minimum(IntegerOrder,A) ≤ x
  Maximum(IntegerOrder,A) ∈ ℤ
  Minimum(IntegerOrder,A) ∈ ℤ
proof -
  from A1 have
    A=0 ∨ HasAmaximum(IntegerOrder,A) and
    A=0 ∨ HasAminimum(IntegerOrder,A)
    using Int˙ZF˙2˙T1 Int˙ZF˙2˙L6 Finite˙ZF˙1˙1˙T1A Finite˙ZF˙1˙1˙T1B
    by auto
  with A2 show
    HasAmaximum(IntegerOrder,A)
    HasAminimum(IntegerOrder,A)
    by auto
  from A1 A2 show
    Maximum(IntegerOrder,A) ∈ A
    Minimum(IntegerOrder,A) ∈ A
    ∀x∈A. x ≤ Maximum(IntegerOrder,A)
    ∀x∈A. Minimum(IntegerOrder,A) ≤ x
    using Int˙ZF˙2˙T1 Finite˙ZF˙1˙T2 by auto
  moreover from A1 have A⊆ℤ using FinD by simp
  ultimately show
    Maximum(IntegerOrder,A) ∈ ℤ
    Minimum(IntegerOrder,A) ∈ ℤ
    by auto
qed
```

Bounded nonempty integer subsets attain maximum and minimum.

```
theorem (in int0) Int˙bounded˙have˙max˙min:
  assumes IsBounded(A,IntegerOrder) and A≠0
  shows
```

```
  HasAmaximum(IntegerOrder,A)
  HasAminimum(IntegerOrder,A)
  Maximum(IntegerOrder,A) ∈ A
  Minimum(IntegerOrder,A) ∈ A
  ∀x∈A. x ≤ Maximum(IntegerOrder,A)
  ∀x∈A. Minimum(IntegerOrder,A) ≤ x
  Maximum(IntegerOrder,A) ∈ ℤ
  Minimum(IntegerOrder,A) ∈ ℤ
  using assms Int˙fin˙have˙max˙min Int˙bounded˙iff˙fin
  by auto
```

Nonempty set of integers that is bounded below attains its minimum.

```
theorem (in int0) int˙bounded˙below˙has˙min:
  assumes A1: IsBoundedBelow(A,IntegerOrder) and A2: A≠0
  shows
  HasAminimum(IntegerOrder,A)
  Minimum(IntegerOrder,A) ∈ A

  ∀x∈A. Minimum(IntegerOrder,A) ≤ x
proof -
  from A1 A2 have
    IntegerOrder –is total on″ ℤ
    trans(IntegerOrder)
    IntegerOrder ⊆ ℤ×ℤ
    ∀A. IsBounded(A,IntegerOrder) ∧ A≠0 ⟶ HasAminimum(IntegerOrder,A)
    A≠0  IsBoundedBelow(A,IntegerOrder)
    using Int˙ZF˙2˙T1 Int˙ZF˙2˙L6 Int˙ZF˙2˙L1B Int˙bounded˙have˙max˙min
    by auto
  then show HasAminimum(IntegerOrder,A)
    by (rule Order˙ZF˙4˙L11)
  then show
    Minimum(IntegerOrder,A) ∈ A
    ∀x∈A. Minimum(IntegerOrder,A) ≤ x
    using Int˙ZF˙2˙L4 Order˙ZF˙4˙L4 by auto
qed
```

Nonempty set of integers that is bounded above attains its maximum.

```
theorem (in int0) int˙bounded˙above˙has˙max:
  assumes A1: IsBoundedAbove(A,IntegerOrder) and A2: A≠0
  shows
  HasAmaximum(IntegerOrder,A)
  Maximum(IntegerOrder,A) ∈ A
  Maximum(IntegerOrder,A) ∈ ℤ
  ∀x∈A. x ≤ Maximum(IntegerOrder,A)
proof -
  from A1 A2 have
    IntegerOrder –is total on″ ℤ
    trans(IntegerOrder) and
    I: IntegerOrder ⊆ ℤ×ℤ and
```

```
    ∀A. IsBounded(A,IntegerOrder) ∧ A≠0 ⟶ HasAmaximum(IntegerOrder,A)
    A≠0  IsBoundedAbove(A,IntegerOrder)
    using Int˙ZF˙2˙T1 Int˙ZF˙2˙L6 Int˙ZF˙2˙L1B Int˙bounded˙have˙max˙min
    by auto
  then show HasAmaximum(IntegerOrder,A)
    by (rule Order˙ZF˙4˙L11A)
  then show
    II: Maximum(IntegerOrder,A) ∈ A and
    ∀x∈A. x ≤ Maximum(IntegerOrder,A)
    using Int˙ZF˙2˙L4 Order˙ZF˙4˙L3 by auto
  from I A1 have A ⊆ ℤ by (rule Order˙ZF˙3˙L1A)
  with II show Maximum(IntegerOrder,A) ∈ ℤ by auto
qed
```

A set defined by separation over a bounded set attains its maximum and minimum.

```
lemma (in int0) Int˙ZF˙1˙4˙L1:
  assumes A1: IsBounded(A,IntegerOrder) and A2: A≠0
  and A3: ∀q∈ℤ. F(q) ∈ ℤ
  and A4: K = −F(q). q ∈ A″
  shows
  HasAmaximum(IntegerOrder,K)
  HasAminimum(IntegerOrder,K)
  Maximum(IntegerOrder,K) ∈ K
  Minimum(IntegerOrder,K) ∈ K
  Maximum(IntegerOrder,K) ∈ ℤ
  Minimum(IntegerOrder,K) ∈ ℤ
  ∀q∈A. F(q) ≤ Maximum(IntegerOrder,K)
  ∀q∈A. Minimum(IntegerOrder,K) ≤ F(q)
  IsBounded(K,IntegerOrder)
proof -
  from A1 have A ∈ Fin(ℤ) using Int˙bounded˙iff˙fin
    by simp
  with A3 have −F(q). q ∈ A″ ∈ Fin(ℤ)
    by (rule fin˙image˙fin)
  with A2 A4 have T1: K ∈ Fin(ℤ)  K≠0 by auto
  then show T2:
    HasAmaximum(IntegerOrder,K)
    HasAminimum(IntegerOrder,K)
    and Maximum(IntegerOrder,K) ∈ K
    Minimum(IntegerOrder,K) ∈ K
    Maximum(IntegerOrder,K) ∈ ℤ
    Minimum(IntegerOrder,K) ∈ ℤ
    using Int˙fin˙have˙max˙min by auto
  { fix q assume q∈A
    with A4 have F(q) ∈ K by auto
    with T1 have
      F(q) ≤ Maximum(IntegerOrder,K)
      Minimum(IntegerOrder,K) ≤ F(q)
```

```
    using Int˙fin˙have˙max˙min by auto
  } then show
    ∀q∈A. F(q) ≤ Maximum(IntegerOrder,K)
    ∀q∈A. Minimum(IntegerOrder,K) ≤ F(q)
    by auto
  from T2 show IsBounded(K,IntegerOrder)
    using Order˙ZF˙4˙L7 Order˙ZF˙4˙L8A IsBounded˙def
    by simp
qed
```

A three element set has a maximume and minimum.

```
lemma (in int0) Int˙ZF˙1˙4˙L1A: assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows
  Maximum(IntegerOrder,–a,b,c″) ∈  ℤ
  a ≤ Maximum(IntegerOrder,–a,b,c″)
  b ≤ Maximum(IntegerOrder,–a,b,c″)
  c ≤ Maximum(IntegerOrder,–a,b,c″)
  using assms Int˙ZF˙2˙T1 Finite˙ZF˙1˙L2A by auto
```

Integer functions attain maxima and minima over intervals.

```
lemma (in int0) Int˙ZF˙1˙4˙L2:
  assumes A1: f:ℤ→ℤ and A2: a≤b
  shows
  maxf(f,a..b) ∈ ℤ
  ∀c ∈ a..b. f(c) ≤ maxf(f,a..b)
  ∃c ∈ a..b. f(c) = maxf(f,a..b)
  minf(f,a..b) ∈ ℤ
  ∀c ∈ a..b. minf(f,a..b) ≤ f(c)
  ∃c ∈ a..b. f(c) = minf(f,a..b)
proof -
  from A2 have T: a∈ℤ  b∈ℤ  a..b ⊆ ℤ
    using Int˙ZF˙2˙L1A Int˙ZF˙2˙L1B Order˙ZF˙2˙L6
    by auto
  with A1 A2 have
    Maximum(IntegerOrder,f(a..b)) ∈ f(a..b)
    ∀x∈f(a..b). x ≤ Maximum(IntegerOrder,f(a..b))
    Maximum(IntegerOrder,f(a..b)) ∈ ℤ
    Minimum(IntegerOrder,f(a..b)) ∈ f(a..b)
    ∀x∈f(a..b). Minimum(IntegerOrder,f(a..b)) ≤ x
    Minimum(IntegerOrder,f(a..b)) ∈ ℤ
    using Int˙ZF˙4˙L8 Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙2˙L6
      Int˙fin˙have˙max˙min by auto
  with A1 T show
    maxf(f,a..b) ∈ ℤ
    ∀c ∈ a..b. f(c) ≤ maxf(f,a..b)
    ∃c ∈ a..b. f(c) = maxf(f,a..b)
    minf(f,a..b) ∈ ℤ
    ∀c ∈ a..b. minf(f,a..b) ≤ f(c)
    ∃c ∈ a..b. f(c) = minf(f,a..b)
```

using func˙imagedef by auto
qed

43.5 The set of nonnegative integers

The set of nonnegative integers looks like the set of natural numbers. We explore that in this section. We also rephrase some lemmas about the set of positive integers known from the theory of oredered grups.

The set of positive integers is closed under addition.

lemma (in int0) pos˙int˙closed˙add:
shows $\mathbb{Z}_+$ –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∈$\mathbb{Z}_+$ b∈$\mathbb{Z}_+$ shows a+b ∈ $\mathbb{Z}_+$
using assms pos˙int˙closed˙add IsOpClosed˙def
by simp

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

lemma (in int0) Int˙ZF˙1˙5˙L1: shows
IsBoundedBelow($\mathbb{Z}^+$,IntegerOrder)
IsBoundedBelow($\mathbb{Z}_+$,IntegerOrder)
using Nonnegative˙def PositiveSet˙def IsBoundedBelow˙def by auto

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

lemma (in int0) Int˙ZF˙1˙5˙L1A: assumes A ⊆ $\mathbb{Z}^+$
shows IsBoundedBelow(A,IntegerOrder)
using assms Int˙ZF˙1˙5˙L1 Order˙ZF˙3˙L12 by blast

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

lemma (in int0) Int˙ZF˙1˙5˙L1B: assumes A1: A ⊆ $\mathbb{Z}_+$
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 ⊆ $\mathbb{Z}_+$ and A ≠ 0
shows
HasAminimum(IntegerOrder,A)
Minimum(IntegerOrder,A) ∈ A
∀x∈A. Minimum(IntegerOrder,A) ≤ x
using assms Int˙ZF˙1˙5˙L1B int˙bounded˙below˙has˙min by auto

Infinite subsets of Z^+ do not have a maximum - If $A \subseteq Z^+$ then for every integer we can find one in the set that is not smaller.

```
lemma (in int0) Int˙ZF˙1˙5˙L2:
  assumes A1: A ⊆ ℤ⁺  and A2: A ∉ Fin(ℤ) and A3: D∈ℤ
  shows ∃n∈A. D≤n
proof -
  { assume ∀n∈A. ¬(D≤n)
    moreover from A1 A3 have D∈ℤ  ∀n∈A. n∈ℤ
      using Nonnegative˙def by auto
    ultimately have ∀n∈A. n≤D
      using Int˙ZF˙2˙L19 by blast
    hence ∀n∈A. ⟨n,D⟩ ∈ IntegerOrder by simp
    then have IsBoundedAbove(A,IntegerOrder)
      by (rule Order˙ZF˙3˙L10)
    with A1 have IsBounded(A,IntegerOrder)
      using Int˙ZF˙1˙5˙L1A IsBounded˙def by simp
    with A2 have False using Int˙bounded˙iff˙fin by auto
  } thus thesis by auto
qed
```

Infinite subsets of Z_+ do not have a maximum - If $A \subseteq Z_+$ then for every integer we can find one in the set that is not smaller. This is very similar to Int˙ZF˙1˙5˙L2, except we have $\mathbb{Z}_+$ instead of $\mathbb{Z}^+$ here.

```
lemma (in int0) Int˙ZF˙1˙5˙L2A:
  assumes A1: A ⊆ ℤ₊  and A2: A ∉ Fin(ℤ) and A3: D∈ℤ
  shows ∃n∈A. D≤n
proof -
{ assume ∀n∈A. ¬(D≤n)
    moreover from A1 A3 have D∈ℤ  ∀n∈A. n∈ℤ
      using PositiveSet˙def by auto
    ultimately have ∀n∈A. n≤D
      using Int˙ZF˙2˙L19 by blast
    hence ∀n∈A. ⟨n,D⟩ ∈ IntegerOrder by simp
    then have IsBoundedAbove(A,IntegerOrder)
      by (rule Order˙ZF˙3˙L10)
    with A1 have IsBounded(A,IntegerOrder)
      using Int˙ZF˙1˙5˙L1B IsBounded˙def by simp
    with A2 have False using Int˙bounded˙iff˙fin by auto
  } thus thesis by auto
qed
```

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

```
lemma (in int0) Int˙decomp: assumes m∈ℤ
  shows Exactly˙1˙of˙3˙holds (m=0,m∈ℤ₊,(-m)∈ℤ₊)
  using assms Int˙ZF˙2˙T1 group3.OrdGroup˙decomp
  by simp
```

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

```
lemma (in int0) int˙decomp˙cases: assumes m∈ℤ
  shows m=0 ∨ m∈ℤ₊ ∨ (-m) ∈ ℤ₊
```

```
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L14
  by simp
```

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

```
lemma (in int0) Int˙ZF˙1˙5˙L3: shows m∈ℤ₊ ⟷ 1≤m
proof
  assume m∈ℤ₊ then have 0≤m  m≠0
    using PositiveSet˙def by auto
  then have 0+1 ≤ m
    using Int˙ZF˙4˙L1B by auto
  then show 1≤m
    using int˙zero˙one˙are˙int Int˙ZF˙1˙T2 group0.group0˙2˙L2
    by simp
next assume 1≤m
  then have m∈ℤ  0≤m  m≠0
    using Int˙ZF˙2˙L1A Int˙ZF˙2˙L16C by auto
  then show m∈ℤ₊ using PositiveSet˙def by auto
qed
```

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

```
lemma (in int0) pos˙int˙closed˙mul˙unfold:
  assumes a∈ℤ₊  b∈ℤ₊
  shows a·b ∈ ℤ₊
  using assms Int˙ZF˙1˙5˙L3 Int˙ZF˙1˙3˙L3 by simp
```

The set of positive integers is closed under multiplication.

```
lemma (in int0) pos˙int˙closed˙mul: shows
  ℤ₊ –is closed under″ IntegerMultiplication
  using pos˙int˙closed˙mul˙unfold IsOpClosed˙def
  by simp
```

It is an overkill to prove that the ring of integers has no zero divisors this way, but why not?

```
lemma (in int0) int˙has˙no˙zero˙divs:
  shows HasNoZeroDivs(ℤ,IntegerAddition,IntegerMultiplication)
  using pos˙int˙closed˙mul Int˙ZF˙1˙3˙T1 ring1.OrdRing˙ZF˙3˙L3
  by simp
```

Nonnegative integers are positive ones plus zero.

```
lemma (in int0) Int˙ZF˙1˙5˙L3A: shows ℤ⁺ = ℤ₊ ∪ –0″
  using Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L24 by simp
```

We can make a function smaller than any constant on a given interval of positive integers by adding another constant.

```
lemma (in int0) Int˙ZF˙1˙5˙L4:
  assumes A1: f:ℤ→ℤ and A2: K∈ℤ N∈ℤ
```

```
  shows ∃C∈ℤ. ∀n∈ℤ+. K ≤ f(n) + C ⟶ N≤n
proof -
  from A2 have N≤1 ∨ 2≤N
    using int˙zero˙one˙are˙int no˙int˙between
    by simp
  moreover
  { assume A3: N≤1
    let C = 0
    have C ∈ ℤ using int˙zero˙one˙are˙int
      by simp
    moreover
    { fix n assume n∈ℤ+
      then have 1 ≤ n using Int˙ZF˙1˙5˙L3
 by simp
      with A3 have N≤n by (rule Int˙order˙transitive)
    } then have  ∀n∈ℤ+. K ≤ f(n) + C ⟶ N≤n
      by auto
    ultimately have ∃C∈ℤ. ∀n∈ℤ+. K ≤ f(n) + C ⟶ N≤n
      by auto }
  moreover
  { let C = K - 1 - maxf(f,1..(N-1))
    assume 2≤N
    then have 2-1 ≤ N-1
      using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L4 int˙ord˙transl˙inv
      by simp
    then have I: 1 ≤ N-1
      using int˙zero˙one˙are˙int Int˙ZF˙1˙2˙L3 by simp
    with A1 A2 have T:
      maxf(f,1..(N-1)) ∈ ℤ  K-1 ∈ ℤ  C ∈ ℤ
      using Int˙ZF˙1˙4˙L2 Int˙ZF˙1˙1˙L5 int˙zero˙one˙are˙int
      by auto
    moreover
    { fix n assume A4: n∈ℤ+
      { assume A5: K ≤ f(n) + C and ¬(N≤n)
 with A2 A4 have n ≤ N-1
   using PositiveSet˙def Int˙ZF˙1˙3˙L6A by simp
 with A4 have n ∈ 1..(N-1)
   using Int˙ZF˙1˙5˙L3 Interval˙def by auto
 with A1 I T have f(n)+C ≤ maxf(f,1..(N-1)) + C
   using Int˙ZF˙1˙4˙L2 int˙ord˙transl˙inv by simp
 with T have f(n)+C ≤ K-1
   using Int˙ZF˙1˙2˙L3 by simp
 with A5 have K ≤  K-1
   by (rule Int˙order˙transitive)
 with A2 have False using Int˙ZF˙1˙2˙L3AA by simp
      } then have K ≤ f(n) + C ⟶ N≤n
 by auto
    } then have ∀n∈ℤ+. K ≤ f(n) + C ⟶ N≤n
      by simp
```

```
    ultimately have ∃C∈ℤ. ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
      by auto }
  ultimately show thesis by auto
qed
```

Absolute value is identity on positive integers.

```
lemma (in int0) Int˙ZF˙1˙5˙L4A:
  assumes a∈ℤ₊ shows abs(a) = a
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙3˙L2B
  by simp
```

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

```
lemma (in int0) int˙one˙two˙are˙pos: shows 𝟙 ∈ ℤ₊  𝟚 ∈ ℤ₊
  using int˙zero˙one˙are˙int int˙ord˙is˙refl refl˙def Int˙ZF˙1˙5˙L3
  Int˙ZF˙2˙L16B by auto
```

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

```
lemma (in int0) Int˙ZF˙1˙5˙L5: assumes A1: f : ℤ→X
  shows f(ℤ₊) ≠ 0
proof -
  have ℤ₊ ⊆ ℤ using PositiveSet˙def by auto
  with A1 show f(ℤ₊) ≠ 0
    using int˙one˙two˙are˙pos func˙imagedef by auto
qed
```

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

```
lemma (in int0) Int˙ZF˙1˙5˙L6: assumes A1: n ∈ ℤ₊
  shows
  𝟘 ≤ n-𝟙
  𝟘 ∈ 𝟘..(n-𝟙)
  𝟘..(n-𝟙) ⊆ ℤ
proof -
  from A1 have 𝟙 ≤ n  (-𝟙) ∈ ℤ
    using Int˙ZF˙1˙5˙L3 int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L4
    by auto
  then have 𝟙-𝟙 ≤ n-𝟙
    using int˙ord˙transl˙inv by simp
  then show 𝟘 ≤ n-𝟙
    using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L4 by simp
  then show 𝟘 ∈ 𝟘..(n-𝟙)
    using int˙zero˙one˙are˙int int˙ord˙is˙refl refl˙def Order˙ZF˙2˙L1B
    by simp
  show 𝟘..(n-𝟙) ⊆ ℤ
    using Int˙ZF˙2˙L1B Order˙ZF˙2˙L6 by simp
qed
```

Intgers greater than one in $\mathbb{Z}_+$ belong to $\mathbb{Z}_+$. This is a property of ordered groups and follows from OrderedGroup˙ZF˙1˙L19, but Isabelle's simplifier has problems using that result directly, so we reprove it specifically for integers.

```
lemma (in int0) Int˙ZF˙1˙5˙L7:  assumes a ∈ ℤ+ and a≤b
  shows b ∈ ℤ+
proof-
  from assms have 1≤a  a≤b
    using Int˙ZF˙1˙5˙L3 by auto
  then have 1≤b by (rule Int˙order˙transitive)
  then show b ∈ ℤ+ using Int˙ZF˙1˙5˙L3 by simp
qed
```

Adding a positive integer increases integers.

```
lemma (in int0) Int˙ZF˙1˙5˙L7A: assumes a∈ℤ  b ∈ ℤ+
  shows a ≤ a+b  a ≠ a+b  a+b ∈ ℤ
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L22
  by auto
```

For any integer m the greater of m and 1 is a positive integer that is greater or equal than m. If we add 1 to it we get a positive integer that is strictly greater than m.

```
lemma (in int0) Int˙ZF˙1˙5˙L7B: assumes a∈ℤ
  shows
  a ≤ GreaterOf(IntegerOrder,1,a)
  GreaterOf(IntegerOrder,1,a) ∈ ℤ+
  GreaterOf(IntegerOrder,1,a) + 1 ∈ ℤ+
  a ≤ GreaterOf(IntegerOrder,1,a) + 1
  a ≠ GreaterOf(IntegerOrder,1,a) + 1
  using assms int˙zero˙not˙one Int˙ZF˙1˙3˙T1 ring1.OrdRing˙ZF˙3˙L12
  by auto
```

The opposite of an element of $\mathbb{Z}_+$ cannot belong to $\mathbb{Z}_+$.

```
lemma (in int0) Int˙ZF˙1˙5˙L8: assumes a ∈ ℤ+
  shows (-a) ∉ ℤ+
  using assms Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L20
  by simp
```

For every integer there is one in $\mathbb{Z}_+$ that is greater or equal.

```
lemma (in int0) Int˙ZF˙1˙5˙L9: assumes a∈ℤ
  shows ∃b∈ℤ+. a≤b
  using assms int˙not˙trivial Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙1˙L23
  by simp
```

A theorem about odd extensions. Recall from OrdereGroup˙ZF.thy that the odd extension of an integer function f defined on $\mathbb{Z}_+$ is the odd function on $\mathbb{Z}$ equal to f on $\mathbb{Z}_+$. First we show that the odd extension is defined on $\mathbb{Z}$.

```
lemma (in int0) Int˙ZF˙1˙5˙L10: assumes f : ℤ+→ℤ
  shows OddExtension(ℤ,IntegerAddition,IntegerOrder,f) : ℤ→ℤ
  using assms Int˙ZF˙2˙T1 group3.odd˙ext˙props by simp
```

On $\mathbb{Z}_+$, the odd extension of f is the same as f.

lemma (in int0) Int˙ZF˙1˙5˙L11: assumes f : $\mathbb{Z}_+ \rightarrow \mathbb{Z}$ and a $\in \mathbb{Z}_+$ and
g = OddExtension($\mathbb{Z}$,IntegerAddition,IntegerOrder,f)
shows g(a) = f(a)
using assms Int˙ZF˙2˙T1 group3.odd˙ext˙props by simp

On -$\mathbb{Z}_+$, the value of the odd extension of f is the negative of $f(-a)$.

lemma (in int0) Int˙ZF˙1˙5˙L12:
assumes f : $\mathbb{Z}_+ \rightarrow \mathbb{Z}$ and a $\in$ (-$\mathbb{Z}_+$) and
g = OddExtension($\mathbb{Z}$,IntegerAddition,IntegerOrder,f)
shows g(a) = -(f(-a))
using assms Int˙ZF˙2˙T1 group3.odd˙ext˙props by simp

Odd extensions are odd on $\mathbb{Z}$.

lemma (in int0) int˙oddext˙is˙odd:
assumes f : $\mathbb{Z}_+ \rightarrow \mathbb{Z}$ and a$\in\mathbb{Z}$ and
g = OddExtension($\mathbb{Z}$,IntegerAddition,IntegerOrder,f)
shows g(-a) = -(g(a))
using assms Int˙ZF˙2˙T1 group3.oddext˙is˙odd by simp

Alternative definition of an odd function.

lemma (in int0) Int˙ZF˙1˙5˙L13: assumes A1: f: $\mathbb{Z}\rightarrow\mathbb{Z}$ shows
($\forall$ a$\in\mathbb{Z}$. f(-a) = (-f(a))) $\longleftrightarrow$ ($\forall$ a$\in\mathbb{Z}$. (-(f(-a))) = f(a))
using assms Int˙ZF˙1˙T2 group0.group0˙6˙L2 by simp

Another way of expressing the fact that odd extensions are odd.

lemma (in int0) int˙oddext˙is˙odd˙alt:
assumes f : $\mathbb{Z}_+ \rightarrow \mathbb{Z}$ and a$\in\mathbb{Z}$ and
g = OddExtension($\mathbb{Z}$,IntegerAddition,IntegerOrder,f)
shows (-g(-a)) = g(a)
using assms Int˙ZF˙2˙T1 group3.oddext˙is˙odd˙alt by simp

43.6 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: $\mathbb{Z}\rightarrow\mathbb{Z}$ and
$\forall$ a$\in\mathbb{Z}$.$\exists$ b$\in\mathbb{Z}_+$.$\forall$ x. b$\leq$x $\longrightarrow$ a $\leq$ f(x) and A $\subseteq \mathbb{Z}$ and
IsBoundedAbove(f(A),IntegerOrder)
shows IsBoundedAbove(A,IntegerOrder)

```
  using assms int˙not˙trivial Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙7˙L1
  by simp
```

If an image of a set defined by separation by a function with infinite positive limit is bounded above, then the set itself is bounded above.

```
lemma (in int0) Int˙ZF˙1˙6˙L2: assumes A1: X≠0 and A2: f: ℤ→ℤ and
  A3: ∀a∈ℤ.∃b∈ℤ₊.∀x. b≤x ⟶ a ≤ f(x) and
  A4: ∀x∈X. b(x) ∈ ℤ  ∧ f(b(x)) ≤ U
  shows ∃u.∀x∈X. b(x) ≤ u
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from A1 A2 A3 A4 have
    group3(G, P, r)
    r –is total on″ G
    G ≠ –TheNeutralElement(G, P)″
    X≠0  f: G→G
    ∀a∈G. ∃b∈PositiveSet(G, P, r). ∀y. ⟨b, y⟩ ∈ r ⟶ ⟨a, f(y)⟩ ∈ r
    ∀x∈X. b(x) ∈ G ∧ ⟨f(b(x)), U⟩ ∈ r
    using int˙not˙trivial Int˙ZF˙2˙T1 by auto
  then have ∃u. ∀x∈X. ⟨b(x), u⟩ ∈ r by (rule group3.OrderedGroup˙ZF˙7˙L2)
  thus thesis by simp
qed
```

If an image of a set defined by separation by a integer function with infinite negative limit is bounded below, then the set itself is bounded above. This is dual to Int˙ZF˙1˙6˙L2.

```
lemma (in int0) Int˙ZF˙1˙6˙L3: assumes A1: X≠0 and A2: f: ℤ→ℤ and
  A3: ∀a∈ℤ.∃b∈ℤ₊.∀y. b≤y ⟶ f(-y) ≤ a and
  A4: ∀x∈X. b(x) ∈ ℤ  ∧ L ≤ f(b(x))
  shows ∃l.∀x∈X. l ≤ b(x)
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from A1 A2 A3 A4 have
    group3(G, P, r)
    r –is total on″ G
    G ≠ –TheNeutralElement(G, P)″
    X≠0  f: G→G
    ∀a∈G. ∃b∈PositiveSet(G, P, r). ∀y.
    ⟨b, y⟩ ∈ r ⟶ ⟨f(GroupInv(G, P)(y)),a⟩ ∈ r
    ∀x∈X. b(x) ∈ G ∧ ⟨L,f(b(x))⟩ ∈ r
    using int˙not˙trivial Int˙ZF˙2˙T1 by auto
  then have ∃l. ∀x∈X. ⟨l, b(x)⟩ ∈ r by (rule group3.OrderedGroup˙ZF˙7˙L3)
  thus thesis by simp
qed
```

The next lemma combines Int˙ZF˙1˙6˙L2 and Int˙ZF˙1˙6˙L3 to show that if the image of a set defined by separation by a function with infinite limits is bounded, then the set itself is bounded. The proof again uses directly a fact from OrderedGroup˙ZF.

```
lemma (in int0) Int˙ZF˙1˙6˙L4:
  assumes A1: X≠0 and A2: f: ℤ→ℤ and
  A3: ∀a∈ℤ.∃b∈ℤ+.∀x. b≤x ⟶ a ≤ f(x) and
  A4: ∀a∈ℤ.∃b∈ℤ+.∀y. b≤y ⟶ f(-y) ≤ a and
  A5: ∀x∈X. b(x) ∈ ℤ ∧ f(b(x)) ≤ U ∧ L ≤ f(b(x))
  shows ∃M.∀x∈X. abs(b(x)) ≤ M
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from A1 A2 A3 A4 A5 have
    group3(G, P, r)
    r –is total on″ G
    G ≠ –TheNeutralElement(G, P)″
    X≠0  f: G→G
    ∀a∈G. ∃b∈PositiveSet(G, P, r). ∀y. ⟨b, y⟩ ∈ r ⟶ ⟨a, f(y)⟩ ∈ r
    ∀a∈G. ∃b∈PositiveSet(G, P, r). ∀y.
    ⟨b, y⟩ ∈ r ⟶ ⟨f(GroupInv(G, P)(y)),a⟩ ∈ r
    ∀x∈X. b(x) ∈ G ∧ ⟨L,f(b(x))⟩ ∈ r ∧ ⟨f(b(x)), U⟩ ∈ r
    using int˙not˙trivial Int˙ZF˙2˙T1 by auto
  then have ∃M. ∀x∈X. ⟨AbsoluteValue(G, P, r)  b(x), M⟩ ∈ r
    by (rule group3.OrderedGroup˙ZF˙7˙L4)
  thus thesis by simp
qed
```

If a function is larger than some constant for arguments large enough, then the image of a set that is bounded below is bounded below. This is not true for ordered groups in general, but only for those for which bounded sets are finite. This does not require the function to have infinite limit, but such functions do have this property.

```
lemma (in int0) Int˙ZF˙1˙6˙L5:
  assumes A1: f: ℤ→ℤ and A2: N∈ℤ and
  A3: ∀m. N≤m ⟶ L ≤ f(m) and
  A4: IsBoundedBelow(A,IntegerOrder)
  shows IsBoundedBelow(f(A),IntegerOrder)
proof -
  from A2 A4 have A = –x∈A. x≤N″ ∪ –x∈A. N≤x″
    using Int˙ZF˙2˙T1 Int˙ZF˙2˙L1C Order˙ZF˙1˙L5
    by simp
  moreover have
    f(–x∈A. x≤N″ ∪ –x∈A. N≤x″) =
    f–x∈A. x≤N″ ∪ f–x∈A. N≤x″
    by (rule image˙Un)
  ultimately have f(A) = f–x∈A. x≤N″ ∪ f–x∈A. N≤x″
```

```
    by simp
  moreover have IsBoundedBelow(f–x∈A. x≤N″,IntegerOrder)
  proof -
    let B = –x∈A. x≤N″
    from A4 have B ∈ Fin(ℤ)
      using Order˙ZF˙3˙L16 Int˙bounded˙iff˙fin by auto
    with A1 have  IsBounded(f(B),IntegerOrder)
      using Finite1˙L6A Int˙bounded˙iff˙fin by simp
    then show IsBoundedBelow(f(B),IntegerOrder)
      using IsBounded˙def by simp
  qed
  moreover have IsBoundedBelow(f–x∈A. N≤x″,IntegerOrder)
  proof -
    let C = –x∈A. N≤x″
    from A4 have C ⊆ ℤ using Int˙ZF˙2˙L1C by auto
    with A1 A3 have ∀y ∈ f(C). ⟨L,y⟩ ∈ IntegerOrder
      using func˙imagedef by simp
    then show IsBoundedBelow(f(C),IntegerOrder)
      by (rule Order˙ZF˙3˙L9)
  qed
  ultimately show IsBoundedBelow(f(A),IntegerOrder)
    using Int˙ZF˙2˙T1 Int˙ZF˙2˙L6 Int˙ZF˙2˙L1B Order˙ZF˙3˙L6
    by simp
qed
```

A function that has an infinite limit can be made arbitrarily large on positive integers by adding a constant. This does not actually require the function to have infinite limit, just to be larger than a constant for arguments large enough.

```
lemma (in int0) Int˙ZF˙1˙6˙L6: assumes A1: N∈ℤ and
  A2: ∀m. N≤m ⟶ L ≤ f(m) and
  A3: f: ℤ→ℤ and A4: K∈ℤ
  shows ∃c∈ℤ. ∀n∈ℤ₊. K ≤ f(n)+c
proof -
  have IsBoundedBelow(ℤ₊,IntegerOrder)
    using Int˙ZF˙1˙5˙L1 by simp
  with A3 A1 A2 have IsBoundedBelow(f(ℤ₊),IntegerOrder)
    by (rule Int˙ZF˙1˙6˙L5)
  with A1 obtain l where I: ∀y∈f(ℤ₊). l ≤ y
    using Int˙ZF˙1˙5˙L5 IsBoundedBelow˙def by auto
  let c = K-l
  from A3 have f(ℤ₊) ≠ 0 using Int˙ZF˙1˙5˙L5
    by simp
  then have ∃y. y ∈ f(ℤ₊) by (rule nonempty˙has˙element)
  then obtain y where y ∈ f(ℤ₊) by auto
  with A4 I have T: l ∈ ℤ  c ∈ ℤ
    using Int˙ZF˙2˙L1A Int˙ZF˙1˙1˙L5 by auto
  { fix n assume A5: n∈ℤ₊
    have ℤ₊ ⊆ ℤ using PositiveSet˙def by auto
```

```
    with A3 I T A5 have l + c ≤ f(n) + c
      using func˙imagedef int˙ord˙transl˙inv by auto
     with I T have l + c ≤ f(n) + c
      using int˙ord˙transl˙inv by simp
    with A4 T have K ≤  f(n) + c
      using Int˙ZF˙1˙2˙L3 by simp
  } then have ∀n∈ℤ+. K ≤  f(n) + c by simp
  with T show thesis by auto
qed
```

If a function has infinite limit, then we can add such constant such that minimum of those arguments for which the function (plus the constant) is larger than another given constant is greater than a third constant. It is not as complicated as it sounds.

```
lemma (in int0) Int˙ZF˙1˙6˙L7:
  assumes A1: f: ℤ→ℤ and A2: K∈ℤ  N∈ℤ and
  A3: ∀a∈ℤ.∃b∈ℤ+.∀x. b≤x ⟶ a ≤ f(x)
  shows ∃C∈ℤ. N ≤ Minimum(IntegerOrder,–n∈ℤ+. K ≤ f(n)+C″)
proof -
  from A1 A2 have ∃C∈ℤ. ∀n∈ℤ+. K ≤ f(n) + C ⟶ N≤n
    using Int˙ZF˙1˙5˙L4 by simp
  then obtain C where I: C∈ℤ and
    II: ∀n∈ℤ+. K ≤ f(n) + C ⟶ N≤n
    by auto
  have antisym(IntegerOrder) using Int˙ZF˙2˙L4 by simp
  moreover have HasAminimum(IntegerOrder,–n∈ℤ+. K ≤ f(n)+C″)
  proof -
    from A2 A3 I have ∃n∈ℤ+.∀x. n≤x ⟶ K-C ≤ f(x)
      using Int˙ZF˙1˙1˙L5 by simp
    then obtain n where
      n∈ℤ+ and ∀x. n≤x ⟶ K-C ≤  f(x)
      by auto
    with A2 I have
      –n∈ℤ+. K ≤ f(n)+C″ ≠ 0
      –n∈ℤ+. K ≤ f(n)+C″ ⊆ ℤ+
      using int˙ord˙is˙refl refl˙def PositiveSet˙def Int˙ZF˙2˙L9C
      by auto
    then show HasAminimum(IntegerOrder,–n∈ℤ+. K ≤ f(n)+C″)
      using Int˙ZF˙1˙5˙L1C by simp
  qed
  moreover from II have
    ∀n ∈ –n∈ℤ+. K ≤ f(n)+C″. ⟨N,n⟩ ∈ IntegerOrder
    by auto
  ultimately have
    ⟨N,Minimum(IntegerOrder,–n∈ℤ+. K ≤ f(n)+C″)⟩ ∈ IntegerOrder
    by (rule Order˙ZF˙4˙L12)
  with I show thesis by auto
qed
```

For any integer m the function $k \mapsto m \cdot k$ has an infinite limit (or negative of that). This is why we put some properties of these functions here, even though they properly belong to a (yet nonexistent) section on homomorphisms. The next lemma shows that the set $\{a \cdot x : x \in Z\}$ can finite only if $a = 0$.

```
lemma (in int0) Int˙ZF˙1˙6˙L8:
  assumes A1: a∈ℤ and A2: −a·x. x∈ℤ″ ∈ Fin(ℤ)
  shows a = 0
proof -
  from A1 have a=0 ∨ (a ≤ -1) ∨ (1≤a)
    using Int˙ZF˙1˙3˙L6C by simp
  moreover
  { assume a ≤ -1
    then have −a·x. x∈ℤ″ ∉ Fin(ℤ)
      using int˙zero˙not˙one Int˙ZF˙1˙3˙T1 ring1.OrdRing˙ZF˙3˙L6
      by simp
    with A2 have False by simp }
  moreover
  { assume 1≤a
    then have −a·x. x∈ℤ″ ∉ Fin(ℤ)
      using int˙zero˙not˙one Int˙ZF˙1˙3˙T1 ring1.OrdRing˙ZF˙3˙L5
    by simp
  with A2 have False by simp }
  ultimately show  a = 0 by auto
qed
```

43.7 Miscelaneous

In this section we put some technical lemmas needed in various other places that are hard to classify.

Suppose we have an integer expression (a meta-function)F such that $F(p)|p|$ is bounded by a linear function of $|p|$, that is for some integers A, B we have $F(p)|p| \leq A|p| + B$. We show that F is then bounded. The proof is easy, we just divide both sides by $|p|$ and take the limit (just kidding).

```
lemma (in int0) Int˙ZF˙1˙7˙L1:
  assumes A1: ∀q∈ℤ. F(q) ∈ ℤ and
  A2:  ∀q∈ℤ. F(q)·abs(q) ≤ A·abs(q) + B and
  A3: A∈ℤ  B∈ℤ
  shows ∃L. ∀p∈ℤ. F(p) ≤ L
proof -
  let I = (-abs(B))..abs(B)
  let K = −F(q). q ∈ I″
  let M = Maximum(IntegerOrder,K)
  let L = GreaterOf(IntegerOrder,M,A+1)
  from A3 A1 have C1:
    IsBounded(I,IntegerOrder)
    I ≠ 0
```

```
    ∀q∈ℤ. F(q) ∈ ℤ
    K = −F(q). q ∈ I″
    using Order˙ZF˙3˙L11 Int˙ZF˙1˙3˙L17 by auto
  then have M ∈ ℤ by (rule Int˙ZF˙1˙4˙L1)
  with A3 have T1: M ≤ L  A+𝟏 ≤ L
    using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L5 Int˙ZF˙1˙3˙L18
    by auto
  from C1 have T2: ∀q∈I. F(q) ≤ M
    by (rule Int˙ZF˙1˙4˙L1)
  { fix p assume A4: p∈ℤ have F(p) ≤ L
    proof -
      { assume abs(p) ≤ abs(B)
 with A4 T1 T2 have F(p) ≤ M  M ≤ L
   using Int˙ZF˙1˙3˙L19 by auto
 then have F(p) ≤ L by (rule Int˙order˙transitive) }
      moreover
      { assume A5: ¬(abs(p) ≤ abs(B))
 from A3 A2 A4 have
   A·abs(p) ∈ ℤ  F(p)·abs(p) ≤ A·abs(p) + B
   using Int˙ZF˙2˙L14 Int˙ZF˙1˙1˙L5 by auto
 moreover from A3 A4 A5 have B ≤ abs(p)
   using Int˙ZF˙1˙3˙L15 by simp
 ultimately have
   F(p)·abs(p) ≤ A·abs(p) + abs(p)
   using Int˙ZF˙2˙L15A by blast
 with A3 A4 have F(p)·abs(p) ≤ (A+𝟏)·abs(p)
   using Int˙ZF˙2˙L14 Int˙ZF˙1˙2˙L7 by simp
 moreover from A3 A1 A4 A5 have
   F(p) ∈ ℤ  A+𝟏 ∈ ℤ  abs(p) ∈ ℤ
   ¬(abs(p) ≤ 𝟎)
   using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L5 Int˙ZF˙2˙L14 Int˙ZF˙1˙3˙L11
   by auto
 ultimately have F(p) ≤ A+𝟏
   using Int˙ineq˙simpl˙positive by simp
 moreover from T1 have  A+𝟏 ≤ L by simp
 ultimately have F(p) ≤ L by (rule Int˙order˙transitive) }
      ultimately show thesis by blast
    qed
  } then have ∀p∈ℤ. F(p) ≤ L by simp
  thus thesis by auto
qed
```

A lemma about splitting (not really, there is some overlap) the 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 assms Int˙ZF˙2˙T1 group3.OrdGroup˙6cases by simp

end
```

44 Division on integers

```
theory IntDiv˙ZF˙IML imports Int˙ZF˙1 ZF.IntDiv

begin
```

This theory translates some results form the Isabelle's IntDiv.thy theory to the notation used by IsarMathLib.

44.1 Quotient and reminder

For any integers m, n , $n > 0$ there are unique integers q, p such that $0 \leq p < n$ and $m = n \cdot q + p$. Number p in this decompsition is usually called m mod n. Standard Isabelle denotes numbers q, p as m zdiv n and m zmod n, resp., and we will use the same notation.

The next lemma is sometimes called the "quotient-reminder theorem".

```
lemma (in int0) IntDiv˙ZF˙1˙L1: assumes m∈ℤ  n∈ℤ
  shows m = n·(m zdiv n) + (m zmod n)
  using assms Int˙ZF˙1˙L2 raw˙zmod˙zdiv˙equality
  by simp
```

If n is greater than 0 then m zmod n is between 0 and $n - 1$.

```
lemma (in int0) IntDiv˙ZF˙1˙L2:
  assumes A1: m∈ℤ and A2: 0≤n  n≠0
  shows
  0 ≤ m zmod n
  m zmod n ≤ n    m zmod n ≠ n
  m zmod n ≤ n-1
proof -
  from A2 have T: n ∈ ℤ
    using Int˙ZF˙2˙L1A by simp
  from A2 have #0 $¡ n using Int˙ZF˙2˙L9 Int˙ZF˙1˙L8
    by auto
  with T show
    0 ≤ m zmod n
    m zmod n ≤ n
    m zmod n ≠ n
    using pos˙mod Int˙ZF˙1˙L8 Int˙ZF˙1˙L8A zmod˙type
      Int˙ZF˙2˙L1 Int˙ZF˙2˙L9AA
    by auto
  then show m zmod n ≤ n-1
```

```
  using Int˙ZF˙4˙L1B by auto
qed
```

$(m \cdot k)$ div $k = m$.

```
lemma (in int0) IntDiv˙ZF˙1˙L3:
 assumes m∈ℤ  k∈ℤ  and k≠𝟬
 shows
 (m·k) zdiv k = m
 (k·m) zdiv k = m
 using assms zdiv˙zmult˙self1 zdiv˙zmult˙self2
  Int˙ZF˙1˙L8 Int˙ZF˙1˙L2 by auto
```

The next lemma essentially translates zdiv˙mono1 from standard Isabelle to our notation.

```
lemma (in int0) IntDiv˙ZF˙1˙L4:
 assumes A1: m ≤ k and A2: 𝟬≤n  n≠𝟬
 shows m zdiv n ≤  k zdiv n
proof -
 from A2 have #0 ≤ n  #0 ≠ n
  using Int˙ZF˙1˙L8 by auto
 with A1 have
  m zdiv n $≤ k zdiv n
  m zdiv n ∈ ℤ    m zdiv k ∈ ℤ
  using Int˙ZF˙2˙L1A Int˙ZF˙2˙L9 zdiv˙mono1
  by auto
 then show (m zdiv n) ≤ (k zdiv n)
  using Int˙ZF˙2˙L1 by simp
qed
```

A quotient-reminder theorem about integers greater than a given product.

```
lemma (in int0) IntDiv˙ZF˙1˙L5:
 assumes A1: n ∈ ℤ₊ and A2: n ≤ k and A3: k·n ≤ m
 shows
 m = n·(m zdiv n) + (m zmod n)
 m = (m zdiv n)·n + (m zmod n)
 (m zmod n) ∈ 𝟬..(n-𝟭)
 k ≤ (m zdiv n)
 m zdiv n ∈ ℤ₊
proof -
 from A2 A3 have T:
  m∈ℤ  n∈ℤ  k∈ℤ  m zdiv n ∈ ℤ
  using Int˙ZF˙2˙L1A by auto
  then show m = n·(m zdiv n) + (m zmod n)
   using IntDiv˙ZF˙1˙L1 by simp
  with T show m = (m zdiv n)·n + (m zmod n)
   using Int˙ZF˙1˙L4 by simp
  from A1 have I: 𝟬≤n  n≠𝟬
   using PositiveSet˙def by auto
  with T show (m zmod n) ∈ 𝟬..(n-𝟭)
```

```
    using IntDiv˙ZF˙1˙L2 Order˙ZF˙2˙L1
    by simp
  from A3 I have (k·n zdiv n) ≤ (m zdiv n)
    using IntDiv˙ZF˙1˙L4 by simp
  with I T show k ≤ (m zdiv n)
    using IntDiv˙ZF˙1˙L3 by simp
  with A1 A2 show m zdiv n ∈ ℤ+
    using Int˙ZF˙1˙5˙L7 by blast
qed
```

end

45 Integers 2

theory Int˙ZF˙2 imports func˙ZF˙1 Int˙ZF˙1 IntDiv˙ZF˙IML Group˙ZF˙3

begin

In this theory file we consider the properties of integers that are needed for the real numbers construction in Real˙ZF series.

45.1 Slopes

In this section we study basic properties of slopes - the integer almost homomorphisms. The general definition of an almost homomorphism f on a group G written in additive notation requires the set $\{f(m+n) - f(m) - f(n) : m, n \in G\}$ to be finite. In this section we establish a definition that is equivalent for integers: that for all integer m, n we have $|f(m+n)-f(m)-f(n)| \leq L$ for some L.

First we extend the standard notation for integers with notation related to slopes. We define slopes as almost homomorphisms on the additive group of integers. The set of slopes is denoted $\mathcal{S}$. We also define "positive" slopes as those that take infinite number of positive values on positive integers. We write δ(s,m,n) to denote the homomorphism difference of s at m, n (i.e the expression $s(m+n) - s(m) - s(n)$). We denote maxδ(s) the maximum absolute value of homomorphism difference of s as m, n range over integers. If s is a slope, then the set of homomorphism differences is finite and this maximum exists. In Group˙ZF˙3 we define the equivalence relation on almost homomorphisms using the notion of a quotient group relation and use "$\approx$" to denote it. As here this symbol seems to be hogged by the standard Isabelle, we will use "$\sim$" instead "$\approx$". We show in this section that $s \sim r$ iff for some L we have $|s(m) - r(m)| \leq L$ for all integer m. The "+" denotes the first operation on almost homomorphisms. For slopes this is addition of functions defined in the natural way. The "$\circ$" symbol denotes the second operation on

almost homomorphisms (see Group˙ZF˙3 for definition), defined for the group of integers. In short "$\circ$" is the composition of slopes. The "$^{-1}$" symbol acts as an infix operator that assigns the value $\min\{n \in Z_+ : p \leq f(n)\}$ to a pair (of sets) f and p. In application f represents a function defined on Z_+ and p is a positive integer. We choose this notation because we use it to construct the right inverse in the ring of classes of slopes and show that this ring is in fact a field. To study the homomorphism difference of the function defined by $p \mapsto f^{-1}(p)$ we introduce the symbol ε defined as $\varepsilon(f, \langle m, n\rangle) = f^{-1}(m+n) - f^{-1}(m) - f^{-1}(n)$. Of course the intention is to use the fact that $\varepsilon(f, \langle m, n\rangle)$ is the homomorphism difference of the function g defined as $g(m) = f^{-1}(m)$. We also define $\gamma(s, m, n)$ as the expression $\delta(f, m, -n) + s(0) - \delta(f, n, -n)$. This is useful because of the identity $f(m-n) = \gamma(m, n) + f(m) - f(n)$ that allows to obtain bounds on the value of a slope at the difference of of two integers. For every integer m we introduce notation m^S defined by $m^E(n) = m \cdot n$. The mapping $q \mapsto q^S$ embeds integers into $\mathcal{S}$ preserving the order, (that is, maps positive integers into $\mathcal{S}_+$).

locale int1 = int0 +

fixes slopes ($\mathcal{S}$)
defines slopes˙def[simp]: $\mathcal{S}$ ≡ AlmostHoms($\mathbb{Z}$,IntegerAddition)

fixes posslopes ($\mathcal{S}_+$)
defines posslopes˙def[simp]: $\mathcal{S}_+$ ≡ –s∈$\mathcal{S}$. s($\mathbb{Z}_+$) ∩ $\mathbb{Z}_+$ ∉ Fin($\mathbb{Z}$)″

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⟩ ∈ $\mathbb{Z}$×$\mathbb{Z}$″)

fixes AlEqRel
defines AlEqRel˙def[simp]:
AlEqRel ≡ QuotientGroupRel($\mathcal{S}$,AlHomOp1($\mathbb{Z}$,IntegerAddition),FinRangeFunctions($\mathbb{Z}$,$\mathbb{Z}$))

fixes AlEq (infix ∼ 68)
defines AlEq˙def[simp]: s ∼ r ≡ ⟨ s,r⟩ ∈ AlEqRel

fixes slope˙add (infix + 70)
defines slope˙add˙def[simp]: s + r ≡ AlHomOp1($\mathbb{Z}$,IntegerAddition)⟨ s,r⟩

fixes slope˙comp (infix ∘ 70)
defines slope˙comp˙def[simp]: s ∘ r ≡ AlHomOp2($\mathbb{Z}$,IntegerAddition)⟨ s,r⟩

fixes neg (-˙ [90] 91)
defines neg˙def[simp]: -s ≡ GroupInv($\mathbb{Z}$,IntegerAddition) O s

fixes slope˙inv (infix $^{-1}$ 71)
defines slope˙inv˙def[simp]:
$f^{-1}(p) \equiv$ Minimum(IntegerOrder,–n∈$\mathbb{Z}_+$. 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 ($^{\cdot S}$)
defines intembed˙def[simp]: m^S ≡ –⟨n,m·n⟩. n∈$\mathbb{Z}$″

We can use theorems proven in the group1 context.

lemma (in int1) Int˙ZF˙2˙1˙L1: shows group1($\mathbb{Z}$,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∈$\mathcal{S}$ and n∈$\mathbb{Z}$ m∈$\mathbb{Z}$
shows
m+n ∈ $\mathbb{Z}$
f(m+n) ∈ $\mathbb{Z}$
f(m) ∈ $\mathbb{Z}$ f(n) ∈ $\mathbb{Z}$
f(m) + f(n) ∈ $\mathbb{Z}$
HomDiff($\mathbb{Z}$,IntegerAddition,f,⟨ m,n⟩) ∈ $\mathbb{Z}$
using assms Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙2˙L4A
by auto

Type information related to the homomorphism difference expression.

lemma (in int1) Int˙ZF˙2˙1˙L2A:
assumes f:$\mathbb{Z}$→$\mathbb{Z}$ and n∈$\mathbb{Z}$ m∈$\mathbb{Z}$
shows
m+n ∈ $\mathbb{Z}$
f(m+n) ∈ $\mathbb{Z}$ f(m) ∈ $\mathbb{Z}$ f(n) ∈ $\mathbb{Z}$
f(m) + f(n) ∈ $\mathbb{Z}$
HomDiff($\mathbb{Z}$,IntegerAddition,f,⟨ m,n⟩) ∈ $\mathbb{Z}$
using assms Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙2˙L4
by auto

Slopes map integers into integers.

lemma (in int1) Int˙ZF˙2˙1˙L2B:
assumes A1: f∈$\mathcal{S}$ and A2: m∈$\mathbb{Z}$
shows f(m) ∈ $\mathbb{Z}$
proof -
from A1 have f:$\mathbb{Z}$→$\mathbb{Z}$ using AlmostHoms˙def by simp
with A2 show f(m) ∈ $\mathbb{Z}$ using apply˙funtype by simp

qed

The homomorphism difference in multiplicative notation is defined as the expression $s(m \cdot n) \cdot (s(m) \cdot s(n))^{-1}$. The next lemma shows that in the additive notation used for integers the homomorphism difference is $f(m + n) - f(m) - f(n)$ which we denote as δ(f,m,n).

```
lemma (in int1) Int˙ZF˙2˙1˙L3:
  assumes f:ℤ→ℤ and m∈ℤ  n∈ℤ
  shows HomDiff(ℤ,IntegerAddition,f,⟨ m,n⟩) = δ(f,m,n)
  using assms Int˙ZF˙2˙1˙L2A Int˙ZF˙1˙T2 group0.group0˙4˙L4A
    HomDiff˙def by auto
```

The next formula restates the definition of the homomorphism difference to express the value an almost homomorphism on a sum.

```
lemma (in int1) Int˙ZF˙2˙1˙L3A:
  assumes A1: f∈𝒮 and A2: m∈ℤ  n∈ℤ
  shows
  f(m+n) = f(m)+(f(n)+δ(f,m,n))
proof -
  from A1 A2 have
    T: f(m)∈ ℤ  f(n) ∈ ℤ  δ(f,m,n) ∈ ℤ and
    HomDiff(ℤ,IntegerAddition,f,⟨ m,n⟩) = δ(f,m,n)
    using Int˙ZF˙2˙1˙L2 AlmostHoms˙def Int˙ZF˙2˙1˙L3 by auto
  with A1 A2 show  f(m+n) = f(m)+(f(n)+δ(f,m,n))
    using Int˙ZF˙2˙1˙L3 Int˙ZF˙1˙L3
      Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙4˙L1
    by simp
qed
```

The homomorphism difference of any integer function is integer.

```
lemma (in int1) Int˙ZF˙2˙1˙L3B:
  assumes f:ℤ→ℤ and m∈ℤ  n∈ℤ
  shows δ(f,m,n) ∈ ℤ
  using assms Int˙ZF˙2˙1˙L2A Int˙ZF˙2˙1˙L3 by simp
```

The value of an integer function at a sum expressed in terms of δ.

```
lemma (in int1) Int˙ZF˙2˙1˙L3C: assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ
  shows f(m+n) = δ(f,m,n) + f(n) + f(m)
proof -
  from A1 A2 have T:
    δ(f,m,n) ∈ ℤ  f(m+n) ∈ ℤ  f(m) ∈ ℤ  f(n) ∈ ℤ
    using Int˙ZF˙1˙1˙L5 apply˙funtype by auto
  then show f(m+n) = δ(f,m,n) + f(n) + f(m)
    using Int˙ZF˙1˙2˙L15 by simp
qed
```

The next lemma presents two ways the set of homomorphism differences can be written.

```
lemma (in int1) Int˙ZF˙2˙1˙L4: assumes A1: f:ℤ→ℤ
  shows –abs(HomDiff(ℤ,IntegerAddition,f,x)). x ∈ ℤ×ℤ″ =
  –abs(δ(f,m,n)). ⟨ m,n⟩ ∈ ℤ×ℤ″
proof -
  from A1 have ∀m∈ℤ. ∀n∈ℤ.
    abs(HomDiff(ℤ,IntegerAddition,f,⟨ m,n⟩)) = abs(δ(f,m,n))
    using Int˙ZF˙2˙1˙L3 by simp
  then show thesis by (rule ZF1˙1˙L4A)
qed
```

If f maps integers into integers and for all $m, n \in Z$ we have $|f(m+n) - f(m) - f(n)| \leq L$ for some L, then f is a slope.

```
lemma (in int1) Int˙ZF˙2˙1˙L5: assumes A1: f:ℤ→ℤ
  and A2: ∀m∈ℤ.∀n∈ℤ. abs(δ(f,m,n)) ≤ L
  shows f∈𝒮
proof -
  let Abs = AbsoluteValue(ℤ,IntegerAddition,IntegerOrder)
  have group3(ℤ,IntegerAddition,IntegerOrder)
    IntegerOrder –is total on″ ℤ
    using Int˙ZF˙2˙T1 by auto
  moreover from A1 A2 have
    ∀x∈ℤ×ℤ. HomDiff(ℤ,IntegerAddition,f,x) ∈ ℤ ∧
    ⟨Abs(HomDiff(ℤ,IntegerAddition,f,x)),L ⟩ ∈ IntegerOrder
    using Int˙ZF˙2˙1˙L2A Int˙ZF˙2˙1˙L3 by auto
  ultimately have
    IsBounded(–HomDiff(ℤ,IntegerAddition,f,x). x∈ℤ×ℤ″,IntegerOrder)
    by (rule group3.OrderedGroup˙ZF˙3˙L9A)
  with A1 show f ∈ 𝒮 using Int˙bounded˙iff˙fin AlmostHoms˙def
    by simp
qed
```

The absolute value of homomorphism difference of a slope s does not exceed maxδ(s).

```
lemma (in int1) Int˙ZF˙2˙1˙L7:
  assumes A1: s∈𝒮 and A2: n∈ℤ  m∈ℤ
  shows
  abs(δ(s,m,n)) ≤ maxδ(s)
  δ(s,m,n) ∈ ℤ   maxδ(s) ∈ ℤ
  (-maxδ(s)) ≤ δ(s,m,n)
proof -
  from A1 A2 show T: δ(s,m,n) ∈ ℤ
    using Int˙ZF˙2˙1˙L2 Int˙ZF˙1˙1˙L5 by simp
  let A = –abs(HomDiff(ℤ,IntegerAddition,s,x)). x∈ℤ×ℤ″
  let B = –abs(δ(s,m,n)). ⟨ m,n⟩ ∈ ℤ×ℤ″
  let d = abs(δ(s,m,n))
  have IsLinOrder(ℤ,IntegerOrder) using Int˙ZF˙2˙T1
    by simp
  moreover have A ∈ Fin(ℤ)
  proof -
```

```
    have ∀k∈ℤ. abs(k) ∈ ℤ using Int˙ZF˙2˙L14 by simp
    moreover from A1 have
      –HomDiff(ℤ,IntegerAddition,s,x). x ∈ ℤ×ℤ″ ∈ Fin(ℤ)
      using AlmostHoms˙def by simp
    ultimately show A ∈ Fin(ℤ) by (rule Finite1˙L6C)
  qed
  moreover have A≠0 by auto
  ultimately have ∀k∈A. ⟨k,Maximum(IntegerOrder,A)⟩ ∈ IntegerOrder
    by (rule Finite˙ZF˙1˙T2)
  moreover from A1 A2 have d∈A using AlmostHoms˙def Int˙ZF˙2˙1˙L4
    by auto
  ultimately have d ≤ Maximum(IntegerOrder,A) by auto
  with A1 show d ≤ maxδ(s)  maxδ(s) ∈ ℤ
    using AlmostHoms˙def Int˙ZF˙2˙1˙L4 Int˙ZF˙2˙L1A
    by auto
  with T show (-maxδ(s)) ≤ δ(s,m,n)
    using Int˙ZF˙1˙3˙L19 by simp
qed
```

A useful estimate for the value of a slope at 0, plus some type information for slopes.

```
lemma (in int1) Int˙ZF˙2˙1˙L8: assumes A1: s∈𝒮
  shows
  abs(s(𝟎)) ≤ maxδ(s)
  𝟎 ≤ maxδ(s)
  abs(s(𝟎)) ∈ ℤ   maxδ(s) ∈ ℤ
  abs(s(𝟎)) + maxδ(s) ∈ ℤ
proof -
  from A1 have s(𝟎) ∈ ℤ
    using int˙zero˙one˙are˙int Int˙ZF˙2˙1˙L2B by simp
  then have I: 𝟎 ≤ abs(s(𝟎))
    and abs(δ(s,𝟎,𝟎)) = abs(s(𝟎))
    using int˙abs˙nonneg int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L4
      Int˙ZF˙2˙L17 by auto
  moreover from A1 have abs(δ(s,𝟎,𝟎)) ≤ maxδ(s)
    using int˙zero˙one˙are˙int Int˙ZF˙2˙1˙L7 by simp
  ultimately show II: abs(s(𝟎)) ≤ maxδ(s)
    by simp
  with I show 𝟎≤maxδ(s) by (rule Int˙order˙transitive)
  with II show
    maxδ(s) ∈ ℤ   abs(s(𝟎)) ∈ ℤ
    abs(s(𝟎)) + maxδ(s) ∈ ℤ
    using Int˙ZF˙2˙L1A Int˙ZF˙1˙1˙L5 by auto
qed
```

Int Group˙ZF˙3.thy we show that finite range functions valued in an abelian group form a normal subgroup of almost homomorphisms. This allows to define the equivalence relation between almost homomorphisms as the relation resulting from dividing by that normal subgroup. Then we show in

Group˙ZF˙3˙4˙L12 that if the difference of f and g has finite range (actually $f(n) \cdot g(n)^{-1}$ as we use multiplicative notation in Group˙ZF˙3.thy), then f and g are equivalent. The next lemma translates that fact into the notation used in int1 context.

```
lemma (in int1) Int˙ZF˙2˙1˙L9: assumes A1: s∈𝒮  r∈𝒮
  and A2: ∀m∈ℤ. abs(s(m)-r(m)) ≤ L
  shows s ∼ r
proof -
  from A1 A2 have
    ∀m∈ℤ. s(m)-r(m) ∈ ℤ ∧ abs(s(m)-r(m)) ≤ L
    using Int˙ZF˙2˙1˙L2B Int˙ZF˙1˙1˙L5 by simp
  then have
    IsBounded(–s(n)-r(n). n∈ℤ˝, IntegerOrder)
    by (rule Int˙ZF˙1˙3˙L20)
  with A1 show s ∼ r using Int˙bounded˙iff˙fin
    Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙4˙L12 by simp
qed
```

A neccessary condition for two slopes to be almost equal. For slopes the definition postulates the set $\{f(m) - g(m) : m \in Z\}$ to be finite. This lemma shows that this implies that $|f(m) - g(m)|$ is bounded (by some integer) as m varies over integers. We also mention here that in this context s ∼ r implies that both s and r are slopes.

```
lemma (in int1) Int˙ZF˙2˙1˙L9A: assumes s ∼ r
  shows
  ∃L∈ℤ. ∀m∈ℤ. abs(s(m)-r(m)) ≤ L
  s∈𝒮  r∈𝒮
  using assms Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙4˙L11
    Int˙ZF˙1˙3˙L20AA QuotientGroupRel˙def by auto
```

Let's recall that the relation of almost equality is an equivalence relation on the set of slopes.

```
lemma (in int1) Int˙ZF˙2˙1˙L9B: shows
  AlEqRel ⊆ 𝒮×𝒮
  equiv(𝒮,AlEqRel)
  using Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙3˙L3 by auto
```

Another version of sufficient condition for two slopes to be almost equal: if the difference of two slopes is a finite range function, then they are almost equal.

```
lemma (in int1) Int˙ZF˙2˙1˙L9C: assumes s∈𝒮  r∈𝒮 and
  s + (-r) ∈ FinRangeFunctions(ℤ,ℤ)
  shows
  s ∼ r
  r ∼ s
  using assms Int˙ZF˙2˙1˙L1
    group1.Group˙ZF˙3˙2˙L13 group1.Group˙ZF˙3˙4˙L12A
```

by auto

If two slopes are almost equal, then the difference has finite range. This is the inverse of Int˙ZF˙2˙1˙L9C.

lemma (in int1) Int˙ZF˙2˙1˙L9D: assumes A1: s ∼ r
shows s + (-r) ∈ FinRangeFunctions(ℤ,ℤ)
proof -
let G = ℤ
let f = IntegerAddition
from A1 have AlHomOp1(G, f)⟨s,GroupInv(AlmostHoms(G, f),AlHomOp1(G, f))(r)⟩
∈ FinRangeFunctions(G, G)
using Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙4˙L12B by auto
with A1 show s + (-r) ∈ FinRangeFunctions(ℤ,ℤ)
using Int˙ZF˙2˙1˙L9A Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙2˙L13
by simp
qed

What is the value of a composition of slopes?

lemma (in int1) Int˙ZF˙2˙1˙L10:
assumes s∈$\mathcal{S}$ r∈$\mathcal{S}$ and m∈ℤ
shows (s∘r)(m) = s(r(m)) s(r(m)) ∈ ℤ
using assms Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙4˙L2 by auto

Composition of slopes is a slope.

lemma (in int1) Int˙ZF˙2˙1˙L11:
assumes s∈$\mathcal{S}$ r∈$\mathcal{S}$
shows s∘r ∈ $\mathcal{S}$
using assms Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙4˙T1 by simp

Negative of a slope is a slope.

lemma (in int1) Int˙ZF˙2˙1˙L12: assumes s∈$\mathcal{S}$ shows -s ∈ $\mathcal{S}$
using assms Int˙ZF˙1˙T2 Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙2˙L13
by simp

What is the value of a negative of a slope?

lemma (in int1) Int˙ZF˙2˙1˙L12A:
assumes s∈$\mathcal{S}$ and m∈ℤ shows (-s)(m) = -(s(m))
using assms Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙2˙L5
by simp

What are the values of a sum of slopes?

lemma (in int1) Int˙ZF˙2˙1˙L12B: assumes s∈$\mathcal{S}$ r∈$\mathcal{S}$ and m∈ℤ
shows (s+r)(m) = s(m) + r(m)
using assms Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙2˙L12
by simp

Sum of slopes is a slope.

lemma (in int1) Int˙ZF˙2˙1˙L12C: assumes s∈$\mathcal{S}$ r∈$\mathcal{S}$
shows s+r ∈ $\mathcal{S}$
using assms Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙2˙L16
by simp

A simple but useful identity.

lemma (in int1) Int˙ZF˙2˙1˙L13:
assumes s∈$\mathcal{S}$ and n∈$\mathbb{Z}$ m∈$\mathbb{Z}$
shows s(n·m) + (s(m) + δ(s,n·m,m)) = s((n+**1**)·m)
using assms Int˙ZF˙1˙1˙L5 Int˙ZF˙2˙1˙L2B Int˙ZF˙1˙2˙L9 Int˙ZF˙1˙2˙L7
by simp

Some estimates for the absolute value of a slope at the opposite integer.

lemma (in int1) Int˙ZF˙2˙1˙L14: assumes A1: s∈$\mathcal{S}$ and A2: m∈$\mathbb{Z}$
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) ∈ $\mathbb{Z}$ abs(s(m)) ∈ $\mathbb{Z}$ s(**0**) ∈ $\mathbb{Z}$ abs(s(**0**)) ∈ $\mathbb{Z}$
δ(s,m,-m) ∈ $\mathbb{Z}$ s(m) ∈ $\mathbb{Z}$ s(-m) ∈ $\mathbb{Z}$
(-(s(m))) ∈ $\mathbb{Z}$ s(**0**) - δ(s,m,-m) ∈ $\mathbb{Z}$
using Int˙ZF˙1˙1˙L4 Int˙ZF˙2˙1˙L2B Int˙ZF˙2˙L14 Int˙ZF˙2˙1˙L2
Int˙ZF˙1˙1˙L5 int˙zero˙one˙are˙int by auto
with A2 show I: s(-m) = s(**0**) - δ(s,m,-m) - s(m)
using Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙2˙L15 by simp
from T have abs(s(**0**) - δ(s,m,-m)) ≤ abs(s(**0**)) + abs(δ(s,m,-m))
using Int˙triangle˙ineq1 by simp
moreover from A1 A2 T have abs(s(**0**)) + abs(δ(s,m,-m)) ≤ **2**·maxδ(s)
using Int˙ZF˙2˙1˙L7 Int˙ZF˙2˙1˙L8 Int˙ZF˙1˙3˙L21 by simp
ultimately have abs(s(**0**) - δ(s,m,-m)) ≤ **2**·maxδ(s)
by (rule Int˙order˙transitive)
moreover
from I have s(m) + s(-m) = s(m) + (s(**0**) - δ(s,m,-m) - s(m))
by simp
with T have abs(s(m) + s(-m)) = abs(s(**0**) - δ(s,m,-m))
using Int˙ZF˙1˙2˙L3 by simp
ultimately show abs(s(m)+s(-m)) ≤ **2**·maxδ(s)
by simp
from I have abs(s(-m)) = abs(s(**0**) - δ(s,m,-m) - s(m))
by simp
with T have
abs(s(-m)) ≤ abs(s(**0**)) + abs(δ(s,m,-m)) + abs(s(m))
using int˙triangle˙ineq3 by simp
moreover from A1 A2 T have
abs(s(**0**)) + abs(δ(s,m,-m)) + abs(s(m)) ≤ **2**·maxδ(s) + abs(s(m))
using Int˙ZF˙2˙1˙L7 Int˙ZF˙2˙1˙L8 Int˙ZF˙1˙3˙L21 int˙ord˙transl˙inv by simp

```
  ultimately show abs(s(-m)) ≤ 2·maxδ(s) + abs(s(m))
    by (rule Int˙order˙transitive)
  from T have s(0) - δ(s,m,-m) ≤ abs(s(0)) + abs(δ(s,m,-m))
    using Int˙ZF˙2˙L15E by simp
  moreover from A1 A2 T have
    abs(s(0)) + abs(δ(s,m,-m)) ≤ abs(s(0)) + maxδ(s)
    using Int˙ZF˙2˙1˙L7 int˙ord˙transl˙inv by simp
  ultimately have s(0) - δ(s,m,-m) ≤ abs(s(0)) + maxδ(s)
    by (rule Int˙order˙transitive)
  with T have
    s(0) - δ(s,m,-m) - s(m) ≤ abs(s(0)) + maxδ(s) - s(m)
    using int˙ord˙transl˙inv by simp
  with I show s(-m) ≤ abs(s(0)) + maxδ(s) - s(m)
    by simp
qed
```

An identity that expresses the value of an integer function at the opposite integer in terms of the value of that function at the integer, zero, and the homomorphism difference. We have a similar identity in Int˙ZF˙2˙1˙L14, but over there we assume that f is a slope.

```
lemma (in int1) Int˙ZF˙2˙1˙L14A: assumes A1: f:ℤ→ℤ and A2: m∈ℤ
  shows f(-m) = (-δ(f,m,-m)) + f(0) - f(m)
proof -
  from A1 A2 have T:
    f(-m) ∈ ℤ  δ(f,m,-m) ∈ ℤ  f(0) ∈ ℤ  f(m) ∈ ℤ
    using Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙1˙L5 int˙zero˙one˙are˙int apply˙funtype
    by auto
   with A2 show f(-m) = (-δ(f,m,-m)) + f(0) - f(m)
     using Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙2˙L15 by simp
qed
```

The next lemma allows to use the expression maxf(f,**0**..M-1). Recall that maxf(f,A) is the maximum of (function) f on (the set) A.

```
lemma (in int1) Int˙ZF˙2˙1˙L15:
  assumes s∈𝒮 and M ∈ ℤ₊
  shows
  maxf(s,0..(M-1)) ∈ ℤ
  ∀n ∈ 0..(M-1). s(n) ≤ maxf(s,0..(M-1))
  minf(s,0..(M-1)) ∈ ℤ
  ∀n ∈ 0..(M-1). minf(s,0..(M-1)) ≤ s(n)
  using assms AlmostHoms˙def Int˙ZF˙1˙5˙L6 Int˙ZF˙1˙4˙L2
  by auto
```

A lower estimate for the value of a slope at $nM + k$.

```
lemma (in int1) Int˙ZF˙2˙1˙L16:
  assumes A1: s∈𝒮  and A2: m∈ℤ and A3: M ∈ ℤ₊ and A4: k ∈ 0..(M-1)
  shows s(m·M) + (minf(s,0..(M-1))- maxδ(s)) ≤ s(m·M+k)
proof -
```

```
  from A3 have 0..(M-1) ⊆ ℤ
    using Int˙ZF˙1˙5˙L6 by simp
  with A1 A2 A3 A4 have T: m·M ∈ ℤ   k ∈ ℤ  s(m·M) ∈ ℤ
    using PositiveSet˙def Int˙ZF˙1˙1˙L5  Int˙ZF˙2˙1˙L2B
    by auto
  with A1 A3 A4 have
    s(m·M) + (minf(s,0..(M-1)) - maxδ(s)) ≤ s(m·M) + (s(k) + δ(s,m·M,k))
    using Int˙ZF˙2˙1˙L15 Int˙ZF˙2˙1˙L7 int˙ineq˙add˙sides int˙ord˙transl˙inv
    by simp
  with A1 T show thesis using Int˙ZF˙2˙1˙L3A by simp
qed
```

Identity is a slope.

```
lemma (in int1) Int˙ZF˙2˙1˙L17: shows id(ℤ) ∈ 𝒮
  using Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙4˙L15 by simp
```

Simple identities about (absolute value of) homomorphism differences.

```
lemma (in int1) Int˙ZF˙2˙1˙L18:
  assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ
  shows
  abs(f(n) + f(m) - f(m+n)) = abs(δ(f,m,n))
  abs(f(m) + f(n) - f(m+n)) = abs(δ(f,m,n))
  (-(f(m))) - f(n) + f(m+n) = δ(f,m,n)
  (-(f(n))) - f(m) + f(m+n) = δ(f,m,n)
  abs((-f(m+n)) + f(m) + f(n)) = abs(δ(f,m,n))
proof -
  from A1 A2 have T:
    f(m+n) ∈ ℤ  f(m) ∈ ℤ  f(n) ∈ ℤ
    f(m+n) - f(m) -  f(n)  ∈ ℤ
    (-(f(m))) ∈ ℤ
    (-f(m+n)) + f(m) + f(n) ∈ ℤ
    using apply˙funtype Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙1˙L5 by auto
  then have
    abs(-(f(m+n) - f(m) -  f(n))) = abs(f(m+n) - f(m) -  f(n))
    using Int˙ZF˙2˙L17 by simp
  moreover from T have
    (-(f(m+n) - f(m) -  f(n))) = f(n) + f(m) - f(m+n)
    using Int˙ZF˙1˙2˙L9A by simp
  ultimately show abs(f(n) + f(m) - f(m+n)) = abs(δ(f,m,n))
    by simp
  moreover from T have f(n) + f(m) = f(m) + f(n)
    using Int˙ZF˙1˙1˙L5 by simp
  ultimately show abs(f(m) + f(n) - f(m+n)) = abs(δ(f,m,n))
    by simp
  from T show
    (-(f(m))) - f(n) + f(m+n) = δ(f,m,n)
    (-(f(n))) - f(m) + f(m+n) = δ(f,m,n)
    using Int˙ZF˙1˙2˙L9 by auto
  from T have
```

```
    abs((-f(m+n)) + f(m) + f(n)) =
    abs(-((-f(m+n)) + f(m) + f(n)))
    using Int˙ZF˙2˙L17 by simp
  also from T have
    abs(-((-f(m+n)) + f(m) + f(n))) = abs(δ(f,m,n))
    using Int˙ZF˙1˙2˙L9 by simp
  finally show abs((-f(m+n)) + f(m) + f(n)) = abs(δ(f,m,n))
    by simp
qed
```

Some identities about the homomorphism difference of odd functions.

```
lemma (in int1) Int˙ZF˙2˙1˙L19:
  assumes A1: f:ℤ→ℤ and A2: ∀x∈ℤ. (-f(-x)) = f(x)
  and A3: m∈ℤ  n∈ℤ
  shows
  abs(δ(f,-m,m+n)) = abs(δ(f,m,n))
  abs(δ(f,-n,m+n)) = abs(δ(f,m,n))
  δ(f,n,-(m+n)) = δ(f,m,n)
  δ(f,m,-(m+n)) = δ(f,m,n)
  abs(δ(f,-m,-n)) = abs(δ(f,m,n))
proof -
  from A1 A2 A3 show
    abs(δ(f,-m,m+n)) = abs(δ(f,m,n))
    abs(δ(f,-n,m+n)) = abs(δ(f,m,n))
    using Int˙ZF˙1˙2˙L3 Int˙ZF˙2˙1˙L18 by auto
  from A3 have T: m+n ∈ ℤ using Int˙ZF˙1˙1˙L5 by simp
  from A1 A2 have I: ∀x∈ℤ. f(-x) = (-f(x))
    using Int˙ZF˙1˙5˙L13 by simp
  with A1 A2 A3 T show
    δ(f,n,-(m+n)) = δ(f,m,n)
    δ(f,m,-(m+n)) = δ(f,m,n)
    using Int˙ZF˙1˙2˙L3 Int˙ZF˙2˙1˙L18 by auto
  from A3 have
    abs(δ(f,-m,-n)) = abs(f(-(m+n)) - f(-m) - f(-n))
    using Int˙ZF˙1˙1˙L5 by simp
  also from A1 A2 A3 T I have ... = abs(δ(f,m,n))
    using Int˙ZF˙2˙1˙L18 by simp
  finally show abs(δ(f,-m,-n)) = abs(δ(f,m,n)) by simp
qed
```

Recall that f is a slope iff $f(m+n)-f(m)-f(n)$ is bounded as m,n ranges over integers. The next lemma is the first step in showing that we only need to check this condition as m,n ranges over positive intergers. Namely we show that if the condition holds for positive integers, then it holds if one integer is positive and the second one is nonnegative.

```
lemma (in int1) Int˙ZF˙2˙1˙L20: assumes A1: f:ℤ→ℤ and
  A2: ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L and
  A3:  m∈ℤ⁺  n∈ℤ₊
  shows
```

```
  0 ≤ L
  abs(δ(f,m,n)) ≤ L + abs(f(0))
proof -
  from A1 A2 have
    δ(f,1,1) ∈ ℤ  and abs(δ(f,1,1)) ≤ L
    using int˙one˙two˙are˙pos PositiveSet˙def Int˙ZF˙2˙1˙L3B
    by auto
  then show I: 0 ≤ L using Int˙ZF˙1˙3˙L19 by simp
  from A1 A3 have T:
    n ∈ ℤ  f(n) ∈ ℤ  f(0) ∈ ℤ
    δ(f,m,n) ∈ ℤ  abs(δ(f,m,n)) ∈ ℤ
    using PositiveSet˙def int˙zero˙one˙are˙int apply˙funtype
      Nonnegative˙def Int˙ZF˙2˙1˙L3B Int˙ZF˙2˙L14 by auto
  from A3 have m=0 ∨ m∈ℤ+ using Int˙ZF˙1˙5˙L3A by auto
  moreover
  { assume m = 0
    with T I have abs(δ(f,m,n)) ≤ L + abs(f(0))
      using Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙2˙L3 Int˙ZF˙2˙L17
 int˙ord˙is˙refl refl˙def Int˙ZF˙2˙L15F by simp }
  moreover
  { assume m∈ℤ+
    with A2 A3 T have abs(δ(f,m,n)) ≤ L + abs(f(0))
      using int˙abs˙nonneg Int˙ZF˙2˙L15F by simp }
   ultimately show abs(δ(f,m,n)) ≤ L + abs(f(0))
     by auto
qed
```

If the slope condition holds for all pairs of integers such that one integer is positive and the second one is nonnegative, then it holds when both integers are nonnegative.

```
lemma (in int1) Int˙ZF˙2˙1˙L21: assumes A1: f:ℤ→ℤ and
  A2: ∀a∈ℤ+. ∀b∈ℤ+. abs(δ(f,a,b)) ≤ L and
  A3: n∈ℤ+  m∈ℤ+
  shows abs(δ(f,m,n)) ≤ L + abs(f(0))
proof -
  from A1 A2 have
    δ(f,1,1) ∈ ℤ  and abs(δ(f,1,1)) ≤ L
    using int˙one˙two˙are˙pos PositiveSet˙def Nonnegative˙def Int˙ZF˙2˙1˙L3B
    by auto
  then have I: 0 ≤ L using Int˙ZF˙1˙3˙L19 by simp
  from A1 A3 have T:
    m ∈ ℤ  f(m) ∈ ℤ  f(0) ∈ ℤ  (-f(0)) ∈ ℤ
    δ(f,m,n) ∈ ℤ  abs(δ(f,m,n)) ∈ ℤ
    using int˙zero˙one˙are˙int apply˙funtype Nonnegative˙def
      Int˙ZF˙2˙1˙L3B Int˙ZF˙2˙L14 Int˙ZF˙1˙1˙L4 by auto
  from A3 have n=0 ∨ n∈ℤ+ using Int˙ZF˙1˙5˙L3A by auto
  moreover
  { assume n=0
     with T have δ(f,m,n) = -f(0)
```

```
    using Int˙ZF˙1˙1˙L4 by simp
  with T have abs(δ(f,m,n)) = abs(f(0))
    using Int˙ZF˙2˙L17 by simp
  with T have abs(δ(f,m,n)) ≤ abs(f(0))
    using int˙ord˙is˙refl refl˙def by simp
  with T I have abs(δ(f,m,n)) ≤ L + abs(f(0))
    using Int˙ZF˙2˙L15F by simp }
 moreover
 { assume n∈ℤ+
  with A2 A3 T have abs(δ(f,m,n)) ≤ L + abs(f(0))
    using int˙abs˙nonneg Int˙ZF˙2˙L15F by simp }
 ultimately show  abs(δ(f,m,n)) ≤ L + abs(f(0))
  by auto
qed
```

If the homomorphism difference is bounded on $\mathbb{Z}_+\times\mathbb{Z}_+$, then it is bounded on $\mathbb{Z}^+\times\mathbb{Z}^+$.

```
lemma (in int1) Int˙ZF˙2˙1˙L22: assumes A1: f:ℤ→ℤ and
 A2: ∀a∈ℤ+. ∀b∈ℤ+. abs(δ(f,a,b)) ≤ L
 shows ∃M. ∀m∈ℤ⁺. ∀n∈ℤ⁺. abs(δ(f,m,n)) ≤ M
proof -
 from A1 A2 have
  ∀m∈ℤ⁺. ∀n∈ℤ⁺. abs(δ(f,m,n)) ≤ L + abs(f(0)) + abs(f(0))
  using Int˙ZF˙2˙1˙L20 Int˙ZF˙2˙1˙L21 by simp
 then show thesis by auto
qed
```

For odd functions we can do better than in Int˙ZF˙2˙1˙L22: if the homomorphism difference of f is bounded on $\mathbb{Z}^+\times\mathbb{Z}^+$, then it is bounded on $\mathbb{Z}\times\mathbb{Z}$, hence f is a slope. Loong prof by splitting the $\mathbb{Z}\times\mathbb{Z}$ into six subsets.

```
lemma (in int1) Int˙ZF˙2˙1˙L23: assumes A1: f:ℤ→ℤ and
 A2: ∀a∈ℤ+. ∀b∈ℤ+. abs(δ(f,a,b)) ≤ L
 and A3: ∀x∈ℤ. (-f(-x)) = f(x)
 shows  f∈𝒮
proof -
 from A1 A2 have
  ∃M.∀a∈ℤ⁺. ∀b∈ℤ⁺. abs(δ(f,a,b)) ≤ M
  by (rule Int˙ZF˙2˙1˙L22)
 then obtain M where I: ∀m∈ℤ⁺. ∀n∈ℤ⁺. abs(δ(f,m,n)) ≤ M
  by auto
 { fix a b assume A4: a∈ℤ  b∈ℤ
  then have
   0≤a ∧ 0≤b  ∨  a≤0 ∧ b≤0  ∨
   a≤0 ∧ 0≤b ∧ 0 ≤ a+b  ∨ a≤0 ∧ 0≤b ∧ a+b ≤ 0  ∨
   0≤a ∧ b≤0 ∧ 0 ≤ a+b  ∨  0≤a ∧ b≤0 ∧ a+b ≤ 0
   using int˙plane˙split˙in6 by simp
  moreover
  { assume 0≤a ∧ 0≤b
   then have a∈ℤ⁺  b∈ℤ⁺
```

```
using Int˙ZF˙2˙L16 by auto
    with I have abs(δ(f,a,b)) ≤ M by simp }
  moreover
  { assume a≤0 ∧ b≤0
    with I have abs(δ(f,-a,-b)) ≤ M
using Int˙ZF˙2˙L10A Int˙ZF˙2˙L16 by simp
    with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
using Int˙ZF˙2˙1˙L19 by simp }
  moreover
  { assume a≤0 ∧ 0≤b ∧ 0 ≤ a+b
    with I have abs(δ(f,-a,a+b)) ≤ M
using Int˙ZF˙2˙L10A Int˙ZF˙2˙L16 by simp
    with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
using Int˙ZF˙2˙1˙L19 by simp }
  moreover
  { assume a≤0 ∧ 0≤b ∧ a+b ≤ 0
    with I have abs(δ(f,b,-(a+b))) ≤ M
using Int˙ZF˙2˙L10A Int˙ZF˙2˙L16 by simp
    with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
using Int˙ZF˙2˙1˙L19 by simp }
  moreover
  { assume 0≤a ∧ b≤0 ∧ 0 ≤ a+b
    with I have abs(δ(f,-b,a+b)) ≤ M
using Int˙ZF˙2˙L10A Int˙ZF˙2˙L16 by simp
    with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
using Int˙ZF˙2˙1˙L19 by simp }
  moreover
  { assume 0≤a ∧ b≤0 ∧ a+b ≤ 0
    with I have abs(δ(f,a,-(a+b))) ≤ M
using Int˙ZF˙2˙L10A Int˙ZF˙2˙L16 by simp
    with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
using Int˙ZF˙2˙1˙L19 by simp }
  ultimately have abs(δ(f,a,b)) ≤ M by auto }
 then have ∀m∈ℤ. ∀n∈ℤ. abs(δ(f,m,n)) ≤ M by simp
 with A1 show f∈𝒮 by (rule Int˙ZF˙2˙1˙L5)
qed
```

If the homomorphism difference of a function defined on positive integers is bounded, then the odd extension of this function is a slope.

```
lemma (in int1) Int˙ZF˙2˙1˙L24:
 assumes A1: f:ℤ+→ℤ and A2: ∀a∈ℤ+. ∀b∈ℤ+. abs(δ(f,a,b)) ≤ L
 shows OddExtension(ℤ,IntegerAddition,IntegerOrder,f) ∈ 𝒮
proof -
 let g = OddExtension(ℤ,IntegerAddition,IntegerOrder,f)
 from A1 have g : ℤ→ℤ
   using Int˙ZF˙1˙5˙L10 by simp
 moreover have ∀a∈ℤ+. ∀b∈ℤ+. abs(δ(g,a,b)) ≤ L
 proof -
   { fix a b assume A3: a∈ℤ+  b∈ℤ+
```

```
      with A1 have abs(δ(f,a,b)) =  abs(δ(g,a,b))
 using pos˙int˙closed˙add˙unfolded Int˙ZF˙1˙5˙L11
 by simp
      moreover from A2 A3 have abs(δ(f,a,b)) ≤ L by simp
      ultimately have abs(δ(g,a,b)) ≤ L by simp
    } then show thesis by simp
  qed
  moreover from A1 have ∀x∈ℤ. (-g(-x)) = g(x)
    using int˙oddext˙is˙odd˙alt by simp
  ultimately show g ∈ 𝒮 by (rule Int˙ZF˙2˙1˙L23)
qed
```

Type information related to γ.

```
lemma (in int1) Int˙ZF˙2˙1˙L25:
  assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ
  shows
  δ(f,m,-n) ∈ ℤ
  δ(f,n,-n) ∈ ℤ
  (-δ(f,n,-n)) ∈ ℤ
  f(𝟎) ∈ ℤ
  γ(f,m,n)  ∈ ℤ
proof -
  from A1 A2 show T1:
    δ(f,m,-n) ∈ ℤ  f(𝟎) ∈ ℤ
    using Int˙ZF˙1˙1˙L4 Int˙ZF˙2˙1˙L3B int˙zero˙one˙are˙int apply˙funtype
    by auto
  from A2 have (-n) ∈ ℤ
    using Int˙ZF˙1˙1˙L4 by simp
  with A1 A2 show δ(f,n,-n) ∈ ℤ
    using Int˙ZF˙2˙1˙L3B by simp
  then show (-δ(f,n,-n)) ∈ ℤ
    using Int˙ZF˙1˙1˙L4 by simp
  with T1 show γ(f,m,n)  ∈ ℤ
    using Int˙ZF˙1˙1˙L5 by simp
qed
```

A couple of formulae involving $f(m-n)$ and $\gamma(f,m,n)$.

```
lemma (in int1) Int˙ZF˙2˙1˙L26:
  assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ
  shows
  f(m-n) = γ(f,m,n) + f(m) - f(n)
  f(m-n) = γ(f,m,n) + (f(m) - f(n))
  f(m-n) + (f(n) - γ(f,m,n)) = f(m)
proof -
  from A1 A2 have T:
    (-n) ∈ ℤ  δ(f,m,-n) ∈ ℤ
    f(𝟎) ∈ ℤ  f(m) ∈ ℤ  f(n) ∈ ℤ  (-f(n)) ∈ ℤ
    (-δ(f,n,-n)) ∈ ℤ
    (-δ(f,n,-n))  + f(𝟎) ∈ ℤ
```

```
    γ(f,m,n) ∈ ℤ
    using  Int˙ZF˙1˙1˙L4 Int˙ZF˙2˙1˙L25 apply˙funtype Int˙ZF˙1˙1˙L5
    by auto
   with A1 A2 have f(m-n) =
    δ(f,m,-n) + ((-δ(f,n,-n)) + f(0) - f(n)) + f(m)
    using Int˙ZF˙2˙1˙L3C Int˙ZF˙2˙1˙L14A by simp
  with T have f(m-n) =
    δ(f,m,-n) + ((-δ(f,n,-n)) + f(0)) + f(m) - f(n)
    using Int˙ZF˙1˙2˙L16 by simp
  moreover from T have
    δ(f,m,-n) + ((-δ(f,n,-n)) + f(0)) = γ(f,m,n)
    using Int˙ZF˙1˙1˙L7 by simp
  ultimately show  I: f(m-n) = γ(f,m,n) + f(m) - f(n)
    by simp
  then have f(m-n) + (f(n) - γ(f,m,n)) =
    (γ(f,m,n) + f(m) - f(n)) + (f(n) - γ(f,m,n))
    by simp
  moreover from T have ... = f(m) using Int˙ZF˙1˙2˙L18
    by simp
  ultimately show f(m-n) + (f(n) - γ(f,m,n)) = f(m)
    by simp
  from T have γ(f,m,n) ∈ ℤ  f(m) ∈ ℤ  (-f(n)) ∈ ℤ
    by auto
  then have
    γ(f,m,n) + f(m) + (-f(n)) =  γ(f,m,n) + (f(m) + (-f(n)))
    by (rule Int˙ZF˙1˙1˙L7)
  with I show  f(m-n) = γ(f,m,n) + (f(m) - f(n)) by simp
qed
```

A formula expressing the difference between $f(m-n-k)$ and $f(m)-f(n)-f(k)$ in terms of γ.

```
lemma (in int1) Int˙ZF˙2˙1˙L26A:
  assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ  k∈ℤ
  shows
  f(m-n-k) - (f(m)- f(n) - f(k)) = γ(f,m-n,k) + γ(f,m,n)
proof -
  from A1 A2 have
    T: m-n ∈ ℤ γ(f,m-n,k) ∈ ℤ  f(m) - f(n) - f(k) ∈ ℤ and
    T1: γ(f,m,n) ∈ ℤ  f(m) - f(n) ∈ ℤ  (-f(k)) ∈ ℤ
    using Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙1˙L5 Int˙ZF˙2˙1˙L25 apply˙funtype
    by auto
  from A1 A2 have
    f(m-n) - f(k) = γ(f,m,n) + (f(m) - f(n)) + (-f(k))
    using Int˙ZF˙2˙1˙L26 by simp
  also from T1 have ... = γ(f,m,n) + (f(m) - f(n) + (-f(k)))
    by (rule Int˙ZF˙1˙1˙L7)
  finally have
    f(m-n) - f(k) = γ(f,m,n) + (f(m) - f(n) - f(k))
    by simp
```

```
  moreover from A1 A2 T have
    f(m-n-k) =  γ(f,m-n,k) + (f(m-n)-f(k))
    using Int˙ZF˙2˙1˙L26 by simp
  ultimately have
    f(m-n-k) - (f(m)- f(n) - f(k)) =
    γ(f,m-n,k) + ( γ(f,m,n) + (f(m) - f(n) - f(k)))
    - (f(m)- f(n) - f(k))
    by simp
  with T T1 show thesis
    using Int˙ZF˙1˙2˙L17 by simp
qed
```

If s is a slope, then $\gamma(s, m, n)$ is uniformly bounded.

```
lemma (in int1) Int˙ZF˙2˙1˙L27: assumes A1: s∈𝒮
  shows ∃L∈ℤ. ∀m∈ℤ.∀n∈ℤ. abs(γ(s,m,n)) ≤ L
proof -
  let L = maxδ(s) + maxδ(s) + abs(s(𝟎))
  from A1 have T:
    maxδ(s) ∈ ℤ  abs(s(𝟎)) ∈ ℤ  L ∈ ℤ
    using Int˙ZF˙2˙1˙L8 int˙zero˙one˙are˙int Int˙ZF˙2˙1˙L2B
      Int˙ZF˙2˙L14 Int˙ZF˙1˙1˙L5 by auto
  moreover
  { fix m
    fix n
    assume A2: m∈ℤ  n∈ℤ
    with A1 have T:
      (-n) ∈ ℤ
      δ(s,m,-n) ∈ ℤ
      δ(s,n,-n) ∈ ℤ
      (-δ(s,n,-n)) ∈ ℤ
      s(𝟎) ∈ ℤ  abs(s(𝟎)) ∈ ℤ
      using Int˙ZF˙1˙1˙L4 AlmostHoms˙def Int˙ZF˙2˙1˙L25 Int˙ZF˙2˙L14
      by auto
    with T have
      abs(δ(s,m,-n) - δ(s,n,-n) + s(𝟎)) ≤
      abs(δ(s,m,-n)) + abs(-δ(s,n,-n)) + abs(s(𝟎))
      using Int˙triangle˙ineq3 by simp
    moreover from A1 A2 T have
      abs(δ(s,m,-n)) + abs(-δ(s,n,-n)) + abs(s(𝟎)) ≤ L
      using Int˙ZF˙2˙1˙L7 int˙ineq˙add˙sides int˙ord˙transl˙inv Int˙ZF˙2˙L17
      by simp
   ultimately have abs(δ(s,m,-n) - δ(s,n,-n) + s(𝟎)) ≤ L
      by (rule Int˙order˙transitive)
    then have abs(γ(s,m,n)) ≤ L by simp }
  ultimately show ∃L∈ℤ. ∀m∈ℤ.∀n∈ℤ. abs(γ(s,m,n)) ≤ L
    by auto
qed
```

If s is a slope, then $s(m) \leq s(m-1) + M$, where L does not depend on m.

```
lemma (in int1) Int˙ZF˙2˙1˙L28: assumes A1: s∈𝒮
  shows ∃M∈ℤ. ∀m∈ℤ. s(m) ≤ s(m-𝟏) + M
proof -
  from A1 have
    ∃L∈ℤ. ∀m∈ℤ.∀n∈ℤ.abs(γ(s,m,n)) ≤ L
    using Int˙ZF˙2˙1˙L27 by simp
  then obtain L where T: L∈ℤ and ∀m∈ℤ.∀n∈ℤ.abs(γ(s,m,n)) ≤ L
    using Int˙ZF˙2˙1˙L27 by auto
  then have I: ∀m∈ℤ.abs(γ(s,m,𝟏)) ≤ L
    using int˙zero˙one˙are˙int by simp
  let M = s(𝟏) + L
  from A1 T have M ∈ ℤ
    using int˙zero˙one˙are˙int Int˙ZF˙2˙1˙L2B Int˙ZF˙1˙1˙L5
    by simp
  moreover
  { fix m assume A2: m∈ℤ
    with A1 have
      T1: s:ℤ→ℤ  m∈ℤ  𝟏∈ℤ and
      T2: γ(s,m,𝟏) ∈ ℤ  s(𝟏) ∈ ℤ
      using int˙zero˙one˙are˙int AlmostHoms˙def
 Int˙ZF˙2˙1˙L25 by auto
    from A2 T1 have T3: s(m-𝟏) ∈ ℤ
      using Int˙ZF˙1˙1˙L5 apply˙funtype by simp
    from I A2 T2 have
      (-γ(s,m,𝟏)) ≤ abs(γ(s,m,𝟏))
      abs(γ(s,m,𝟏)) ≤ L
      using Int˙ZF˙2˙L19C by auto
    then have (-γ(s,m,𝟏)) ≤ L
      by (rule Int˙order˙transitive)
    with T2 T3 have
      s(m-𝟏) + (s(𝟏) - γ(s,m,𝟏)) ≤ s(m-𝟏) + M
      using int˙ord˙transl˙inv by simp
    moreover from T1 have
      s(m-𝟏) + (s(𝟏) - γ(s,m,𝟏)) = s(m)
      by (rule Int˙ZF˙2˙1˙L26)
    ultimately have s(m) ≤ s(m-𝟏) + M  by simp  }
  ultimately show ∃M∈ℤ. ∀m∈ℤ. s(m) ≤ s(m-𝟏) + M
    by auto
qed
```

If s is a slope, then the difference between $s(m-n-k)$ and $s(m)-s(n)-s(k)$ is uniformly bounded.

```
lemma (in int1) Int˙ZF˙2˙1˙L29: assumes A1: s∈𝒮
  shows
  ∃M∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. abs(s(m-n-k) - (s(m)-s(n)-s(k))) ≤M
proof -
  from A1 have ∃L∈ℤ. ∀m∈ℤ.∀n∈ℤ. abs(γ(s,m,n)) ≤ L
    using Int˙ZF˙2˙1˙L27 by simp
  then obtain L where I: L∈ℤ and
```

```
    II: ∀m∈ℤ.∀n∈ℤ. abs(γ(s,m,n)) ≤ L
    by auto
  from I have L+L ∈ ℤ
    using Int˙ZF˙1˙1˙L5 by simp
  moreover
  { fix m n k assume A2: m∈ℤ  n∈ℤ  k∈ℤ
    with A1 have T:
      m-n ∈ ℤ  γ(s,m-n,k) ∈ ℤ  γ(s,m,n) ∈ ℤ
      using Int˙ZF˙1˙1˙L5 AlmostHoms˙def Int˙ZF˙2˙1˙L25
      by auto
    then have
      I: abs(γ(s,m-n,k) + γ(s,m,n)) ≤ abs(γ(s,m-n,k)) + abs(γ(s,m,n))
      using Int˙triangle˙ineq by simp
    from II A2 T have
      abs(γ(s,m-n,k)) ≤ L
      abs(γ(s,m,n)) ≤ L
      by auto
    then have abs(γ(s,m-n,k)) + abs(γ(s,m,n)) ≤ L+L
      using int˙ineq˙add˙sides by simp
    with I have abs(γ(s,m-n,k) + γ(s,m,n)) ≤ L+L
      by (rule Int˙order˙transitive)
    moreover from A1 A2 have
      s(m-n-k) - (s(m)- s(n) - s(k)) = γ(s,m-n,k) + γ(s,m,n)
      using AlmostHoms˙def Int˙ZF˙2˙1˙L26A by simp
    ultimately have
      abs(s(m-n-k) - (s(m)- s(n) - s(k))) ≤ L+L
      by simp }
  ultimately show thesis by auto
qed
```

If s is a slope, then we can find integers M, K such that $s(m-n-k) \leq s(m) - s(n) - s(k) + M$ and $s(m) - s(n) - s(k) + K \leq s(m-n-k)$, for all integer m, n, k.

```
lemma (in int1) Int˙ZF˙2˙1˙L30: assumes A1: s∈𝒮
  shows
  ∃M∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m-n-k) ≤ s(m)-s(n)-s(k)+M
  ∃K∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m)-s(n)-s(k)+K ≤ s(m-n-k)
proof -
  from A1 have
    ∃M∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. abs(s(m-n-k) - (s(m)-s(n)-s(k))) ≤M
    using Int˙ZF˙2˙1˙L29 by simp
  then obtain M where I: M∈ℤ and II:
    ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. abs(s(m-n-k) - (s(m)-s(n)-s(k))) ≤M
    by auto
  from I have III: (-M) ∈ ℤ using Int˙ZF˙1˙1˙L4 by simp
  { fix m n k assume A2: m∈ℤ  n∈ℤ  k∈ℤ
    with A1 have s(m-n-k) ∈ ℤ  and s(m)-s(n)-s(k) ∈ ℤ
      using Int˙ZF˙1˙1˙L5 Int˙ZF˙2˙1˙L2B by auto
    moreover from II A2 have
```

```
    abs(s(m-n-k) - (s(m)-s(n)-s(k))) ≤M
    by simp
  ultimately have
    s(m-n-k) ≤ s(m)-s(n)-s(k)+M ∧
    s(m)-s(n)-s(k) - M ≤ s(m-n-k)
    using Int˙triangle˙ineq2 by simp
 } then have
    ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m-n-k) ≤ s(m)-s(n)-s(k)+M
    ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m)-s(n)-s(k) - M ≤ s(m-n-k)
  by auto
 with I III show
  ∃M∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m-n-k) ≤ s(m)-s(n)-s(k)+M
  ∃K∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m)-s(n)-s(k)+K ≤ s(m-n-k)
  by auto
qed
```

By definition functions f, g are almost equal if $f - g^*$ is bounded. In the next lemma we show it is sufficient to check the boundedness on positive integers.

```
lemma (in int1) Int˙ZF˙2˙1˙L31: assumes A1: s∈𝒮  r∈𝒮
 and A2: ∀m∈ℤ₊. abs(s(m)-r(m)) ≤ L
 shows s ∼ r
proof -
 let a = abs(s(𝟎) - r(𝟎))
 let c = 𝟐·maxδ(s) + 𝟐·maxδ(r) + L
 let M = Maximum(IntegerOrder,–a,L,c″)
 from A2 have abs(s(𝟏)-r(𝟏)) ≤ L
   using int˙one˙two˙are˙pos by simp
 then have T: L∈ℤ using Int˙ZF˙2˙L1A by simp
 moreover from A1 have a ∈ ℤ
   using int˙zero˙one˙are˙int Int˙ZF˙2˙1˙L2B
     Int˙ZF˙1˙1˙L5 Int˙ZF˙2˙L14 by simp
 moreover from A1 T have c ∈ ℤ
   using Int˙ZF˙2˙1˙L8 int˙two˙three˙are˙int Int˙ZF˙1˙1˙L5
   by simp
 ultimately have
   I: a ≤ M and
   II: L ≤ M and
   III: c ≤ M
   using Int˙ZF˙1˙4˙L1A by auto

 { fix m assume A5: m∈ℤ
   with A1 have T:
     s(m) ∈ ℤ  r(m) ∈ ℤ  s(m) - r(m) ∈ ℤ
     s(-m) ∈ ℤ  r(-m) ∈ ℤ
     using Int˙ZF˙2˙1˙L2B Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙1˙L5
     by auto
   from A5 have m=𝟎 ∨ m∈ℤ₊ ∨ (-m) ∈ ℤ₊
     using int˙decomp˙cases by simp
```

```
    moreover
    { assume m=0
      with I have abs(s(m) - r(m)) ≤ M
 by simp }
    moreover
    { assume m∈ℤ+
      with A2 II have
 abs(s(m)-r(m)) ≤ L and L≤M
 by auto
      then have abs(s(m)-r(m)) ≤ M
 by (rule Int˙order˙transitive) }
    moreover
    { assume A6: (-m) ∈ ℤ+
      from T have abs(s(m)-r(m)) ≤
 abs(s(m)+s(-m)) + abs(r(m)+r(-m)) + abs(s(-m)-r(-m))
 using Int˙ZF˙1˙3˙L22A by simp
      moreover
      from A1 A2 III A5 A6 have
 abs(s(m)+s(-m)) + abs(r(m)+r(-m)) + abs(s(-m)-r(-m)) ≤ c
 c ≤ M
 using Int˙ZF˙2˙1˙L14 int˙ineq˙add˙sides by auto
      then have
 abs(s(m)+s(-m)) + abs(r(m)+r(-m)) + abs(s(-m)-r(-m)) ≤ M
 by (rule Int˙order˙transitive)
      ultimately have  abs(s(m)-r(m)) ≤ M
 by (rule Int˙order˙transitive) }
    ultimately have abs(s(m) - r(m)) ≤ M
      by auto
  } then have ∀m∈ℤ. abs(s(m)-r(m)) ≤ M
    by simp
  with A1 show s ∼ r by (rule Int˙ZF˙2˙1˙L9)
qed
```

A sufficient condition for an odd slope to be almost equal to identity: If for all positive integers the value of the slope at m is between m and m plus some constant independent of m, then the slope is almost identity.

```
lemma (in int1) Int˙ZF˙2˙1˙L32: assumes A1: s∈𝒮  M∈ℤ
  and A2: ∀m∈ℤ+. m ≤ s(m) ∧ s(m) ≤ m+M
  shows s ∼ id(ℤ)
proof -
  let r = id(ℤ)
  from A1 have s∈𝒮  r ∈ 𝒮
    using Int˙ZF˙2˙1˙L17 by auto
  moreover from A1 A2 have ∀m∈ℤ+. abs(s(m)-r(m)) ≤ M
    using Int˙ZF˙1˙3˙L23 PositiveSet˙def id˙conv by simp
  ultimately show s ∼ id(ℤ) by (rule Int˙ZF˙2˙1˙L31)
qed
```

A lemma about adding a constant to slopes. This is actually proven in

Group˙ZF˙3˙5˙L1, in Group˙ZF˙3.thy here we just refer to that lemma to show it in notation used for integers. Unfortunately we have to use raw set notation in the proof.

```
lemma (in int1) Int˙ZF˙2˙1˙L33:
  assumes A1: s∈𝒮 and A2: c∈ℤ and
  A3: r = −⟨m,s(m)+c⟩. m∈ℤ″
  shows
  ∀m∈ℤ. r(m) = s(m)+c
  r∈𝒮
  s ∼ r
proof -
  let G = ℤ
  let f = IntegerAddition
  let AH = AlmostHoms(G, f)
  from assms have I:
    group1(G, f)
    s ∈ AlmostHoms(G, f)
    c ∈ G
    r = −⟨x, f⟨s(x), c⟩⟩. x ∈ G″
    using Int˙ZF˙2˙1˙L1 by auto
  then have ∀x∈G. r(x) = f⟨s(x),c⟩
    by (rule group1.Group˙ZF˙3˙5˙L1)
  moreover from I have r ∈ AlmostHoms(G, f)
    by (rule group1.Group˙ZF˙3˙5˙L1)
  moreover from I have
     ⟨s, r⟩ ∈ QuotientGroupRel(AlmostHoms(G, f), AlHomOp1(G, f), FinRange-
Functions(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
```

45.2 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 $|f(p \cdot q) - p \cdot f(q)| \leq (|p| + 1)\cdot\max\delta(f)$. The proof is by induction on p and the next lemma is the induction step for the case when $0 \leq p$.

```
lemma (in int1) Int˙ZF˙2˙2˙L2:
  assumes A1: f∈𝒮 and A2: 0≤p  q∈ℤ
  and A3: abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
  shows
  abs(f((p+1)·q)-(p+1)·f(q)) ≤ (abs(p+1)+ 1)·maxδ(f)
proof -
```

```
  from A2 have q∈ℤ  p·q ∈ ℤ
    using Int˙ZF˙2˙L1A Int˙ZF˙1˙1˙L5 by auto
  with A1 have I: abs(δ(f,p·q,q)) ≤ maxδ(f) by (rule Int˙ZF˙2˙1˙L7)
  moreover note A3
  moreover from A1 A2 have
    abs(f((p+1)·q)-(p+1)·f(q)) ≤ abs(δ(f,p·q,q))+abs(f(p·q)-p·f(q))
    using AlmostHoms˙def Int˙ZF˙2˙L1A Int˙ZF˙2˙2˙L1 by simp
  ultimately have
    abs(f((p+1)·q)-(p+1)·f(q)) ≤ maxδ(f)+(abs(p)+1)·maxδ(f)
    by (rule Int˙ZF˙2˙L15)
  moreover from I A2 have
    maxδ(f)+(abs(p)+1)·maxδ(f) = (abs(p+1)+ 1)·maxδ(f)
    using Int˙ZF˙2˙L1A Int˙ZF˙1˙2˙L2 by simp
  ultimately show
    abs(f((p+1)·q)-(p+1)·f(q)) ≤ (abs(p+1)+ 1)·maxδ(f)
    by simp
qed
```

If f is a slope, then $|f(p \cdot q) - p \cdot f(q)| \leq (|p| + 1)\cdot\max\delta$. The proof is by induction on p and the next lemma is the induction step for the case when $p \leq 0$.

```
lemma (in int1) Int˙ZF˙2˙2˙L3:
  assumes A1: f∈𝒮 and A2: p≤0  q∈ℤ
  and A3: abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
  shows  abs(f((p-1)·q)-(p-1)·f(q)) ≤ (abs(p-1)+ 1)·maxδ(f)
proof -
  from A2 have q∈ℤ  (p-1)·q ∈ ℤ
    using Int˙ZF˙2˙L1A int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L5 by auto
  with A1 have I: abs(δ(f,(p-1)·q,q)) ≤ maxδ(f) by (rule Int˙ZF˙2˙1˙L7)
  moreover note A3
  moreover from A1 A2 have
    abs(f((p-1)·q)-(p-1)·f(q)) ≤ abs(δ(f,(p-1)·q,q))+abs(f(p·q)-p·f(q))
    using AlmostHoms˙def Int˙ZF˙2˙L1A Int˙ZF˙2˙2˙L1 by simp
  ultimately have
    abs(f((p-1)·q)-(p-1)·f(q)) ≤ maxδ(f)+(abs(p)+1)·maxδ(f)
    by (rule Int˙ZF˙2˙L15)
  with I A2 show thesis using Int˙ZF˙2˙L1A Int˙ZF˙1˙2˙L5 by simp
qed
```

If f is a slope, then $|f(p \cdot q) - p \cdot f(q)| \leq (|p| + 1)\cdot\max\delta(f)$. Proof by cases on $0 \leq p$.

```
lemma (in int1) Int˙ZF˙2˙2˙L4:
  assumes A1: f∈𝒮 and A2: p∈ℤ q∈ℤ
  shows abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
proof -
  { assume 0≤p
    moreover from A1 A2 have abs(f(0·q)-0·f(q)) ≤ (abs(0)+1)·maxδ(f)
      using int˙zero˙one˙are˙int Int˙ZF˙2˙1˙L2B Int˙ZF˙1˙1˙L4
 Int˙ZF˙2˙1˙L8 Int˙ZF˙2˙L18 by simp
```

```
    moreover from A1 A2 have
      ∀p. 0≤p ∧ abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f) ⟶
      abs(f((p+1)·q)-(p+1)·f(q)) ≤ (abs(p+1)+ 1)·maxδ(f)
      using Int˙ZF˙2˙2˙L2 by simp
    ultimately have abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
      by (rule Induction˙on˙int) }
  moreover
  { assume ¬(0≤p)
    with A2 have p≤0 using Int˙ZF˙2˙L19A by simp
    moreover from A1 A2 have abs(f(0·q)-0·f(q)) ≤ (abs(0)+1)·maxδ(f)
      using int˙zero˙one˙are˙int Int˙ZF˙2˙1˙L2B Int˙ZF˙1˙1˙L4
 Int˙ZF˙2˙1˙L8 Int˙ZF˙2˙L18 by simp
    moreover from A1 A2 have
      ∀p. p≤0 ∧ abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f) ⟶
      abs(f((p-1)·q)-(p-1)·f(q)) ≤ (abs(p-1)+ 1)·maxδ(f)
      using Int˙ZF˙2˙2˙L3 by simp
    ultimately have abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
      by (rule Back˙induct˙on˙int) }
  ultimately show thesis by blast
qed
```

The next elegant result is Lemma 7 in the Arthan's paper [2].

```
lemma (in int1) Arthan˙Lem˙7:
 assumes A1: f∈𝒮 and A2: p∈ℤ  q∈ℤ
  shows abs(q·f(p)-p·f(q)) ≤ (abs(p)+abs(q)+2)·maxδ(f)
proof -
  from A1 A2 have T:
    q·f(p)-f(p·q) ∈ ℤ
    f(p·q)-p·f(q) ∈ ℤ
    f(q·p) ∈ ℤ  f(p·q) ∈ ℤ
    q·f(p) ∈ ℤ  p·f(q) ∈ ℤ
    maxδ(f) ∈ ℤ
    abs(q) ∈ ℤ  abs(p) ∈ ℤ
    using Int˙ZF˙1˙1˙L5 Int˙ZF˙2˙1˙L2B Int˙ZF˙2˙1˙L7 Int˙ZF˙2˙L14 by auto
  moreover have abs(q·f(p)-f(p·q)) ≤ (abs(q)+1)·maxδ(f)
  proof -
    from A1 A2 have abs(f(q·p)-q·f(p)) ≤ (abs(q)+1)·maxδ(f)
      using Int˙ZF˙2˙2˙L4 by simp
    with T A2 show thesis
      using Int˙ZF˙2˙L20 Int˙ZF˙1˙1˙L5 by simp
  qed
  moreover from A1 A2 have abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
    using Int˙ZF˙2˙2˙L4 by simp
  ultimately have
    abs(q·f(p)-f(p·q)+(f(p·q)-p·f(q))) ≤ (abs(q)+1)·maxδ(f)+(abs(p)+1)·maxδ(f)
    using Int˙ZF˙2˙L21 by simp
  with T show thesis using Int˙ZF˙1˙2˙L9 int˙zero˙one˙are˙int Int˙ZF˙1˙2˙L10
    by simp
qed
```

This is Lemma 8 in the Arthan's paper.

```
lemma (in int1) Arthan˙Lem˙8: assumes A1: f∈𝒮
  shows ∃A B. A∈ℤ ∧ B∈ℤ ∧ (∀p∈ℤ. abs(f(p)) ≤ A·abs(p)+B)
proof -
  let A = maxδ(f) + abs(f(1))
  let B = 3·maxδ(f)
  from A1 have A∈ℤ B∈ℤ
    using int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L5 Int˙ZF˙2˙1˙L2B
      Int˙ZF˙2˙1˙L7 Int˙ZF˙2˙L14 by auto
  moreover have ∀p∈ℤ. abs(f(p)) ≤ A·abs(p)+B
  proof
    fix p assume A2: p∈ℤ
    with A1 have T:
      f(p) ∈ ℤ  abs(p) ∈ ℤ  f(1) ∈ ℤ
      p·f(1) ∈ ℤ  3∈ℤ  maxδ(f) ∈ ℤ
      using Int˙ZF˙2˙1˙L2B Int˙ZF˙2˙L14 int˙zero˙one˙are˙int
 Int˙ZF˙1˙1˙L5 Int˙ZF˙2˙1˙L7 by auto
    from A1 A2 have
      abs(1·f(p)-p·f(1)) ≤ (abs(p)+abs(1)+2)·maxδ(f)
      using int˙zero˙one˙are˙int Arthan˙Lem˙7 by simp
    with T have abs(f(p)) ≤ abs(p·f(1))+(abs(p)+3)·maxδ(f)
      using Int˙ZF˙2˙L16A Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙2˙L11
 Int˙triangle˙ineq2 by simp
    with A2 T show abs(f(p)) ≤ A·abs(p)+B
      using Int˙ZF˙1˙3˙L14 by simp
  qed
  ultimately show thesis by auto
qed
```

If f and g are slopes, then $f \circ g$ is equivalent (almost equal) to $g \circ f$. This is Theorem 9 in Arthan's paper [2].

```
theorem (in int1) Arthan˙Th˙9: assumes A1: f∈𝒮  g∈𝒮
  shows f∘g ∼ g∘f
proof -
   from A1 have
      ∃A B. A∈ℤ ∧ B∈ℤ ∧ (∀p∈ℤ. abs(f(p)) ≤ A·abs(p)+B)
      ∃C D. C∈ℤ ∧ D∈ℤ ∧ (∀p∈ℤ. abs(g(p)) ≤ C·abs(p)+D)
      using Arthan˙Lem˙8 by auto
    then obtain A B C D where D1: A∈ℤ B∈ℤ C∈ℤ D∈ℤ and D2:
      ∀p∈ℤ. abs(f(p)) ≤ A·abs(p)+B
      ∀p∈ℤ. abs(g(p)) ≤ C·abs(p)+D
      by auto
    let E = maxδ(g)·(A+1) + maxδ(f)·(C+1)
    let F = (B·maxδ(g) + 2·maxδ(g)) + (D·maxδ(f) + 2·maxδ(f))
  { fix p assume A2: p∈ℤ
    with A1 have T1:
      g(p) ∈ ℤ  f(p) ∈ ℤ  abs(p) ∈ ℤ  2 ∈ ℤ
      f(g(p)) ∈ ℤ  g(f(p)) ∈ ℤ  f(g(p)) - g(f(p)) ∈ ℤ
      p·f(g(p)) ∈ ℤ  p·g(f(p)) ∈ ℤ
```

abs(f(g(p))-g(f(p))) ∈ ℤ
using Int˙ZF˙2˙1˙L2B Int˙ZF˙2˙1˙L10 Int˙ZF˙1˙1˙L5 Int˙ZF˙2˙L14 int˙two˙three˙are˙int
by auto
with A1 A2 have
abs((f(g(p))-g(f(p)))·p) ≤
(abs(p)+abs(f(p))+**2**)·maxδ(g) + (abs(p)+abs(g(p))+**2**)·maxδ(f)
using Arthan˙Lem˙7 Int˙ZF˙1˙2˙L10A Int˙ZF˙1˙2˙L12 by simp
moreover have
(abs(p)+abs(f(p))+**2**)·maxδ(g) + (abs(p)+abs(g(p))+**2**)·maxδ(f) ≤
((maxδ(g)·(A+**1**) + maxδ(f)·(C+**1**)))·abs(p) +
((B·maxδ(g) + **2**·maxδ(g)) + (D·maxδ(f) + **2**·maxδ(f)))
proof -
from D2 A2 T1 have
abs(p)+abs(f(p))+**2** ≤ abs(p)+(A·abs(p)+B)+**2**
abs(p)+abs(g(p))+**2** ≤ abs(p)+(C·abs(p)+D)+**2**
using Int˙ZF˙2˙L15C by auto
with A1 have
(abs(p)+abs(f(p))+**2**)·maxδ(g) ≤ (abs(p)+(A·abs(p)+B)+**2**)·maxδ(g)
(abs(p)+abs(g(p))+**2**)·maxδ(f) ≤ (abs(p)+(C·abs(p)+D)+**2**)·maxδ(f)
using Int˙ZF˙2˙1˙L8 Int˙ZF˙1˙3˙L13 by auto
moreover from A1 D1 T1 have
(abs(p)+(A·abs(p)+B)+**2**)·maxδ(g) =
maxδ(g)·(A+**1**)·abs(p) + (B·maxδ(g) + **2**·maxδ(g))
(abs(p)+(C·abs(p)+D)+**2**)·maxδ(f) =
maxδ(f)·(C+**1**)·abs(p) + (D·maxδ(f) + **2**·maxδ(f))
using Int˙ZF˙2˙1˙L8 Int˙ZF˙1˙2˙L13 by auto
ultimately have
(abs(p)+abs(f(p))+**2**)·maxδ(g) + (abs(p)+abs(g(p))+**2**)·maxδ(f) ≤
(maxδ(g)·(A+**1**)·abs(p) + (B·maxδ(g) + **2**·maxδ(g))) +
(maxδ(f)·(C+**1**)·abs(p) + (D·maxδ(f) + **2**·maxδ(f)))
using int˙ineq˙add˙sides by simp
moreover from A1 A2 D1 have abs(p) ∈ ℤ
maxδ(g)·(A+**1**) ∈ ℤ B·maxδ(g) + **2**·maxδ(g) ∈ ℤ
maxδ(f)·(C+**1**) ∈ ℤ D·maxδ(f) + **2**·maxδ(f) ∈ ℤ
using Int˙ZF˙2˙L14 Int˙ZF˙2˙1˙L8 int˙zero˙one˙are˙int
Int˙ZF˙1˙1˙L5 int˙two˙three˙are˙int by auto
ultimately show thesis using Int˙ZF˙1˙2˙L14 by simp
qed
ultimately have
abs((f(g(p))-g(f(p)))·p) ≤ E·abs(p) + F
by (rule Int˙order˙transitive)
with A2 T1 have
abs(f(g(p))-g(f(p)))·abs(p) ≤ E·abs(p) + F
abs(f(g(p))-g(f(p))) ∈ ℤ
using Int˙ZF˙1˙3˙L5 by auto
} then have
∀p∈ℤ. abs(f(g(p))-g(f(p))) ∈ ℤ
∀p∈ℤ. abs(f(g(p))-g(f(p)))·abs(p) ≤ E·abs(p) + F
by auto

```
  moreover from A1 D1 have E ∈ ℤ  F ∈ ℤ
    using int˙zero˙one˙are˙int int˙two˙three˙are˙int Int˙ZF˙2˙1˙L8 Int˙ZF˙1˙1˙L5
    by auto
  ultimately have
    ∃L. ∀p∈ℤ. abs(f(g(p))-g(f(p))) ≤ L
    by (rule Int˙ZF˙1˙7˙L1)
  with A1 obtain L where ∀p∈ℤ. abs((f∘g)(p)-(g∘f)(p)) ≤ L
    using Int˙ZF˙2˙1˙L10 by auto
  moreover from A1 have f∘g ∈ 𝒮  g∘f ∈ 𝒮
    using Int˙ZF˙2˙1˙L11 by auto
  ultimately show f∘g ∼ g∘f using Int˙ZF˙2˙1˙L9 by auto
qed

end
```

46 Integers 3

```
theory Int˙ZF˙3 imports Int˙ZF˙2

begin
```

This theory is a continuation of Int˙ZF˙2. We consider here the properties of slopes (almost homomorphisms on integers) that allow to define the order relation and multiplicative inverse on real numbers. We also prove theorems that allow to show completeness of the order relation of real numbers we define in Real˙ZF.

46.1 Positive slopes

This section provides background material for defining the order relation on real numbers.

Positive slopes are functions (of course.)

```
lemma (in int1) Int˙ZF˙2˙3˙L1: assumes A1: f∈𝒮₊ shows f:ℤ→ℤ
  using assms AlmostHoms˙def PositiveSet˙def by simp
```

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

```
lemma (in int1) Int˙ZF˙2˙3˙L1A:
  assumes A1: f∈𝒮₊ and A2: ∃n ∈ f(ℤ₊) ∩ ℤ₊. a≤n
  shows ∃M∈ℤ₊. a ≤ f(M)
proof -
 from A1 have f:ℤ→ℤ  ℤ₊ ⊆ ℤ
    using AlmostHoms˙def PositiveSet˙def by auto
 with A2 show thesis using func˙imagedef by auto
qed
```

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

```
lemma (in int1) Arthan˙Lem˙3:
  assumes A1: f∈𝒮₊ and A2: D ∈ ℤ₊
  shows ∃M∈ℤ₊. ∀m∈ℤ₊. (m+1)·D ≤ f(m·M)
proof -
  let E = maxδ(f) + D
  let A = f(ℤ₊) ∩ ℤ₊
  from A1 A2 have I: D≤E
    using Int˙ZF˙1˙5˙L3 Int˙ZF˙2˙1˙L8 Int˙ZF˙2˙L1A Int˙ZF˙2˙L15D
    by simp
  from A1 A2 have A ⊆ ℤ₊  A ∉ Fin(ℤ)  2·E ∈ ℤ
    using int˙two˙three˙are˙int Int˙ZF˙2˙1˙L8 PositiveSet˙def Int˙ZF˙1˙1˙L5
    by auto
  with A1 have ∃M∈ℤ₊.  2·E ≤ f(M)
    using Int˙ZF˙1˙5˙L2A Int˙ZF˙2˙3˙L1A by simp
  then obtain M where II: M∈ℤ₊  and III: 2·E ≤ f(M)
    by auto
  { fix m assume m∈ℤ₊ then have A4: 1≤m
      using Int˙ZF˙1˙5˙L3 by simp
    moreover from II III have (1+1) ·E ≤ f(1·M)
      using PositiveSet˙def Int˙ZF˙1˙1˙L4 by simp
    moreover have ∀k.
      1≤k ∧ (k+1)·E ≤ f(k·M) ⟶ (k+1+1)·E ≤ f((k+1)·M)
    proof -
      { fix k assume A5: 1≤k  and A6: (k+1)·E ≤ f(k·M)
 with A1 A2 II have T:
   k∈ℤ  M∈ℤ  k+1 ∈ ℤ  E∈ℤ  (k+1)·E ∈ ℤ  2·E ∈ ℤ
   using Int˙ZF˙2˙L1A PositiveSet˙def int˙zero˙one˙are˙int
     Int˙ZF˙1˙1˙L5 Int˙ZF˙2˙1˙L8 by auto
 from A1 A2 A5 II have
   δ(f,k·M,M) ∈ ℤ   abs(δ(f,k·M,M)) ≤ maxδ(f)   0≤D
   using Int˙ZF˙2˙L1A PositiveSet˙def Int˙ZF˙1˙1˙L5
     Int˙ZF˙2˙1˙L7 Int˙ZF˙2˙L16C by auto
 with III A6 have
   (k+1)·E + (2·E - E) ≤ f(k·M) + (f(M) + δ(f,k·M,M))
   using Int˙ZF˙1˙3˙L19A int˙ineq˙add˙sides by simp
 with A1 T have (k+1+1)·E ≤ f((k+1)·M)
   using Int˙ZF˙1˙1˙L1 int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L4
     Int˙ZF˙1˙2˙L11 Int˙ZF˙2˙1˙L13 by simp
      } then show thesis by simp
    qed
    ultimately have (m+1)·E ≤ f(m·M) by (rule Induction˙on˙int)
    with A4 I have (m+1)·D ≤ f(m·M) using Int˙ZF˙1˙3˙L13A
      by simp
  } then have ∀m∈ℤ₊.(m+1)·D ≤ f(m·M) by simp
  with II show thesis by auto
qed
```

A special case of Arthan˙Lem˙3 when $D = 1$.

```
corollary (in int1) Arthan˙L˙3˙spec: assumes A1: f ∈ 𝒮₊
```

```
  shows ∃M∈ℤ+.∀n∈ℤ+. n+1 ≤ f(n·M)
proof -
  have ∀n∈ℤ+. n+1 ∈ ℤ
    using PositiveSet˙def int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L5
    by simp
  then have ∀n∈ℤ+. (n+1)·1 = n+1
    using Int˙ZF˙1˙1˙L4 by simp
  moreover from A1 have ∃M∈ℤ+. ∀n∈ℤ+. (n+1)·1 ≤ f(n·M)
    using int˙one˙two˙are˙pos Arthan˙Lem˙3 by simp
  ultimately show thesis by simp
qed
```

We know from Group˙ZF˙3.thy that finite range functions are almost homomorphisms. Besides reminding that fact for slopes the next lemma shows that finite range functions do not belong to $\mathcal{S}_+$. This is important, because the projection of the set of finite range functions defines zero in the real number construction in Real˙ZF˙x.thy series, while the projection of $\mathcal{S}_+$ becomes the set of (strictly) positive reals. We don't want zero to be positive, do we? The next lemma is a part of Lemma 5 in the Arthan's paper [2].

```
lemma (in int1) Int˙ZF˙2˙3˙L1B:
  assumes A1: f ∈ FinRangeFunctions(ℤ,ℤ)
  shows f∈𝒮   f ∉ 𝒮+
proof -
  from A1 show f∈𝒮 using Int˙ZF˙2˙1˙L1 group1.Group˙ZF˙3˙3˙L1
    by auto
  have ℤ+ ⊆ ℤ using PositiveSet˙def by auto
  with A1 have f(ℤ+) ∈ Fin(ℤ)
    using Finite1˙L21 by simp
  then have f(ℤ+) ∩ ℤ+ ∈ Fin(ℤ)
    using Fin˙subset˙lemma by blast
  thus f ∉ 𝒮+ by auto
qed
```

We want to show that if f is a slope and neither f nor $-f$ are in $\mathcal{S}_+$, then f is bounded. The next lemma is the first step towards that goal and shows that if slope is not in $\mathcal{S}_+$ then $f(\mathbb{Z}_+)$ is bounded above.

```
lemma (in int1) Int˙ZF˙2˙3˙L2: assumes A1: f∈𝒮 and A2: f ∉ 𝒮+
  shows IsBoundedAbove(f(ℤ+), IntegerOrder)
proof -
  from A1 have f:ℤ→ℤ using AlmostHoms˙def by simp
  then have f(ℤ+) ⊆ ℤ using func1˙1˙L6 by simp
  moreover from A1 A2 have f(ℤ+) ∩ ℤ+ ∈ Fin(ℤ) by auto
  ultimately show thesis using Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙2˙L4
    by simp
qed
```

If f is a slope and $-f \notin \mathcal{S}_+$, then $f(\mathbb{Z}_+)$ is bounded below.

```
lemma (in int1) Int˙ZF˙2˙3˙L3: assumes A1: f∈𝒮 and A2: -f ∉ 𝒮+
```

```
  shows IsBoundedBelow(f(ℤ+), IntegerOrder)
proof -
  from A1 have T: f:ℤ→ℤ using AlmostHoms˙def by simp
  then have (-(f(ℤ+))) = (-f)(ℤ+)
    using Int˙ZF˙1˙T2 group0˙2˙T2 PositiveSet˙def func1˙1˙L15C
    by auto
  with A1 A2 T show IsBoundedBelow(f(ℤ+), IntegerOrder)
    using Int˙ZF˙2˙1˙L12 Int˙ZF˙2˙3˙L2 PositiveSet˙def func1˙1˙L6
      Int˙ZF˙2˙T1 group3.OrderedGroup˙ZF˙2˙L5 by simp
qed
```

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

```
lemma (in int1) Int˙ZF˙2˙3˙L4:
  assumes A1: f∈𝒮 and A2: m∈ℤ
  and A3: ∀n∈ℤ+. abs(f(n)) ≤ L
  shows abs(f(m)) ≤ 𝟐·maxδ(f) + L
proof -
  from A1 A3 have
    𝟎 ≤ abs(f(𝟏))  abs(f(𝟏)) ≤ L
    using int˙zero˙one˙are˙int Int˙ZF˙2˙1˙L2B int˙abs˙nonneg int˙one˙two˙are˙pos
    by auto
  then have II: 𝟎≤L by (rule Int˙order˙transitive)
  note A2
  moreover have abs(f(𝟎)) ≤ 𝟐·maxδ(f) + L
  proof -
    from A1 have
      abs(f(𝟎)) ≤ maxδ(f)  𝟎 ≤ maxδ(f)
      and T: maxδ(f) ∈ ℤ
      using Int˙ZF˙2˙1˙L8 by auto
    with II have abs(f(𝟎)) ≤ maxδ(f) + maxδ(f) + L
      using Int˙ZF˙2˙L15F by simp
    with T show thesis using Int˙ZF˙1˙1˙L4 by simp
  qed
  moreover from A1 A3 II have
    ∀n∈ℤ+. abs(f(n)) ≤ 𝟐·maxδ(f) + L
    using Int˙ZF˙2˙1˙L8 Int˙ZF˙1˙3˙L5A Int˙ZF˙2˙L15F
    by simp
  moreover have ∀n∈ℤ+. abs(f(-n)) ≤ 𝟐·maxδ(f) + L
  proof
    fix n assume n∈ℤ+
    with A1 A3 have
      𝟐·maxδ(f) ∈ ℤ
      abs(f(-n)) ≤ 𝟐·maxδ(f) + abs(f(n))
      abs(f(n)) ≤ L
      using int˙two˙three˙are˙int Int˙ZF˙2˙1˙L8 Int˙ZF˙1˙1˙L5
 PositiveSet˙def Int˙ZF˙2˙1˙L14 by auto
    then show abs(f(-n)) ≤ 𝟐·maxδ(f) + L
      using Int˙ZF˙2˙L15A by blast
  qed
```

```
  ultimately show thesis by (rule Int˙ZF˙2˙L19B)
qed
```

A slope whose image of the set of positive integers is bounded is a finite range function.

```
lemma (in int1) Int˙ZF˙2˙3˙L4A:
  assumes A1: f∈𝒮 and A2: IsBounded(f(ℤ₊), IntegerOrder)
  shows f ∈ FinRangeFunctions(ℤ,ℤ)
proof -
  have T1: ℤ₊ ⊆ ℤ using PositiveSet˙def by auto
  from A1 have T2: f:ℤ→ℤ using AlmostHoms˙def by simp
  from A2 obtain L where ∀a∈f(ℤ₊). abs(a) ≤ L
    using Int˙ZF˙1˙3˙L20A by auto
  with T2 T1 have ∀n∈ℤ₊. abs(f(n)) ≤ L
    by (rule func1˙1˙L15B)
  with A1 have ∀m∈ℤ. abs(f(m)) ≤ 2·maxδ(f) + L
    using Int˙ZF˙2˙3˙L4 by simp
  with T2 have f(ℤ) ∈ Fin(ℤ)
    by (rule Int˙ZF˙1˙3˙L20C)
  with T2 show f ∈ FinRangeFunctions(ℤ,ℤ)
    using FinRangeFunctions˙def by simp
qed
```

A slope whose image of the set of positive integers is bounded below is a finite range function or a positive slope.

```
lemma (in int1) Int˙ZF˙2˙3˙L4B:
  assumes f∈𝒮 and IsBoundedBelow(f(ℤ₊), IntegerOrder)
  shows f ∈ FinRangeFunctions(ℤ,ℤ) ∨ f∈𝒮₊
  using assms Int˙ZF˙2˙3˙L2 IsBounded˙def Int˙ZF˙2˙3˙L4A
  by auto
```

If one slope is not greater then another on positive integers, then they are almost equal or the difference is a positive slope.

```
lemma (in int1) Int˙ZF˙2˙3˙L4C: assumes A1: f∈𝒮  g∈𝒮 and
  A2: ∀n∈ℤ₊. f(n) ≤ g(n)
  shows f∼g ∨ g + (-f) ∈ 𝒮₊
proof -
  let h = g + (-f)
  from A1 have (-f) ∈ 𝒮 using Int˙ZF˙2˙1˙L12
    by simp
  with A1 have I: h ∈ 𝒮 using Int˙ZF˙2˙1˙L12C
    by simp
  moreover have IsBoundedBelow(h(ℤ₊), IntegerOrder)
  proof -
    from I have
      h:ℤ→ℤ and ℤ₊⊆ℤ using AlmostHoms˙def PositiveSet˙def
      by auto
    moreover from A1 A2 have ∀n∈ℤ₊. ⟨0, h(n)⟩ ∈ IntegerOrder
```

```
      using Int˙ZF˙2˙1˙L2B PositiveSet˙def Int˙ZF˙1˙3˙L10A
 Int˙ZF˙2˙1˙L12 Int˙ZF˙2˙1˙L12B Int˙ZF˙2˙1˙L12A
      by simp
    ultimately show IsBoundedBelow(h(ℤ+), IntegerOrder)
      by (rule func˙ZF˙8˙L1)
  qed
  ultimately have h ∈ FinRangeFunctions(ℤ,ℤ) ∨ h∈𝒮+
    using Int˙ZF˙2˙3˙L4B by simp
  with A1 show f∼g ∨ g + (-f) ∈ 𝒮+
    using Int˙ZF˙2˙1˙L9C by auto
qed
```

Positive slopes are arbitrarily large for large enough arguments.

```
lemma (in int1) Int˙ZF˙2˙3˙L5:
  assumes A1: f∈𝒮+ and A2: K∈ℤ
  shows ∃N∈ℤ+. ∀m. N≤m ⟶ K ≤ f(m)
proof -
  from A1 obtain M where I: M∈ℤ+ and II: ∀n∈ℤ+. n+𝟏 ≤ f(n·M)
    using Arthan˙L˙3˙spec by auto
  let j = GreaterOf(IntegerOrder,M,K - (minf(f,𝟎..(M-𝟏)) - maxδ(f)) - 𝟏)
  from A1 I have T1:
    minf(f,𝟎..(M-𝟏)) - maxδ(f) ∈ ℤ  M∈ℤ
    using Int˙ZF˙2˙1˙L15 Int˙ZF˙2˙1˙L8 Int˙ZF˙1˙1˙L5 PositiveSet˙def
    by auto
  with A2 I have T2:
    K - (minf(f,𝟎..(M-𝟏)) - maxδ(f)) ∈ ℤ
    K - (minf(f,𝟎..(M-𝟏)) - maxδ(f)) - 𝟏 ∈ ℤ
    using Int˙ZF˙1˙1˙L5 int˙zero˙one˙are˙int by auto
  with T1 have III: M ≤ j  and
    K - (minf(f,𝟎..(M-𝟏)) - maxδ(f)) - 𝟏 ≤ j
    using Int˙ZF˙1˙3˙L18 by auto
  with A2 T1 T2 have
    IV: K ≤ j+𝟏 + (minf(f,𝟎..(M-𝟏)) - maxδ(f))
    using int˙zero˙one˙are˙int Int˙ZF˙2˙L9C by simp
  let N = GreaterOf(IntegerOrder,𝟏,j·M)
  from T1 III have T3: j ∈ ℤ  j·M ∈ ℤ
    using Int˙ZF˙2˙L1A Int˙ZF˙1˙1˙L5 by auto
  then have V: N ∈ ℤ+ and VI: j·M ≤ N
    using int˙zero˙one˙are˙int Int˙ZF˙1˙5˙L3 Int˙ZF˙1˙3˙L18
    by auto
  { fix m
    let n = m zdiv M
    let k = m zmod M
    assume N≤m
    with VI have j·M ≤ m by (rule Int˙order˙transitive)
    with I III have
      VII: m = n·M+k
      j ≤ n  and
      VIII: n ∈ ℤ+  k ∈ 𝟎..(M-𝟏)
```

```
    using IntDiv˙ZF˙1˙L5 by auto
  with II have
    j + 1 ≤ n + 1  n+1 ≤ f(n·M)
    using int˙zero˙one˙are˙int int˙ord˙transl˙inv by auto
  then have j + 1 ≤  f(n·M)
    by (rule Int˙order˙transitive)
  with T1 have
    j+1 + (minf(f,0..(M-1)) - maxδ(f)) ≤
    f(n·M) + (minf(f,0..(M-1)) - maxδ(f))
    using int˙ord˙transl˙inv by simp
  with IV have K ≤ f(n·M) + (minf(f,0..(M-1)) - maxδ(f))
    by (rule Int˙order˙transitive)
  moreover from A1 I VIII have
    f(n·M) + (minf(f,0..(M-1))- maxδ(f)) ≤ f(n·M+k)
    using PositiveSet˙def Int˙ZF˙2˙1˙L16 by simp
  ultimately have K ≤ f(n·M+k)
    by (rule Int˙order˙transitive)
  with VII have K ≤ f(m) by simp
  } then have  ∀m. N≤m ⟶ K ≤ f(m)
    by simp
  with V show thesis by auto
qed
```

Positive slopes are arbitrarily small for small enough arguments. Kind of dual to Int˙ZF˙2˙3˙L5.

```
lemma (in int1) Int˙ZF˙2˙3˙L5A: assumes A1: f∈𝒮₊ and A2: K∈ℤ
  shows ∃N∈ℤ₊. ∀m. N≤m ⟶ f(-m) ≤ K
proof -
  from A1 have T1: abs(f(0)) + maxδ(f) ∈ ℤ
    using Int˙ZF˙2˙1˙L8 by auto
  with A2 have abs(f(0)) + maxδ(f) - K ∈ ℤ
    using Int˙ZF˙1˙1˙L5 by simp
  with A1 have
    ∃N∈ℤ₊. ∀m. N≤m ⟶ abs(f(0)) + maxδ(f) - K ≤ f(m)
    using Int˙ZF˙2˙3˙L5 by simp
  then obtain N where I: N∈ℤ₊ and II:
    ∀m. N≤m ⟶  abs(f(0)) + maxδ(f) - K ≤ f(m)
    by auto
  { fix m assume A3: N≤m
    with A1 have
      f(-m) ≤ abs(f(0)) + maxδ(f) - f(m)
      using Int˙ZF˙2˙L1A Int˙ZF˙2˙1˙L14 by simp
    moreover
    from II T1 A3 have abs(f(0)) + maxδ(f) - f(m) ≤
      (abs(f(0)) + maxδ(f)) -(abs(f(0)) + maxδ(f) - K)
      using Int˙ZF˙2˙L10 int˙ord˙transl˙inv by simp
    with A2 T1 have abs(f(0)) + maxδ(f) - f(m) ≤ K
      using Int˙ZF˙1˙2˙L3 by simp
    ultimately have f(-m) ≤ K
```

```
    by (rule Int˙order˙transitive)
  } then have ∀m. N≤m  ⟶ f(-m) ≤ K
    by simp
  with I show thesis by auto
qed
```

A special case of Int˙ZF˙2˙3˙L5 where $K = 1$.

```
corollary (in int1) Int˙ZF˙2˙3˙L6: assumes f∈𝒮₊
  shows ∃N∈ℤ₊. ∀m. N≤m ⟶ f(m) ∈ ℤ₊
  using assms int˙zero˙one˙are˙int Int˙ZF˙2˙3˙L5 Int˙ZF˙1˙5˙L3
  by simp
```

A special case of Int˙ZF˙2˙3˙L5 where $m = N$.

```
corollary (in int1) Int˙ZF˙2˙3˙L6A: assumes f∈𝒮₊ and K∈ℤ
   shows ∃N∈ℤ₊. K ≤ f(N)
proof -
  from assms have ∃N∈ℤ₊. ∀m. N≤m ⟶ K ≤ f(m)
    using Int˙ZF˙2˙3˙L5 by simp
  then obtain N where I: N ∈ ℤ₊  and II: ∀m. N≤m ⟶ K ≤ f(m)
    by auto
  then show thesis using PositiveSet˙def int˙ord˙is˙refl refl˙def
    by auto
qed
```

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

```
lemma (in int1) Int˙ZF˙2˙3˙L7: assumes A1: f∈𝒮
  and A2: ∀K∈ℤ. ∃n∈ℤ₊. K ≤ f(n)
  shows f ∈ 𝒮₊
proof -
  { fix K assume K∈ℤ
    with A2 obtain n where n∈ℤ₊  K ≤ f(n)
      by auto
    moreover from A1 have ℤ₊ ⊆ ℤ  f:ℤ→ℤ
      using PositiveSet˙def AlmostHoms˙def by auto
    ultimately have ∃m ∈ f(ℤ₊). K ≤ m
      using func1˙1˙L15D by auto
  } then have ∀K∈ℤ. ∃m ∈ f(ℤ₊). K ≤ m by simp
  with A1 show f ∈ 𝒮₊ using Int˙ZF˙4˙L9 Int˙ZF˙2˙3˙L2
    by auto
qed
```

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

```
theorem (in int1) Int˙ZF˙2˙3˙L8:
  assumes A1: f∈𝒮 and A2: f ∉ FinRangeFunctions(ℤ,ℤ)
  shows (f ∈ 𝒮₊) Xor ((-f) ∈ 𝒮₊)
proof -
  have T1: ℤ₊ ⊆ ℤ using PositiveSet˙def by auto
  from A1 have T2: f:ℤ→ℤ  using AlmostHoms˙def by simp
```

```
  then have I: f(ℤ+) ⊆ ℤ using func1˙1˙L6 by auto
  from A1 A2 have f ∈ 𝒮+ ∨ (-f) ∈ 𝒮+
    using Int˙ZF˙2˙3˙L2 Int˙ZF˙2˙3˙L3 IsBounded˙def Int˙ZF˙2˙3˙L4A
    by blast
  moreover have ¬(f ∈ 𝒮+ ∧ (-f) ∈ 𝒮+)
  proof -
    { assume A3: f ∈ 𝒮+  and A4: (-f) ∈ 𝒮+
      from A3 obtain N1 where
 I: N1∈ℤ+ and II: ∀m. N1≤m ⟶ f(m) ∈ ℤ+
 using Int˙ZF˙2˙3˙L6 by auto
      from A4 obtain N2 where
 III: N2∈ℤ+ and IV: ∀m. N2≤m ⟶ (-f)(m) ∈ ℤ+
 using Int˙ZF˙2˙3˙L6 by auto
      let N = GreaterOf(IntegerOrder,N1,N2)
      from I III have N1 ≤ N  N2 ≤ N
 using PositiveSet˙def Int˙ZF˙1˙3˙L18 by auto
      with A1 II IV have
 f(N) ∈ ℤ+  (-f)(N) ∈ ℤ+  (-f)(N) = -(f(N))
 using Int˙ZF˙2˙L1A PositiveSet˙def Int˙ZF˙2˙1˙L12A
 by auto
      then have False using Int˙ZF˙1˙5˙L8 by simp
    } thus thesis by auto
  qed
  ultimately show (f ∈ 𝒮+) Xor ((-f) ∈ 𝒮+)
    using Xor˙def by simp
qed
```

The sum of positive slopes is a positive slope.

```
theorem (in int1) sum˙of˙pos˙sls˙is˙pos˙sl:
  assumes A1: f ∈ 𝒮+  g ∈ 𝒮+
  shows f+g ∈ 𝒮+
proof -
  { fix K assume K∈ℤ
    with A1 have ∃N∈ℤ+. ∀m. N≤m ⟶ K ≤ f(m)
      using Int˙ZF˙2˙3˙L5 by simp
    then obtain N where I: N∈ℤ+ and II: ∀m. N≤m ⟶ K ≤ f(m)
      by auto
    from A1 have ∃M∈ℤ+. ∀m. M≤m ⟶ 𝟬 ≤ g(m)
      using int˙zero˙one˙are˙int Int˙ZF˙2˙3˙L5 by simp
    then obtain M where III: M∈ℤ+ and IV: ∀m. M≤m ⟶ 𝟬 ≤ g(m)
      by auto
    let L = GreaterOf(IntegerOrder,N,M)
    from I III have V: L ∈ ℤ+  ℤ+ ⊆ ℤ
      using GreaterOf˙def PositiveSet˙def by auto
    moreover from A1 V have (f+g)(L) = f(L) + g(L)
      using Int˙ZF˙2˙1˙L12B by auto
    moreover from I II III IV have K ≤ f(L) + g(L)
      using PositiveSet˙def Int˙ZF˙1˙3˙L18 Int˙ZF˙2˙L15F
      by simp
```

```
    ultimately have L ∈ ℤ+  K ≤ (f+g)(L)
      by auto
    then have ∃n ∈ℤ+. K ≤ (f+g)(n)
      by auto
  } with A1 show f+g ∈ 𝒮+
    using Int˙ZF˙2˙1˙L12C Int˙ZF˙2˙3˙L7 by simp
qed
```

The composition of positive slopes is a positive slope.

```
theorem (in int1) comp˙of˙pos˙sls˙is˙pos˙sl:
  assumes A1: f ∈ 𝒮+  g ∈ 𝒮+
  shows f∘g ∈ 𝒮+
proof -
  { fix K assume K∈ℤ
    with A1 have ∃N∈ℤ+. ∀m. N≤m ⟶ K ≤ f(m)
      using Int˙ZF˙2˙3˙L5 by simp
    then obtain N where N∈ℤ+ and I: ∀m. N≤m ⟶ K ≤ f(m)
      by auto
    with A1 have ∃M∈ℤ+. N ≤ g(M)
      using PositiveSet˙def Int˙ZF˙2˙3˙L6A by simp
    then obtain M where M∈ℤ+  N ≤ g(M)
      by auto
    with A1 I have ∃M∈ℤ+. K ≤ (f∘g)(M)
      using PositiveSet˙def Int˙ZF˙2˙1˙L10
      by auto
  } with A1 show f∘g ∈ 𝒮+
    using Int˙ZF˙2˙1˙L11 Int˙ZF˙2˙3˙L7
    by simp
qed
```

A slope equivalent to a positive one is positive.

```
lemma (in int1) Int˙ZF˙2˙3˙L9:
  assumes A1: f ∈ 𝒮+ and A2: ⟨f,g⟩ ∈ AlEqRel shows g ∈ 𝒮+
proof -
  from A2 have T: g∈𝒮 and ∃L∈ℤ. ∀m∈ℤ. abs(f(m)-g(m)) ≤ L
    using Int˙ZF˙2˙1˙L9A by auto
   then obtain L where
     I: L∈ℤ  and II: ∀m∈ℤ. abs(f(m)-g(m)) ≤ L
     by auto
  { fix K assume A3: K∈ℤ
    with I have K+L ∈ ℤ
      using Int˙ZF˙1˙1˙L5 by simp
    with A1 obtain M where III: M∈ℤ+  and IV: K+L ≤ f(M)
      using Int˙ZF˙2˙3˙L6A by auto
    with A1 A3 I have  K ≤ f(M)-L
      using PositiveSet˙def Int˙ZF˙2˙1˙L2B Int˙ZF˙2˙L9B
      by simp
    moreover from A1 T II III have
      f(M)-L ≤ g(M)
```

```
    using PositiveSet˙def Int˙ZF˙2˙1˙L2B Int˙triangle˙ineq2
    by simp
  ultimately have K ≤  g(M)
    by (rule Int˙order˙transitive)
  with III have ∃n∈ℤ₊. K ≤ g(n)
    by auto
 } with T show g ∈ 𝒮₊
  using Int˙ZF˙2˙3˙L7 by simp
qed
```

The set of positive slopes is saturated with respect to the relation of equivalence of slopes.

```
lemma (in int1) pos˙slopes˙saturated: shows IsSaturated(AlEqRel,𝒮₊)
proof -
 have
  equiv(𝒮,AlEqRel)
  AlEqRel ⊆ 𝒮 × 𝒮
  using Int˙ZF˙2˙1˙L9B by auto
 moreover have 𝒮₊ ⊆ 𝒮 by auto
 moreover have ∀f∈𝒮₊. ∀g∈𝒮. ⟨f,g⟩ ∈ AlEqRel ⟶ g ∈ 𝒮₊
  using Int˙ZF˙2˙3˙L9 by blast
 ultimately show IsSaturated(AlEqRel,𝒮₊)
  by (rule EquivClass˙3˙L3)
qed
```

A technical lemma involving a projection of the set of positive slopes and a logical epression with exclusive or.

```
lemma (in int1) Int˙ZF˙2˙3˙L10:
 assumes A1: f∈𝒮  g∈𝒮
 and A2: R = −AlEqRel−s″. s∈𝒮₊″
 and A3: (f∈𝒮₊) Xor (g∈𝒮₊)
 shows (AlEqRel−f″ ∈ R) Xor (AlEqRel−g″ ∈ R)
proof -
 from A1 A2 A3 have
  equiv(𝒮,AlEqRel)
  IsSaturated(AlEqRel,𝒮₊)
  𝒮₊ ⊆ 𝒮
  f∈𝒮  g∈𝒮
  R = −AlEqRel−s″. s∈𝒮₊″
  (f∈𝒮₊) Xor (g∈𝒮₊)
  using pos˙slopes˙saturated Int˙ZF˙2˙1˙L9B by auto
 then show thesis by (rule EquivClass˙3˙L7)
qed
```

Identity function is a positive slope.

```
lemma (in int1) Int˙ZF˙2˙3˙L11: shows id(ℤ) ∈ 𝒮₊
proof -
 let f = id(ℤ)
```

```
  { fix K assume K∈ℤ
    then obtain n where T: n∈ℤ+ and K≤n
      using Int˙ZF˙1˙5˙L9 by auto
    moreover from T have f(n) = n
      using PositiveSet˙def by simp
    ultimately have  n∈ℤ+ and K≤f(n)
      by auto
    then have ∃n∈ℤ+. K≤f(n) by auto
  } then show f ∈ 𝒮+
    using Int˙ZF˙2˙1˙L17 Int˙ZF˙2˙3˙L7 by simp
qed
```

The identity function is not almost equal to any bounded function.

```
lemma (in int1) Int˙ZF˙2˙3˙L12: assumes A1: f ∈ FinRangeFunctions(ℤ,ℤ)
  shows ¬(id(ℤ) ∼ f)
proof -
  { from A1 have id(ℤ) ∈ 𝒮+
      using Int˙ZF˙2˙3˙L11 by simp
    moreover assume ⟨id(ℤ),f⟩ ∈ AlEqRel
    ultimately have  f ∈ 𝒮+
      by (rule Int˙ZF˙2˙3˙L9)
    with A1 have False using Int˙ZF˙2˙3˙L1B
      by simp
  } then show ¬(id(ℤ) ∼ f) by auto
qed
```

46.2 Inverting slopes

Not every slope is a 1:1 function. However, we can still invert slopes in the sense that if f is a slope, then we can find a slope g such that $f \circ g$ is almost equal to the identity function. The goal of this this section is to establish this fact for positive slopes.

If f is a positive slope, then for every positive integer p the set $\{n \in Z_+ : p \leq f(n)\}$ is a nonempty subset of positive integers. Recall that $f^{-1}(p)$ is the notation for the smallest element of this set.

```
lemma (in int1) Int˙ZF˙2˙4˙L1:
  assumes A1: f ∈ 𝒮+ and A2: p∈ℤ+ and A3: A = –n∈ℤ+. p ≤ f(n)"
  shows
  A ⊆ ℤ+
  A ≠ 0
  f⁻¹(p) ∈ A
  ∀m∈A. f⁻¹(p) ≤ m
proof -
  from A3 show I: A ⊆ ℤ+ by auto
  from A1 A2 have ∃n∈ℤ+. p ≤ f(n)
    using PositiveSet˙def Int˙ZF˙2˙3˙L6A by simp
  with A3 show II: A ≠ 0 by auto
```

```
  from A3 I II show
    f⁻¹(p) ∈ A
    ∀m∈A. f⁻¹(p) ≤ m
    using Int˙ZF˙1˙5˙L1C by auto
qed
```

If f is a positive slope and p is a positive integer p, then $f^{-1}(p)$ (defined as the minimum of the set $\{n \in Z_+ : p \leq f(n)\}$) is a (well defined) positive integer.

```
lemma (in int1) Int˙ZF˙2˙4˙L2:
  assumes f ∈ 𝒮₊ and p∈ℤ₊
  shows
  f⁻¹(p) ∈ ℤ₊
  p ≤ f(f⁻¹(p))
  using assms Int˙ZF˙2˙4˙L1 by auto
```

If f is a positive slope and p is a positive integer such that $n \leq f(p)$, then $f^{-1}(n) \leq p$.

```
lemma (in int1) Int˙ZF˙2˙4˙L3:
  assumes f ∈ 𝒮₊ and  m∈ℤ₊  p∈ℤ₊ and m ≤ f(p)
  shows f⁻¹(m) ≤ p
  using assms Int˙ZF˙2˙4˙L1 by simp
```

An upper bound $f(f^{-1}(m) - 1)$ for positive slopes.

```
lemma (in int1) Int˙ZF˙2˙4˙L4:
  assumes A1: f ∈ 𝒮₊ and A2: m∈ℤ₊ and A3: f⁻¹(m)-𝟏 ∈ ℤ₊
  shows f(f⁻¹(m)-𝟏) ≤ m   f(f⁻¹(m)-𝟏) ≠ m
proof -
  from A1 A2 have T: f⁻¹(m) ∈ ℤ using Int˙ZF˙2˙4˙L2 PositiveSet˙def
    by simp
  from A1 A3 have f:ℤ→ℤ  and f⁻¹(m)-𝟏 ∈ ℤ
    using Int˙ZF˙2˙3˙L1 PositiveSet˙def by auto
  with A1 A2 have T1: f(f⁻¹(m)-𝟏) ∈ ℤ  m∈ℤ
    using apply˙funtype PositiveSet˙def by auto
  { assume m ≤ f(f⁻¹(m)-𝟏)
    with A1 A2 A3 have f⁻¹(m) ≤ f⁻¹(m)-𝟏
      by (rule Int˙ZF˙2˙4˙L3)
    with T have False using Int˙ZF˙1˙2˙L3AA
      by simp
  } then have I: ¬(m ≤ f(f⁻¹(m)-𝟏)) by auto
  with T1 show f(f⁻¹(m)-𝟏) ≤ m
    by (rule Int˙ZF˙2˙L19)
  from T1 I show f(f⁻¹(m)-𝟏) ≠ m
    by (rule Int˙ZF˙2˙L19)
qed
```

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

```
lemma (in int1) Int˙ZF˙2˙4˙L5:
```

```
  assumes A1: f ∈ 𝒮₊ and A2: m∈ℤ₊ and A3: m≤n
  shows f⁻¹(m) ≤ f⁻¹(n)
proof -
  from A2 A3 have T: n ∈ ℤ₊ using Int˙ZF˙1˙5˙L7 by blast
  with A1 have n ≤ f(f⁻¹(n)) using Int˙ZF˙2˙4˙L2
    by simp
  with A3 have m ≤ f(f⁻¹(n)) by (rule Int˙order˙transitive)
  with A1 A2 T show f⁻¹(m) ≤ f⁻¹(n)
    using Int˙ZF˙2˙4˙L2 Int˙ZF˙2˙4˙L3 by simp
qed
```

If $f^{-1}(m)$ is positive and n is a positive integer, then, then $f^{-1}(m+n)-1$ is positive.

```
lemma (in int1) Int˙ZF˙2˙4˙L6:
  assumes A1: f ∈ 𝒮₊ and A2: m∈ℤ₊  n∈ℤ₊ and
  A3: f⁻¹(m)-𝟏 ∈ ℤ₊
  shows f⁻¹(m+n)-𝟏 ∈ ℤ₊
proof -
  from A1 A2 have f⁻¹(m)-𝟏 ≤  f⁻¹(m+n) - 𝟏
     using PositiveSet˙def Int˙ZF˙1˙5˙L7A Int˙ZF˙2˙4˙L2
       Int˙ZF˙2˙4˙L5 int˙zero˙one˙are˙int Int˙ZF˙1˙1˙L4
       int˙ord˙transl˙inv by simp
  with A3 show f⁻¹(m+n)-𝟏 ∈ ℤ₊ using Int˙ZF˙1˙5˙L7
    by blast
qed
```

If f is a slope, then $f(f^{-1}(m+n)-f^{-1}(m)-f^{-1}(n))$ is uniformly bounded above and below. Will it be the messiest IsarMathLib proof ever? Only time will tell.

```
lemma (in int1) Int˙ZF˙2˙4˙L7:  assumes A1: f ∈ 𝒮₊ and
  A2: ∀m∈ℤ₊. f⁻¹(m)-𝟏 ∈ ℤ₊
  shows
  ∃U∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤ U
  ∃N∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. N ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
proof -
  from A1 have ∃L∈ℤ. ∀r∈ℤ. f(r) ≤ f(r-𝟏) + L
    using Int˙ZF˙2˙1˙L28 by simp
  then obtain L where
    I: L∈ℤ and II: ∀r∈ℤ. f(r) ≤ f(r-𝟏) + L
    by auto
  from A1 have
    ∃M∈ℤ. ∀r∈ℤ.∀p∈ℤ.∀q∈ℤ. f(r-p-q) ≤ f(r)-f(p)-f(q)+M
    ∃K∈ℤ. ∀r∈ℤ.∀p∈ℤ.∀q∈ℤ. f(r)-f(p)-f(q)+K ≤ f(r-p-q)
    using Int˙ZF˙2˙1˙L30 by auto
  then obtain M K where III: M∈ℤ and
    IV: ∀r∈ℤ.∀p∈ℤ.∀q∈ℤ. f(r-p-q) ≤ f(r)-f(p)-f(q)+M
    and
    V: K∈ℤ and VI: ∀r∈ℤ.∀p∈ℤ.∀q∈ℤ. f(r)-f(p)-f(q)+K ≤ f(r-p-q)
    by auto
```

```
 from I III V have
   L+M ∈ ℤ  (-L) - L + K ∈ ℤ
   using Int˙ZF˙1˙1˙L4 Int˙ZF˙1˙1˙L5 by auto
 moreover
   { fix m n
     assume A3: m∈ℤ₊ n∈ℤ₊
     have  f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤  L+M ∧
(-L)-L+K ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
     proof -
let r = f⁻¹(m+n)
let p = f⁻¹(m)
let q = f⁻¹(n)
from A1 A3 have T1:
  p ∈ ℤ₊  q ∈ ℤ₊  r ∈ ℤ₊
  using Int˙ZF˙2˙4˙L2 pos˙int˙closed˙add˙unfolded by auto
with A3 have T2:
  m ∈ ℤ  n ∈ ℤ  p ∈ ℤ  q ∈ ℤ  r ∈ ℤ
  using PositiveSet˙def by auto
from A2 A3 have T3:
  r-1 ∈ ℤ₊ p-1 ∈ ℤ₊  q-1 ∈ ℤ₊
  using pos˙int˙closed˙add˙unfolded by auto
from A1 A3 have VII:
  m+n ≤ f(r)
  m ≤ f(p)
  n ≤ f(q)
  using Int˙ZF˙2˙4˙L2 pos˙int˙closed˙add˙unfolded by auto
from A1 A3 T3 have VIII:
  f(r-1) ≤ m+n
  f(p-1) ≤ m
  f(q-1) ≤ n
  using pos˙int˙closed˙add˙unfolded Int˙ZF˙2˙4˙L4 by auto
have f(r-p-q) ≤ L+M
proof -
  from IV T2 have f(r-p-q) ≤ f(r)-f(p)-f(q)+M
    by simp
  moreover
  from I II T2 VIII have
    f(r) ≤ f(r-1) + L
    f(r-1) + L ≤ m+n+L
    using int˙ord˙transl˙inv by auto
  then have f(r) ≤  m+n+L
    by (rule Int˙order˙transitive)
  with VII have f(r) - f(p) ≤ m+n+L-m
    using int˙ineq˙add˙sides by simp
  with I T2 VII have f(r) - f(p) - f(q) ≤ n+L-n
    using Int˙ZF˙1˙2˙L9 int˙ineq˙add˙sides by simp
  with I III T2 have f(r) - f(p) - f(q) + M ≤ L+M
    using Int˙ZF˙1˙2˙L3 int˙ord˙transl˙inv by simp
  ultimately show f(r-p-q) ≤ L+M
```

```
    by (rule Int˙order˙transitive)
  qed
  moreover have (-L)-L +K ≤ f(r-p-q)
  proof -
    from I II T2 VIII have
      f(p) ≤ f(p-1) + L
      f(p-1) + L ≤ m +L
      using int˙ord˙transl˙inv by auto
    then have f(p) ≤  m +L
      by (rule Int˙order˙transitive)
    with VII have m+n -(m+L) ≤ f(r) - f(p)
      using int˙ineq˙add˙sides by simp
    with I T2 have n - L ≤  f(r) - f(p)
      using Int˙ZF˙1˙2˙L9 by simp
    moreover
    from I II T2 VIII have
      f(q) ≤ f(q-1) + L
      f(q-1) + L ≤ n +L
      using int˙ord˙transl˙inv by auto
    then have f(q) ≤  n +L
      by (rule Int˙order˙transitive)
    ultimately have
      n - L - (n+L) ≤  f(r) - f(p) - f(q)
      using int˙ineq˙add˙sides by simp
    with I V T2 have
      (-L)-L +K ≤  f(r) - f(p) - f(q) + K
      using Int˙ZF˙1˙2˙L3 int˙ord˙transl˙inv by simp
    moreover from VI T2 have
      f(r) - f(p) - f(q) + K ≤ f(r-p-q)
      by simp
    ultimately show (-L)-L +K ≤ f(r-p-q)
      by (rule Int˙order˙transitive)
  qed
  ultimately show
    f(r-p-q) ≤  L+M ∧
    (-L)-L+K ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
    by simp
      qed
    }
  ultimately show
    ∃U∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤ U
    ∃N∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. N ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
    by auto
qed
```

The expression $f^{-1}(m+n) - f^{-1}(m) - f^{-1}(n)$ is uniformly bounded for all pairs $\langle m, n\rangle \in \mathbb{Z}_+ \times \mathbb{Z}_+$. Recall that in the int1 context ε(f,x) is defined so that $\varepsilon(f, \langle m, n\rangle) = f^{-1}(m+n) - f^{-1}(m) - f^{-1}(n)$.

lemma (in int1) Int˙ZF˙2˙4˙L8: assumes A1: f $\in \mathcal{S}_+$ and

A2: $\forall$ m$\in\mathbb{Z}_+$. f^{-1}(m)-**1** $\in \mathbb{Z}_+$
shows $\exists$ M. $\forall$ x$\in\mathbb{Z}_+\times\mathbb{Z}_+$. abs($\varepsilon$(f,x)) $\leq$ M
proof -
from A1 A2 have
$\exists$ U$\in\mathbb{Z}$. $\forall$ m$\in\mathbb{Z}_+$. $\forall$ n$\in\mathbb{Z}_+$. f(f^{-1}(m+n)-f^{-1}(m)-f^{-1}(n)) $\leq$ U
$\exists$ N$\in\mathbb{Z}$. $\forall$ m$\in\mathbb{Z}_+$. $\forall$ n$\in\mathbb{Z}_+$. N $\leq$ 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:
$\forall$ m$\in\mathbb{Z}_+$. $\forall$ n$\in\mathbb{Z}_+$. f(f^{-1}(m+n)-f^{-1}(m)-f^{-1}(n)) $\leq$ U
$\forall$ m$\in\mathbb{Z}_+$. $\forall$ n$\in\mathbb{Z}_+$. N $\leq$ f(f^{-1}(m+n)-f^{-1}(m)-f^{-1}(n))
by auto
have $\mathbb{Z}_+\times\mathbb{Z}_+ \neq 0$ using int˙one˙two˙are˙pos by auto
moreover from A1 have f: $\mathbb{Z}\rightarrow\mathbb{Z}$
using AlmostHoms˙def by simp
moreover from A1 have
$\forall$ a$\in\mathbb{Z}$.$\exists$ b$\in\mathbb{Z}_+$.$\forall$ x. b$\leq$x $\longrightarrow$ a $\leq$ f(x)
using Int˙ZF˙2˙3˙L5 by simp
moreover from A1 have
$\forall$ a$\in\mathbb{Z}$.$\exists$ b$\in\mathbb{Z}_+$.$\forall$ y. b$\leq$y $\longrightarrow$ f(-y) $\leq$ a
using Int˙ZF˙2˙3˙L5A by simp
moreover have
$\forall$ x$\in\mathbb{Z}_+\times\mathbb{Z}_+$. ε(f,x) $\in \mathbb{Z} \wedge$ f(ε(f,x)) $\leq$ U $\wedge$ N $\leq$ f(ε(f,x))
proof -
{ fix x assume A3: x $\in \mathbb{Z}_+\times\mathbb{Z}_+$
let m = fst(x)
let n = snd(x)
from A3 have T: m $\in \mathbb{Z}_+$ n $\in \mathbb{Z}_+$ m+n $\in \mathbb{Z}_+$
using pos˙int˙closed˙add˙unfolded by auto
with A1 have
f^{-1}(m+n) $\in \mathbb{Z}$ f^{-1}(m) $\in \mathbb{Z}$ f^{-1}(n) $\in \mathbb{Z}$
using Int˙ZF˙2˙4˙L2 PositiveSet˙def by auto
with I T have
ε(f,x) $\in \mathbb{Z} \wedge$ f(ε(f,x)) $\leq$ U $\wedge$ N $\leq$ f(ε(f,x))
using Int˙ZF˙1˙1˙L5 by auto
} thus thesis by simp
qed
ultimately show $\exists$ M.$\forall$ x$\in\mathbb{Z}_+\times\mathbb{Z}_+$. abs($\varepsilon$(f,x)) $\leq$ 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 $\in \mathcal{S}_+$ and A2: g = $-\langle$p,f^{-1}(p)$\rangle$. p$\in\mathbb{Z}_+$"
shows
g : $\mathbb{Z}_+\rightarrow\mathbb{Z}_+$
g : $\mathbb{Z}_+\rightarrow\mathbb{Z}$
proof -
from A1 have

$\forall$p$\in\mathbb{Z}_+$. $f^{-1}(p) \in \mathbb{Z}_+$
$\forall$p$\in\mathbb{Z}_+$. $f^{-1}(p) \in \mathbb{Z}$
using Int˙ZF˙2˙4˙L2 PositiveSet˙def by auto
with A2 show
g : $\mathbb{Z}_+\to\mathbb{Z}_+$ and g : $\mathbb{Z}_+\to\mathbb{Z}$
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 $\in \mathcal{S}_+$ and A2: g = $-\langle p,f^{-1}(p)\rangle$. p$\in\mathbb{Z}_+$″ and A3: p$\in\mathbb{Z}_+$
shows g(p) = $f^{-1}(p)$
proof -
from A1 A2 have g : $\mathbb{Z}_+\to\mathbb{Z}_+$ 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 $\in \mathcal{S}_+$ and
A2: $\forall$m$\in\mathbb{Z}_+$. $f^{-1}(m)$-$\mathbf{1} \in \mathbb{Z}_+$ and
A3: g = $-\langle p,f^{-1}(p)\rangle$. p$\in\mathbb{Z}_+$″
shows OddExtension($\mathbb{Z}$,IntegerAddition,IntegerOrder,g) $\in \mathcal{S}$
proof -
from A1 A2 have $\exists$L. $\forall$x$\in\mathbb{Z}_+\times\mathbb{Z}_+$. abs($\varepsilon$(f,x)) $\leq$ L
using Int˙ZF˙2˙4˙L8 by simp
then obtain L where I: $\forall$x$\in\mathbb{Z}_+\times\mathbb{Z}_+$. abs($\varepsilon$(f,x)) $\leq$ L
by auto
from A1 A3 have g : $\mathbb{Z}_+\to\mathbb{Z}$ using Int˙ZF˙2˙4˙L9
by simp
moreover have $\forall$m$\in\mathbb{Z}_+$. $\forall$n$\in\mathbb{Z}_+$. abs(δ(g,m,n)) $\leq$ L
proof-
{ fix m n
assume A4: m$\in\mathbb{Z}_+$ n$\in\mathbb{Z}_+$
then have $\langle m,n\rangle \in \mathbb{Z}_+\times\mathbb{Z}_+$ by simp
with I have abs(ε(f,$\langle m,n\rangle$)) $\leq$ L by simp
moreover have ε(f,$\langle m,n\rangle$) = 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)) $\leq$ L by simp
} thus $\forall$m$\in\mathbb{Z}_+$. $\forall$n$\in\mathbb{Z}_+$. abs(δ(g,m,n)) $\leq$ 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 $\in \mathcal{S}_+$ 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
```

46.3 Completeness

In this section we consider properties of slopes that are needed for the proof of completeness of real numbers constructred in Real˙ZF˙1.thy. In particular

we consider properties of embedding of integers into the set of slopes by the mapping $m \mapsto m^S$, where m^S is defined by $m^S(n) = m \cdot n$.

If m is an integer, then m^S is a slope whose value is $m \cdot n$ for every integer.

```
lemma (in int1) Int˙ZF˙2˙5˙L1: assumes A1: m ∈ ℤ
  shows
  ∀n ∈ ℤ. (m^S)(n) = m·n
  m^S ∈ 𝒮
proof -
  from A1 have I: m^S:ℤ→ℤ
    using Int˙ZF˙1˙1˙L5 ZF˙fun˙from˙total by simp
  then show II: ∀n ∈ ℤ. (m^S)(n) = m·n using ZF˙fun˙from˙tot˙val
    by simp
  { fix n k
    assume A2: n∈ℤ  k∈ℤ
    with A1 have T: m·n ∈ ℤ  m·k ∈ ℤ
      using Int˙ZF˙1˙1˙L5 by auto
    from A1 A2 II T  have δ(m^S,n,k) = m·k - m·k
      using Int˙ZF˙1˙1˙L5 Int˙ZF˙1˙1˙L1 Int˙ZF˙1˙2˙L3
      by simp
    also from T have ... = 0 using Int˙ZF˙1˙1˙L4
      by simp
    finally have δ(m^S,n,k) = 0 by simp
    then have abs(δ(m^S,n,k)) ≤ 0
      using Int˙ZF˙2˙L18 int˙zero˙one˙are˙int int˙ord˙is˙refl refl˙def
      by simp
  } then have ∀n∈ℤ.∀k∈ℤ. abs(δ(m^S,n,k)) ≤ 0
    by simp
  with I show  m^S ∈ 𝒮 by (rule Int˙ZF˙2˙1˙L5)
qed
```

For any slope f there is an integer m such that there is some slope g that is almost equal to m^S and dominates f in the sense that $f \leq g$ on positive integers (which implies that either g is almost equal to f or $g - f$ is a positive slope. This will be used in Real˙ZF˙1.thy to show that for any real number there is an integer that (whose real embedding) is greater or equal.

```
lemma (in int1) Int˙ZF˙2˙5˙L2: assumes A1: f ∈ 𝒮
  shows ∃m∈ℤ. ∃g∈𝒮. (m^S∼g ∧ (f∼g ∨ g+(-f) ∈ 𝒮₊))
proof -
  from A1 have
    ∃m k. m∈ℤ ∧ k∈ℤ ∧ (∀p∈ℤ. abs(f(p)) ≤ m·abs(p)+k)
    using Arthan˙Lem˙8 by simp
  then obtain m k where I: m∈ℤ and II: k∈ℤ and
    III: ∀p∈ℤ. abs(f(p)) ≤ m·abs(p)+k
    by auto
  let g = −⟨n,m^S(n) +k⟩. n∈ℤ″
  from I have IV: m^S ∈ 𝒮 using Int˙ZF˙2˙5˙L1 by simp
  with II have V: g∈𝒮 and VI: m^S∼g using Int˙ZF˙2˙1˙L33
```

```
    by auto
  { fix n assume A2: n∈ℤ+
    with A1 have f(n) ∈ ℤ
      using Int˙ZF˙2˙1˙L2B PositiveSet˙def by simp
    then have f(n) ≤ abs(f(n)) using Int˙ZF˙2˙L19C
      by simp
    moreover
    from III A2 have abs(f(n)) ≤ m·abs(n) + k
      using PositiveSet˙def by simp
    with A2 have abs(f(n)) ≤ m·n+k
      using Int˙ZF˙1˙5˙L4A by simp
    ultimately have f(n) ≤ m·n+k
      by (rule Int˙order˙transitive)
    moreover
    from II IV A2 have g(n) = (m^S)(n)+k
      using Int˙ZF˙2˙1˙L33 PositiveSet˙def by simp
    with I A2 have g(n) = m·n+k
      using Int˙ZF˙2˙5˙L1 PositiveSet˙def by simp
    ultimately have f(n) ≤ g(n)
      by simp
  } then have ∀n∈ℤ+. f(n) ≤ g(n)
    by simp
  with A1 V have f∼g ∨ g + (-f) ∈ 𝒮+
    using Int˙ZF˙2˙3˙L4C by simp
  with I V VI show thesis by auto
qed
```

The negative of an integer embeds in slopes as a negative of the orgiginal embedding.

```
lemma (in int1) Int˙ZF˙2˙5˙L3: assumes A1:  m ∈ ℤ
  shows (-m)^S = -(m^S)
proof -
  from A1 have (-m)^S: ℤ→ℤ and (-(m^S)): ℤ→ℤ
    using Int˙ZF˙1˙1˙L4 Int˙ZF˙2˙5˙L1 AlmostHoms˙def Int˙ZF˙2˙1˙L12
    by auto
  moreover have ∀n∈ℤ. ((-m)^S)(n) = (-(m^S))(n)
  proof
    fix n assume A2: n∈ℤ
    with A1 have
      ((-m)^S)(n) = (-m)·n
      (-(m^S))(n) = -(m·n)
      using Int˙ZF˙1˙1˙L4 Int˙ZF˙2˙5˙L1 Int˙ZF˙2˙1˙L12A
      by auto
    with A1 A2 show ((-m)^S)(n) = (-(m^S))(n)
      using Int˙ZF˙1˙1˙L5 by simp
  qed
  ultimately show (-m)^S = -(m^S) using fun˙extension˙iff
    by simp
qed
```

The sum of embeddings is the embeding of the sum.

```
lemma (in int1) Int˙ZF˙2˙5˙L3A: assumes A1: m∈ℤ  k∈ℤ
  shows (m^S) + (k^S) = ((m+k)^S)
proof -
  from A1 have T1: m+k ∈ ℤ using Int˙ZF˙1˙1˙L5
    by simp
  with A1 have T2:
    (m^S) ∈ 𝒮  (k^S) ∈ 𝒮
    (m+k)^S  ∈ 𝒮
    (m^S) + (k^S) ∈ 𝒮
    using Int˙ZF˙2˙5˙L1 Int˙ZF˙2˙1˙L12C by auto
  then have
    (m^S) + (k^S) : ℤ→ℤ
    (m+k)^S : ℤ→ℤ
    using AlmostHoms˙def by auto
  moreover have ∀n∈ℤ. ((m^S) + (k^S))(n) = ((m+k)^S)(n)
  proof
    fix n assume A2: n∈ℤ
    with A1 T1 T2 have  ((m^S) + (k^S))(n) = (m+k)·n
      using Int˙ZF˙2˙1˙L12B Int˙ZF˙2˙5˙L1 Int˙ZF˙1˙1˙L1
      by simp
    also from T1 A2 have ... = ((m+k)^S)(n)
      using Int˙ZF˙2˙5˙L1 by simp
    finally show ((m^S) + (k^S))(n) = ((m+k)^S)(n)
      by simp
  qed
  ultimately show (m^S) + (k^S) = ((m+k)^S)
    using fun˙extension˙iff by simp
qed
```

The composition of embeddings is the embeding of the product.

```
lemma (in int1) Int˙ZF˙2˙5˙L3B: assumes A1: m∈ℤ  k∈ℤ
  shows (m^S) ∘ (k^S) = ((m·k)^S)
proof -
  from A1 have T1: m·k ∈ ℤ using Int˙ZF˙1˙1˙L5
    by simp
  with A1 have T2:
    (m^S) ∈ 𝒮  (k^S) ∈ 𝒮
    (m·k)^S  ∈ 𝒮
    (m^S) ∘ (k^S) ∈ 𝒮
    using Int˙ZF˙2˙5˙L1 Int˙ZF˙2˙1˙L11 by auto
  then have
    (m^S) ∘ (k^S) : ℤ→ℤ
    (m·k)^S : ℤ→ℤ
    using AlmostHoms˙def by auto
  moreover have ∀n∈ℤ. ((m^S) ∘ (k^S))(n) = ((m·k)^S)(n)
  proof
    fix n assume A2: n∈ℤ
    with A1 T2 have
```

```
    ((m^S) ∘ (k^S))(n) = (m^S)(k·n)
     using Int˙ZF˙2˙1˙L10 Int˙ZF˙2˙5˙L1 by simp
  moreover
  from A1 A2 have k·n ∈ ℤ using Int˙ZF˙1˙1˙L5
    by simp
  with A1 A2 have (m^S)(k·n) = m·k·n
    using Int˙ZF˙2˙5˙L1 Int˙ZF˙1˙1˙L7 by simp
  ultimately have ((m^S) ∘ (k^S))(n) = m·k·n
    by simp
  also from T1 A2 have m·k·n = ((m·k)^S)(n)
    using Int˙ZF˙2˙5˙L1 by simp
  finally show ((m^S) ∘ (k^S))(n) = ((m·k)^S)(n)
    by simp
 qed
 ultimately show (m^S) ∘ (k^S) = ((m·k)^S)
  using fun˙extension˙iff by simp
qed
```

Embedding integers in slopes preserves order.

```
lemma (in int1) Int˙ZF˙2˙5˙L4: assumes A1:  m≤n
 shows (m^S) ∼ (n^S) ∨ (n^S)+(-(m^S)) ∈ 𝒮_+
proof -
 from A1 have m^S ∈ 𝒮 and n^S ∈ 𝒮
  using Int˙ZF˙2˙L1A Int˙ZF˙2˙5˙L1 by auto
 moreover from A1 have ∀k∈ℤ_+. (m^S)(k) ≤ (n^S)(k)
  using Int˙ZF˙1˙3˙L13B Int˙ZF˙2˙L1A PositiveSet˙def Int˙ZF˙2˙5˙L1
  by simp
 ultimately show thesis using Int˙ZF˙2˙3˙L4C
  by simp
qed
```

We aim at showing that $m \mapsto m^S$ is an injection modulo the relation of almost equality. To do that we first show that if m^S has finite range, then $m = 0$.

```
lemma (in int1) Int˙ZF˙2˙5˙L5:
 assumes m∈ℤ and m^S ∈ FinRangeFunctions(ℤ,ℤ)
 shows m=𝟬
 using assms FinRangeFunctions˙def Int˙ZF˙2˙5˙L1 AlmostHoms˙def
  func˙imagedef Int˙ZF˙1˙6˙L8 by simp
```

Embeddings of two integers are almost equal only if the integers are equal.

```
lemma (in int1) Int˙ZF˙2˙5˙L6:
 assumes A1: m∈ℤ  k∈ℤ and A2: (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($\mathbb{Z}$,$\mathbb{Z}$)
using Int˙ZF˙2˙5˙L1 Int˙ZF˙2˙1˙L9D by simp
ultimately have (m-k)^S ∈ FinRangeFunctions($\mathbb{Z}$,$\mathbb{Z}$)
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
$\mathbf{1}^S$ = id($\mathbb{Z}$)
$\mathbf{0}^S$ ∈ FinRangeFunctions($\mathbb{Z}$,$\mathbb{Z}$)
proof -
have id($\mathbb{Z}$) = −⟨x,x⟩. x∈$\mathbb{Z}$″
using id˙def by blast
then show $\mathbf{1}^S$ = id($\mathbb{Z}$) using Int˙ZF˙1˙1˙L4 by simp
have −$\mathbf{0}^S$(n). n∈$\mathbb{Z}$″ = −**0**·n. n∈$\mathbb{Z}$″
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 −$\mathbf{0}^S$(n). n∈$\mathbb{Z}$″ = −**0**″ by simp
then have −$\mathbf{0}^S$(n). n∈$\mathbb{Z}$″ ∈ Fin($\mathbb{Z}$)
using int˙zero˙one˙are˙int Finite1˙L16 by simp
moreover have $\mathbf{0}^S$: $\mathbb{Z}$→$\mathbb{Z}$
using int˙zero˙one˙are˙int Int˙ZF˙2˙5˙L1 AlmostHoms˙def
by simp
ultimately show $\mathbf{0}^S$ ∈ FinRangeFunctions($\mathbb{Z}$,$\mathbb{Z}$)
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 ∈ $\mathcal{S}$ and A2: N ∈ $\mathbb{Z}$ M ∈ $\mathbb{Z}$ and
A3: ∀n∈$\mathbb{Z}_+$. M·n ≤ f(N·n)
shows M^S ∼ f∘(N^S) ∨ (f∘(N^S)) + (-(M^S)) ∈ $\mathcal{S}_+$
proof -
from A1 A2 have M^S ∈ $\mathcal{S}$ f∘(N^S) ∈ $\mathcal{S}$
using Int˙ZF˙2˙5˙L1 Int˙ZF˙2˙1˙L11 by auto
moreover from A1 A2 A3 have ∀n∈$\mathbb{Z}_+$. (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 $\in \mathcal{S}$ and A2: N $\in \mathbb{Z}$ M $\in \mathbb{Z}$ and
A3: $\forall$n$\in\mathbb{Z}_+$. f(N·n) $\leq$ M·n
shows f$\circ$(N^S) $\sim$ (M^S) $\vee$ (M^S) + (-(f$\circ$(N^S))) $\in \mathcal{S}_+$
proof -
from A1 A2 have f$\circ$(N^S) $\in \mathcal{S}$ M^S $\in \mathcal{S}$
using Int˙ZF˙2˙5˙L1 Int˙ZF˙2˙1˙L11 by auto
moreover from A1 A2 A3 have $\forall$n$\in\mathbb{Z}_+$. (f$\circ$(N^S))(n) $\leq$ (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

47 Construction real numbers - the generic part

theory Real˙ZF imports Int˙ZF˙IML Ring˙ZF˙1

begin

The goal of the Real˙ZF series of theory files is to provide a contruction of the set of real numbers. There are several ways to construct real numbers. Most common start from the rational numbers and use Dedekind cuts or Cauchy sequences. Real˙ZF˙x.thy series formalizes an alternative approach that constructs real numbers directly from the group of integers. Our formalization is mostly based on [2]. Different variants of this contruction are also described in [1] and [3]. I recommend to read these papers, but for the impatient here is a short description: we take a set of maps $s : Z \to Z$ such that the set $\{s(m+n) - s(m) - s(n)\}_{n,m\in Z}$ is finite (Z means the integers here). We call these maps slopes. Slopes form a group with the natural addition $(s+r)(n) = s(n)+r(n)$. The maps such that the set $s(Z)$ is finite (finite range functions) form a subgroup of slopes. The additive group of real numbers is defined as the quotient group of slopes by the (sub)group of finite range functions. The multiplication is defined as the projection of the composition of slopes into the resulting quotient (coset) space.

47.1 The definition of real numbers

This section contains the construction of the ring of real numbers as classes of slopes - integer almost homomorphisms. The real definitions are in Group˙ZF˙2 theory, here we just specialize the definitions of almost homomorphisms, their equivalence and operations to the additive group of integers from the general case of abelian groups considered in Group˙ZF˙2.

The set of slopes is defined as the set of almost homomorphisms on the additive group of integers.

definition
Slopes ≡ AlmostHoms(int,IntegerAddition)

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

definition
SlopeOp1 ≡ AlHomOp1(int,IntegerAddition)

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

definition
SlopeOp2 ≡ AlHomOp2(int,IntegerAddition)

Bounded integer maps are functions from integers to integers that have finite range. They play a role of zero in the set of real numbers we are constructing.

definition
BoundedIntMaps ≡ FinRangeFunctions(int,int)

Bounded integer maps form a normal subgroup of slopes. The equivalence relation on slopes is the (group) quotient relation defined by this subgroup.

definition
SlopeEquivalenceRel ≡ QuotientGroupRel(Slopes,SlopeOp1,BoundedIntMaps)

The set of real numbers is the set of equivalence classes of slopes.

definition
RealNumbers ≡ Slopes//SlopeEquivalenceRel

The addition on real numbers is defined as the projection of pointwise addition of slopes on the quotient. This means that the additive group of real numbers is the quotient group: the group of slopes (with pointwise addition) defined by the normal subgroup of bounded integer maps.

definition
RealAddition ≡ ProjFun2(Slopes,SlopeEquivalenceRel,SlopeOp1)

Multiplication is defined as the projection of composition of slopes on the quotient. The fact that it works is probably the most surprising part of the construction.

```
definition
  RealMultiplication ≡ ProjFun2(Slopes,SlopeEquivalenceRel,SlopeOp2)
```

We first show that we can use theorems proven in some proof contexts (locales). The locale group1 requires assumption that we deal with an abelian group. The next lemma allows to use all theorems proven in the context called group1.

```
lemma Real˙ZF˙1˙L1: shows group1(int,IntegerAddition)
  using group1˙axioms.intro group1˙def Int˙ZF˙1˙T2 by simp
```

Real numbers form a ring. This is a special case of the theorem proven in Ring˙ZF˙1.thy, where we show the same in general for almost homomorphisms rather than slopes.

```
theorem Real˙ZF˙1˙T1: shows IsAring(RealNumbers,RealAddition,RealMultiplication)
proof -
  let AH = AlmostHoms(int,IntegerAddition)
  let Op1 = AlHomOp1(int,IntegerAddition)
  let FR = FinRangeFunctions(int,int)
  let Op2 = AlHomOp2(int,IntegerAddition)
  let R = QuotientGroupRel(AH,Op1,FR)
  let A = ProjFun2(AH,R,Op1)
  let M = ProjFun2(AH,R,Op2)
  have IsAring(AH//R,A,M) using Real˙ZF˙1˙L1 group1.Ring˙ZF˙1˙1˙T1
    by simp
  then show thesis using Slopes˙def SlopeOp2˙def SlopeOp1˙def
    BoundedIntMaps˙def SlopeEquivalenceRel˙def RealNumbers˙def
    RealAddition˙def RealMultiplication˙def by simp
qed
```

We can use theorems proven in group0 and group1 contexts applied to the group of real numbers.

```
lemma Real˙ZF˙1˙L2: shows
  group0(RealNumbers,RealAddition)
  RealAddition –is commutative on″ RealNumbers
  group1(RealNumbers,RealAddition)
proof -
  have
    IsAgroup(RealNumbers,RealAddition)
    RealAddition –is commutative on″ RealNumbers
    using Real˙ZF˙1˙T1 IsAring˙def by auto
  then show
    group0(RealNumbers,RealAddition)
    RealAddition –is commutative on″ RealNumbers
    group1(RealNumbers,RealAddition)
    using group1˙axioms.intro group0˙def group1˙def
    by auto
qed
```

Let's define some notation.

```
locale real0 =

  fixes real (ℝ)
  defines real˙def [simp]: ℝ ≡ RealNumbers

  fixes ra (infixl + 69)
  defines ra˙def [simp]: a+ b ≡ RealAddition⟨a,b⟩

  fixes rminus (- ˙ 72)
  defines rminus˙def [simp]:-a ≡ GroupInv(ℝ,RealAddition)(a)

  fixes rsub (infixl - 69)
  defines rsub˙def [simp]: a-b ≡ a+(-b)

  fixes rm (infixl · 70)
  defines rm˙def [simp]: a·b ≡ RealMultiplication⟨a,b⟩

  fixes rzero (𝟎)
  defines rzero˙def [simp]:
  𝟎 ≡ TheNeutralElement(RealNumbers,RealAddition)

  fixes rone (𝟏)
  defines rone˙def [simp]:
  𝟏 ≡ TheNeutralElement(RealNumbers,RealMultiplication)

  fixes rtwo (𝟐)
  defines rtwo˙def [simp]: 𝟐  ≡ 𝟏+𝟏

  fixes non˙zero (ℝ0)
  defines non˙zero˙def[simp]: ℝ0 ≡ ℝ-–𝟎″

  fixes inv (˙−1  [90] 91)
  defines inv˙def[simp]:
  a−1 ≡ GroupInv(ℝ0,restrict(RealMultiplication,ℝ0×ℝ0))(a)
```

In real0 context all theorems proven in the ring0, context are valid.

```
lemma (in real0) Real˙ZF˙1˙L3: shows
  ring0(ℝ,RealAddition,RealMultiplication)
  using Real˙ZF˙1˙T1 ring0˙def ring0.Ring˙ZF˙1˙L1
  by auto
```

Lets try out our notation to see that zero and one are real numbers.

```
lemma (in real0) Real˙ZF˙1˙L4: shows 𝟎∈ℝ  𝟏∈ℝ
  using Real˙ZF˙1˙L3 ring0.Ring˙ZF˙1˙L2 by auto
```

The lemma below lists some properties that require one real number to state.

```
lemma (in real0) Real˙ZF˙1˙L5: assumes A1: a∈ℝ
  shows
  (-a) ∈ ℝ
```

(-(-a)) = a
a+**0** = a
0+a = a
a·**1** = a
1·a = a
a-a = **0**
a-**0** = a
using assms Real˙ZF˙1˙L3 ring0.Ring˙ZF˙1˙L3 by auto

The lemma below lists some properties that require two real numbers to state.

lemma (in real0) Real˙ZF˙1˙L6: assumes a∈ℝ b∈ℝ
shows
a+b ∈ ℝ
a-b ∈ ℝ
a·b ∈ ℝ
a+b = b+a
(-a)·b = -(a·b)
a·(-b) = -(a·b)
using assms Real˙ZF˙1˙L3 ring0.Ring˙ZF˙1˙L4 ring0.Ring˙ZF˙1˙L7
by auto

Multiplication of reals is associative.

lemma (in real0) Real˙ZF˙1˙L6A: assumes a∈ℝ b∈ℝ c∈ℝ
shows a·(b·c) = (a·b)·c
using assms Real˙ZF˙1˙L3 ring0.Ring˙ZF˙1˙L11
by simp

Addition is distributive with respect to multiplication.

lemma (in real0) Real˙ZF˙1˙L7: assumes a∈ℝ b∈ℝ c∈ℝ
shows
a·(b+c) = a·b + a·c
(b+c)·a = b·a + c·a
a·(b-c) = a·b - a·c
(b-c)·a = b·a - c·a
using assms Real˙ZF˙1˙L3 ring0.ring˙oper˙distr ring0.Ring˙ZF˙1˙L8
by auto

A simple rearrangement with four real numbers.

lemma (in real0) Real˙ZF˙1˙L7A:
assumes a∈ℝ b∈ℝ c∈ℝ d∈ℝ
shows a-b + (c-d) = a+c-b-d
using assms Real˙ZF˙1˙L2 group0.group0˙4˙L8A by simp

RealAddition is defined as the projection of the first operation on slopes (that is, slope addition) on the quotient (slopes divided by the "almost equal" relation. The next lemma plays with definitions to show that this is the same as the operation induced on the appriopriate quotient group.

The names AH, Op1 and FR are used in group1 context to denote almost homomorphisms, the first operation on AH and finite range functions resp.

```
lemma Real˙ZF˙1˙L8: assumes
  AH = AlmostHoms(int,IntegerAddition) and
  Op1 = AlHomOp1(int,IntegerAddition) and
  FR = FinRangeFunctions(int,int)
  shows RealAddition = QuotientGroupOp(AH,Op1,FR)
  using assms RealAddition˙def SlopeEquivalenceRel˙def
    QuotientGroupOp˙def Slopes˙def SlopeOp1˙def BoundedIntMaps˙def
  by simp
```

The symbol **0** in the real0 context is defined as the neutral element of real addition. The next lemma shows that this is the same as the neutral element of the appriopriate quotient group.

```
lemma (in real0) Real˙ZF˙1˙L9: assumes
  AH = AlmostHoms(int,IntegerAddition) and
  Op1 = AlHomOp1(int,IntegerAddition) and
  FR = FinRangeFunctions(int,int) and
  r = QuotientGroupRel(AH,Op1,FR)
  shows
  TheNeutralElement(AH//r,QuotientGroupOp(AH,Op1,FR)) = 0
  SlopeEquivalenceRel = r
  using assms Slopes˙def Real˙ZF˙1˙L8 RealNumbers˙def
    SlopeEquivalenceRel˙def SlopeOp1˙def BoundedIntMaps˙def
  by auto
```

Zero is the class of any finite range function.

```
lemma (in real0) Real˙ZF˙1˙L10:
  assumes A1: s ∈ Slopes
  shows SlopeEquivalenceRel–s″ = 0 ⟷ s ∈ BoundedIntMaps
proof -
  let AH = AlmostHoms(int,IntegerAddition)
  let Op1 = AlHomOp1(int,IntegerAddition)
  let FR = FinRangeFunctions(int,int)
  let r = QuotientGroupRel(AH,Op1,FR)
  let e = TheNeutralElement(AH//r,QuotientGroupOp(AH,Op1,FR))
  from A1 have
    group1(int,IntegerAddition)
    s∈AH
    using Real˙ZF˙1˙L1 Slopes˙def
    by auto
  then have r–s″ = e ⟷ s ∈ FR
    using group1.Group˙ZF˙3˙3˙L5 by simp
  moreover have
    r = SlopeEquivalenceRel
    e = 0
    FR = BoundedIntMaps
    using SlopeEquivalenceRel˙def Slopes˙def SlopeOp1˙def
```

```
    BoundedIntMaps˙def Real˙ZF˙1˙L9 by auto
  ultimately show thesis by simp
qed
```

We will need a couple of results from Group˙ZF˙3.thy The first two that state that the definition of addition and multiplication of real numbers are consistent, that is the result does not depend on the choice of the slopes representing the numbers. The second one implies that what we call SlopeEquivalenceRel is actually an equivalence relation on the set of slopes. We also show that the neutral element of the multiplicative operation on reals (in short number 1) is the class of the identity function on integers.

```
lemma Real˙ZF˙1˙L11: shows
  Congruent2(SlopeEquivalenceRel,SlopeOp1)
  Congruent2(SlopeEquivalenceRel,SlopeOp2)
  SlopeEquivalenceRel ⊆ Slopes × Slopes
  equiv(Slopes, SlopeEquivalenceRel)
  SlopeEquivalenceRel–id(int)″ =
  TheNeutralElement(RealNumbers,RealMultiplication)
  BoundedIntMaps ⊆ Slopes
proof -
  let G = int
  let f = IntegerAddition
  let AH = AlmostHoms(int,IntegerAddition)
  let Op1 = AlHomOp1(int,IntegerAddition)
  let Op2 = AlHomOp2(int,IntegerAddition)
  let FR = FinRangeFunctions(int,int)
  let R = QuotientGroupRel(AH,Op1,FR)
   have
     Congruent2(R,Op1)
     Congruent2(R,Op2)
    using Real˙ZF˙1˙L1 group1.Group˙ZF˙3˙4˙L13A group1.Group˙ZF˙3˙3˙L4
    by auto
  then show
    Congruent2(SlopeEquivalenceRel,SlopeOp1)
    Congruent2(SlopeEquivalenceRel,SlopeOp2)
    using SlopeEquivalenceRel˙def SlopeOp1˙def Slopes˙def
      BoundedIntMaps˙def SlopeOp2˙def by auto
  have equiv(AH,R)
    using Real˙ZF˙1˙L1 group1.Group˙ZF˙3˙3˙L3 by simp
  then show equiv(Slopes,SlopeEquivalenceRel)
    using BoundedIntMaps˙def SlopeEquivalenceRel˙def SlopeOp1˙def Slopes˙def
    by simp
  then show SlopeEquivalenceRel ⊆ Slopes × Slopes
    using equiv˙type by simp
  have R–id(int)″ = TheNeutralElement(AH//R,ProjFun2(AH,R,Op2))
    using Real˙ZF˙1˙L1 group1.Group˙ZF˙3˙4˙T2 by simp
  then show  SlopeEquivalenceRel–id(int)″ =
    TheNeutralElement(RealNumbers,RealMultiplication)
```

```
    using Slopes˙def RealNumbers˙def
    SlopeEquivalenceRel˙def SlopeOp1˙def BoundedIntMaps˙def
    RealMultiplication˙def SlopeOp2˙def
    by simp
  have FR ⊆ AH using Real˙ZF˙1˙L1 group1.Group˙ZF˙3˙3˙L1
    by simp
  then show BoundedIntMaps ⊆ Slopes
    using BoundedIntMaps˙def Slopes˙def by simp
qed
```

A one-side implication of the equivalence from Real˙ZF˙1˙L10: the class of a bounded integer map is the real zero.

```
lemma (in real0) Real˙ZF˙1˙L11A: assumes s ∈ BoundedIntMaps
  shows SlopeEquivalenceRel–s″ = 0
  using assms Real˙ZF˙1˙L11 Real˙ZF˙1˙L10 by auto
```

The next lemma is rephrases the result from Group˙ZF˙3.thy that says that the negative (the group inverse with respect to real addition) of the class of a slope is the class of that slope composed with the integer additive group inverse. The result and proof is not very readable as we use mostly generic set theory notation with long names here. Real˙ZF˙1.thy contains the same statement written in a more readable notation: $[-s] = -[s]$.

```
lemma (in real0) Real˙ZF˙1˙L12: assumes A1: s ∈ Slopes and
  Dr: r = QuotientGroupRel(Slopes,SlopeOp1,BoundedIntMaps)
  shows r–GroupInv(int,IntegerAddition) O s″ = -(r–s″)
proof -
  let G = int
  let f = IntegerAddition
  let AH = AlmostHoms(int,IntegerAddition)
  let Op1 = AlHomOp1(int,IntegerAddition)
  let FR = FinRangeFunctions(int,int)
  let F = ProjFun2(Slopes,r,SlopeOp1)
  from A1 Dr have
    group1(G, f)
    s ∈ AlmostHoms(G, f)
    r = QuotientGroupRel(
    AlmostHoms(G, f), AlHomOp1(G, f), FinRangeFunctions(G, G))
    and F = ProjFun2(AlmostHoms(G, f), r, AlHomOp1(G, f))
    using Real˙ZF˙1˙L1 Slopes˙def SlopeOp1˙def BoundedIntMaps˙def
    by auto
  then have
    r–GroupInv(G, f) O s″ =
    GroupInv(AlmostHoms(G, f) // r, F)(r  –s″)
    using group1.Group˙ZF˙3˙3˙L6 by simp
  with Dr show thesis
    using RealNumbers˙def Slopes˙def SlopeEquivalenceRel˙def RealAddition˙def
    by simp
qed
```

Two classes are equal iff the slopes that represent them are almost equal.

```
lemma Real˙ZF˙1˙L13: assumes s ∈ Slopes  p ∈ Slopes
  and r = SlopeEquivalenceRel
  shows r–s″ = r–p″ ⟷ ⟨s,p⟩ ∈  r
  using assms Real˙ZF˙1˙L11 eq˙equiv˙class equiv˙class˙eq
  by blast
```

Identity function on integers is a slope. Thislemma 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.

```
lemma Real˙ZF˙1˙L14: shows id(int) ∈ Slopes
proof -
  have id(int) ∈ AlmostHoms(int,IntegerAddition)
    using Real˙ZF˙1˙L1 group1.Group˙ZF˙3˙4˙L15
    by simp
  then show thesis using Slopes˙def by simp
qed

end
```

48 Construction of real numbers

```
theory Real˙ZF˙1 imports Real˙ZF Int˙ZF˙3 OrderedField˙ZF

begin
```

In this theory file we continue the construction of real numbers started in Real˙ZF to a succesful conclusion. We put here those parts of the construction that can not be done in the general settings of abelian groups and require integers.

48.1 Definitions and notation

In this section we define notions and notation needed for the rest of the construction.

We define positive slopes as those that take an infinite number of posititive values on the positive integers (see Int˙ZF˙2 for properties of positive slopes).

```
definition
  PositiveSlopes ≡ –s ∈ Slopes.
  s(PositiveIntegers) ∩  PositiveIntegers ∉ Fin(int)″
```

The order on the set of real numbers is constructed by specifying the set of positive reals. This set is defined as the projection of the set of positive slopes.

definition
PositiveReals ≡ –SlopeEquivalenceRel–s″. s ∈ PositiveSlopes″

The order relation on real numbers is constructed from the set of positive elements in a standard way (see section "Alternative definitions" in OrderedGroup˙ZF.)

definition
OrderOnReals ≡ OrderFromPosSet(RealNumbers,RealAddition,PositiveReals)

The next locale extends the locale real0 to define notation specific to the construction of real numbers. The notation follows the one defined in Int˙ZF˙2.thy. If m is an integer, then the real number which is the class of the slope $n \mapsto m \cdot n$ is denoted 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 ($\mathcal{S}$)
defines slopes˙def[simp]: $\mathcal{S}$ ≡ AlmostHoms(int,IntegerAddition)

fixes posslopes ($\mathcal{S}_+$)
defines posslopes˙def[simp]: $\mathcal{S}_+$ ≡ PositiveSlopes

fixes slope˙class ([˙])
defines slope˙class˙def[simp]: [f] ≡ SlopeEquivalenceRel–f″

fixes slope˙neg (-˙ [90] 91)
defines slope˙neg˙def[simp]: -s ≡ GroupInv(int,IntegerAddition) O s

fixes lesseqr (infix $\leq$ 60)
defines lesseqr˙def[simp]: a $\leq$ b $\equiv$ $\langle$a,b$\rangle$ $\in$ OrderOnReals

fixes sless (infix ¡ 60)
defines sless˙def[simp]: a ¡ b $\equiv$ a$\leq$b $\wedge$ a$\neq$b

fixes positivereals ($\mathbb{R}_+$)
defines positivereals˙def[simp]: $\mathbb{R}_+$ $\equiv$ PositiveSet($\mathbb{R}$,RealAddition,OrderOnReals)

fixes intembed ($\cdot^R$ [90] 91)
defines intembed˙def[simp]:
m^R $\equiv$ [–$\langle$n,IntegerMultiplication$\langle$m,n$\rangle$ $\rangle$. n $\in$ int″]

fixes floor ($\lfloor$ ˙ $\rfloor$)
defines floor˙def[simp]:
$\lfloor$a$\rfloor$ $\equiv$ Maximum(IntegerOrder,–m $\in$ int. m^R $\leq$ a″)

fixes Γ
defines Γ˙def[simp]: Γ(S,p) $\equiv$ Maximum(IntegerOrder,–$\lfloor p^R \cdot x \rfloor$. x$\in$S″)

fixes ia (infixl + 69)
defines ia˙def[simp]: a+b $\equiv$ IntegerAddition$\langle$ a,b$\rangle$

fixes iminus (- ˙ 72)
defines iminus˙def[simp]: -a $\equiv$ GroupInv(int,IntegerAddition)(a)

fixes isub (infixl - 69)
defines isub˙def[simp]: a-b $\equiv$ a+ (- b)

fixes intpositives ($\mathbb{Z}_+$)
defines intpositives˙def[simp]:
$\mathbb{Z}_+$ $\equiv$ PositiveSet(int,IntegerAddition,IntegerOrder)

fixes zlesseq (infix $\leq$ 60)
defines lesseq˙def[simp]: m $\leq$ n $\equiv$ $\langle$m,n$\rangle$ $\in$ IntegerOrder

fixes imult (infixl $\cdot$ 70)
defines imult˙def[simp]: a$\cdot$b $\equiv$ IntegerMultiplication$\langle$ a,b$\rangle$

fixes izero ($\mathbf{0}_Z$)
defines izero˙def[simp]: $\mathbf{0}_Z$ $\equiv$ TheNeutralElement(int,IntegerAddition)

fixes ione ($\mathbf{1}_Z$)
defines ione˙def[simp]: $\mathbf{1}_Z$ $\equiv$ TheNeutralElement(int,IntegerMultiplication)

fixes itwo ($\mathbf{2}_Z$)
defines itwo˙def[simp]: $\mathbf{2}_Z$ $\equiv$ $\mathbf{1}_Z$+$\mathbf{1}_Z$

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

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

48.2 Multiplication of real numbers

Multiplication of real numbers is defined as a projection of composition of slopes onto the space of equivalence classes of slopes. Thus, the product of the real numbers given as classes of slopes s and r is defined as the class of $s \circ r$. The goal of this section is to show that multiplication defined this way is commutative.

Let's recall a theorem from Int˙ZF˙2.thy that states that if f, g are slopes, then $f \circ g$ is equivalent to $g \circ f$. Here we conclude from that that the classes of $f \circ g$ and $g \circ f$ are the same.

```
lemma (in real1) Real˙ZF˙1˙1˙L2: assumes A1: f ∈ 𝒮  g ∈ 𝒮
  shows [f∘g] = [g∘f]
proof -
  from A1 have f∘g ∼ g∘f
    using Slopes˙def int1.Arthan˙Th˙9 SlopeOp1˙def BoundedIntMaps˙def
      SlopeEquivalenceRel˙def SlopeOp2˙def by simp
  then show thesis using Real˙ZF˙1˙L11 equiv˙class˙eq
    by simp
qed
```

Classes of slopes are real numbers.

```
lemma (in real1) Real˙ZF˙1˙1˙L3: assumes A1: f ∈ 𝒮
  shows [f] ∈ ℝ
proof -
  from A1 have [f] ∈ Slopes//SlopeEquivalenceRel
    using Slopes˙def quotientI by simp
  then show [f] ∈ ℝ using RealNumbers˙def by simp
qed
```

Each real number is a class of a slope.

```
lemma (in real1) Real˙ZF˙1˙1˙L3A: assumes A1: a∈ℝ
  shows ∃f∈𝒮 . a = [f]
proof -
  from A1 have a ∈ 𝒮//SlopeEquivalenceRel
    using RealNumbers˙def Slopes˙def by simp
  then show thesis using quotient˙def
    by simp
qed
```

It is useful to have the definition of addition and multiplication in the real1 context notation.

```
lemma (in real1) Real˙ZF˙1˙1˙L4:
  assumes A1: f ∈ 𝒮  g ∈ 𝒮
  shows
  [f] + [g] = [f+g]
  [f] · [g] = [f∘g]
proof -
  let r = SlopeEquivalenceRel
  have [f]·[g] = ProjFun2(𝒮,r,SlopeOp2)⟨[f],[g]⟩
    using RealMultiplication˙def Slopes˙def by simp
  also from A1 have ... = [f∘g]
    using Real˙ZF˙1˙L11 EquivClass˙1˙L10 Slopes˙def
    by simp
  finally show [f] · [g] = [f∘g] by simp
  have [f] + [g] = ProjFun2(𝒮,r,SlopeOp1)⟨[f],[g]⟩
    using RealAddition˙def Slopes˙def by simp
  also from A1 have ... = [f+g]
    using Real˙ZF˙1˙L11 EquivClass˙1˙L10 Slopes˙def
    by simp
  finally show [f] + [g] = [f+g] by simp
qed
```

The next lemma is essentially the same as Real˙ZF˙1˙L12, but written in the notation defined in the real1 context. It states that if f is a slope, then $-[f] = [-f]$.

```
lemma (in real1) Real˙ZF˙1˙1˙L4A: assumes f ∈ 𝒮
  shows [-f] = -[f]
  using assms Slopes˙def SlopeEquivalenceRel˙def Real˙ZF˙1˙L12
  by simp
```

Subtracting real numbers correspods to adding the opposite slope.

```
lemma (in real1) Real˙ZF˙1˙1˙L4B: assumes A1: f ∈ 𝒮  g ∈ 𝒮
  shows [f] - [g] = [f+(-g)]
proof -
  from A1 have [f+(-g)] = [f] + [-g]
    using Slopes˙def BoundedIntMaps˙def int1.Int˙ZF˙2˙1˙L12
      Real˙ZF˙1˙1˙L4 by simp
  with A1 show [f] - [g] = [f+(-g)]
    using Real˙ZF˙1˙1˙L4A by simp
qed
```

Multiplication of real numbers is commutative.

```
theorem (in real1) real˙mult˙commute: assumes A1: a∈ℝ  b∈ℝ
  shows a·b = b·a
proof -
  from A1 have
    ∃f∈𝒮 . a = [f]
    ∃g∈𝒮 . b = [g]
    using Real˙ZF˙1˙1˙L3A by auto
```

then obtain f g where
f ∈ $\mathcal{S}$ g ∈ $\mathcal{S}$ and a = [f] b = [g]
by auto
then show a·b = b·a
using Real˙ZF˙1˙1˙L4 Real˙ZF˙1˙1˙L2 by simp
qed

Multiplication is commutative on reals.

lemma real˙mult˙commutative: shows
RealMultiplication –is commutative on″ RealNumbers
using real1.real˙mult˙commute IsCommutative˙def
by simp

The neutral element of multiplication of reals (denoted as **1** in the real1 context) is the class of identity function on integers. This is really shown in Real˙ZF˙1˙L11, here we only rewrite it in the notation used in the real1 context.

lemma (in real1) real˙one˙cl˙identity: shows [id(int)] = **1**
using Real˙ZF˙1˙L11 by simp

If f is bounded, then its class is the neutral element of additive operation on reals (denoted as **0** in the real1 context).

lemma (in real1) real˙zero˙cl˙bounded˙map:
assumes f ∈ BoundedIntMaps shows [f] = **0**
using assms Real˙ZF˙1˙L11A by simp

Two real numbers are equal iff the slopes that represent them are almost equal. This is proven in Real˙ZF˙1˙L13, here we just rewrite it in the notation used in the real1 context.

lemma (in real1) Real˙ZF˙1˙1˙L5:
assumes f ∈ $\mathcal{S}$ g ∈ $\mathcal{S}$
shows [f] = [g] ⟷ f ∼ g
using assms Slopes˙def Real˙ZF˙1˙L13 by simp

If the pair of function belongs to the slope equivalence relation, then their classes are equal. This is convenient, because we don't need to assume that f, g are slopes (follows from the fact that $f \sim g$).

lemma (in real1) Real˙ZF˙1˙1˙L5A: assumes f ∼ g
shows [f] = [g]
using assms Real˙ZF˙1˙L11 Slopes˙def Real˙ZF˙1˙1˙L5
by auto

Identity function on integers is a slope. This is proven in Real˙ZF˙1˙L13, here we just rewrite it in the notation used in the real1 context.

lemma (in real1) id˙on˙int˙is˙slope: shows id(int) ∈ $\mathcal{S}$
using Real˙ZF˙1˙L14 Slopes˙def by simp

A result from Int˙ZF˙2.thy: the identity function on integers is not almost equal to any bounded function.

```
lemma (in real1) Real˙ZF˙1˙1˙L7:
  assumes A1: f ∈ BoundedIntMaps
  shows ¬(id(int) ∼ f)
  using assms Slopes˙def SlopeOp1˙def BoundedIntMaps˙def
    SlopeEquivalenceRel˙def BoundedIntMaps˙def int1.Int˙ZF˙2˙3˙L12
  by simp
```

Zero is not one.

```
lemma (in real1) real˙zero˙not˙one: shows 1≠0
proof -
  { assume A1: 1=0
    have ∃f ∈ 𝒮. 0 = [f]
      using  Real˙ZF˙1˙L4 Real˙ZF˙1˙1˙L3A by simp
    with A1 have
      ∃f ∈ 𝒮. [id(int)] = [f] ∧ [f] = 0
      using real˙one˙cl˙identity by auto
    then have False using Real˙ZF˙1˙1˙L5 Slopes˙def
      Real˙ZF˙1˙L10 Real˙ZF˙1˙1˙L7 id˙on˙int˙is˙slope
      by auto
  } then show 1≠0 by auto
qed
```

Negative of a real number is a real number. Property of groups.

```
lemma (in real1) Real˙ZF˙1˙1˙L8: assumes a∈ℝ shows (-a) ∈ ℝ
  using assms Real˙ZF˙1˙L2 group0.inverse˙in˙group
  by simp
```

An identity with three real numbers.

```
lemma (in real1) Real˙ZF˙1˙1˙L9: assumes a∈ℝ  b∈ℝ  c∈ℝ
  shows a·(b·c) = a·c·b
  using assms real˙mult˙commutative Real˙ZF˙1˙L3 ring0.Ring˙ZF˙2˙L4
  by simp
```

48.3 The order on reals

In this section we show that the order relation defined by prescribing the set of positive reals as the projection of the set of positive slopes makes the ring of real numbers into an ordered ring. We also collect the facts about ordered groups and rings that we use in the construction.

Positive slopes are slopes and positive reals are real.

```
lemma Real˙ZF˙1˙2˙L1: shows
  PositiveSlopes ⊆ Slopes
  PositiveReals ⊆ RealNumbers
proof -
```

```
  have PositiveSlopes =
    –s ∈ Slopes. s(PositiveIntegers) ∩ PositiveIntegers ∉ Fin(int)″
    using PositiveSlopes˙def by simp
  then show PositiveSlopes ⊆ Slopes by (rule subset˙with˙property)
  then have
    –SlopeEquivalenceRel–s″. s ∈ PositiveSlopes ″ ⊆
    Slopes//SlopeEquivalenceRel
    using EquivClass˙1˙L1A by simp
  then show PositiveReals ⊆ RealNumbers
    using PositiveReals˙def RealNumbers˙def by simp
qed
```

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

```
lemma (in real1) Real˙ZF˙1˙2˙L2:
  shows a ∈ PositiveReals ⟷ (∃f∈𝒮₊. a = [f])
proof
  assume a ∈ PositiveReals
  then have a ∈ –([s]). s ∈ 𝒮₊″ using PositiveReals˙def
    by simp
  then show ∃f∈𝒮₊. a = [f] by auto
next assume ∃f∈𝒮₊. a = [f]
  then have  a ∈ –([s]). s ∈ 𝒮₊″ by auto
  then show a ∈ PositiveReals using PositiveReals˙def
    by simp
qed
```

Let's recall from Int˙ZF˙2.thy that the sum and composition of positive slopes is a positive slope.

```
lemma (in real1) Real˙ZF˙1˙2˙L3:
  assumes f∈𝒮₊  g∈𝒮₊
  shows
  f+g ∈ 𝒮₊
  f∘g ∈ 𝒮₊
  using assms Slopes˙def PositiveSlopes˙def PositiveIntegers˙def
    SlopeOp1˙def int1.sum˙of˙pos˙sls˙is˙pos˙sl
    SlopeOp2˙def int1.comp˙of˙pos˙sls˙is˙pos˙sl
  by auto
```

Bounded integer maps are not positive slopes.

```
lemma (in real1) Real˙ZF˙1˙2˙L5:
  assumes f ∈ BoundedIntMaps
  shows f ∉ 𝒮₊
  using assms BoundedIntMaps˙def Slopes˙def PositiveSlopes˙def
    PositiveIntegers˙def int1.Int˙ZF˙2˙3˙L1B by simp
```

The set of positive reals is closed under addition and multiplication. Zero (the neutral element of addition) is not a positive number.

```
lemma (in real1) Real˙ZF˙1˙2˙L6: shows
```

```
  PositiveReals –is closed under″ RealAddition
  PositiveReals –is closed under″ RealMultiplication
  0 ∉ PositiveReals
proof -
  { fix a fix b
    assume a ∈ PositiveReals and b ∈ PositiveReals
    then obtain f g where
      I: f ∈ 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 assms Slopes˙def SlopeEquivalenceRel˙def BoundedIntMaps˙def
    PositiveSlopes˙def PositiveIntegers˙def
    Real˙ZF˙1˙L10 int1.Int˙ZF˙2˙3˙L8 by simp
```

The next lemma rephrases Int˙ZF˙2˙3˙L10 in the notation used in real1 context.

```
lemma (in real1) Real˙ZF˙1˙2˙L8:
  assumes A1: f ∈ 𝒮  g ∈ 𝒮
  and A2: (f ∈ 𝒮₊) Xor (g ∈ 𝒮₊)
  shows ([f] ∈ PositiveReals) Xor ([g] ∈ PositiveReals)
  using assms PositiveReals˙def SlopeEquivalenceRel˙def Slopes˙def
    SlopeOp1˙def BoundedIntMaps˙def PositiveSlopes˙def PositiveIntegers˙def
    int1.Int˙ZF˙2˙3˙L10 by simp
```

The trichotomy law for the (potential) order on reals: if $a \neq 0$, then either a is positive or $-a$ is potitive.

```
lemma (in real1) Real˙ZF˙1˙2˙L9:
  assumes A1: a∈ℝ and A2: a≠𝟎
  shows (a ∈ PositiveReals) Xor ((-a) ∈ PositiveReals)
proof -
  from A1 obtain f where I: f ∈ 𝒮  a = [f]
    using Real˙ZF˙1˙1˙L3A by auto
  with A2 have ([f] ∈ PositiveReals) Xor ([-f] ∈ PositiveReals)
    using Slopes˙def BoundedIntMaps˙def int1.Int˙ZF˙2˙1˙L12
      Real˙ZF˙1˙2˙L7 Real˙ZF˙1˙2˙L8 by simp
  with I show (a ∈ PositiveReals) Xor ((-a) ∈ PositiveReals)
    using Real˙ZF˙1˙1˙L4A by simp
qed
```

Finally we are ready to prove that real numbers form an ordered ring with no zero divisors.

```
theorem reals˙are˙ord˙ring: shows
  IsAnOrdRing(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
  OrderOnReals –is total on″ RealNumbers
  PositiveSet(RealNumbers,RealAddition,OrderOnReals) = PositiveReals
  HasNoZeroDivs(RealNumbers,RealAddition,RealMultiplication)
proof -
  let R = RealNumbers
  let A = RealAddition
  let M = RealMultiplication
  let P = PositiveReals
  let r = OrderOnReals
  let z = TheNeutralElement(R, A)
  have I:
    ring0(R, A, M)
    M –is commutative on″ R
    P ⊆ R
    P –is closed under″ A
    TheNeutralElement(R, A) ∉ P
    ∀a∈R. a ≠ z ⟶ (a ∈ P) Xor (GroupInv(R, A)(a) ∈ P)
    P –is closed under″ M
    r = OrderFromPosSet(R, A, P)
    using real0.Real˙ZF˙1˙L3 real˙mult˙commutative Real˙ZF˙1˙2˙L1
      real1.Real˙ZF˙1˙2˙L6 real1.Real˙ZF˙1˙2˙L9 OrderOnReals˙def
    by auto
  then show IsAnOrdRing(R, A, M, r)
    by (rule ring0.ring˙ord˙by˙positive˙set)
  from I show r –is total on″ R
    by (rule ring0.ring˙ord˙by˙positive˙set)
  from I show PositiveSet(R,A,r) = P
    by (rule ring0.ring˙ord˙by˙positive˙set)
  from I show HasNoZeroDivs(R,A,M)
    by (rule ring0.ring˙ord˙by˙positive˙set)
```

qed

All theorems proven in the ring1 (about ordered rings), group3 (about ordered groups) and group1 (about groups) contexts are valid as applied to ordered real numbers with addition and (real) order.

```
lemma Real˙ZF˙1˙2˙L10: shows
  ring1(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
  IsAnOrdGroup(RealNumbers,RealAddition,OrderOnReals)
  group3(RealNumbers,RealAddition,OrderOnReals)
  OrderOnReals –is total on″ RealNumbers
proof -
  show ring1(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
    using reals˙are˙ord˙ring OrdRing˙ZF˙1˙L2 by simp
  then show
    IsAnOrdGroup(RealNumbers,RealAddition,OrderOnReals)
    group3(RealNumbers,RealAddition,OrderOnReals)
    OrderOnReals –is total on″ RealNumbers
    using ring1.OrdRing˙ZF˙1˙L4 by auto
qed
```

If $a = b$ or $b - a$ is positive, then a is less or equal b.

```
lemma (in real1) Real˙ZF˙1˙2˙L11: assumes A1: a∈ℝ  b∈ℝ and
  A3: a=b ∨ b-a ∈ PositiveReals
  shows a≤b
  using assms reals˙are˙ord˙ring Real˙ZF˙1˙2˙L10
    group3.OrderedGroup˙ZF˙1˙L30 by simp
```

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

```
lemma (in real1) Real˙ZF˙1˙2˙L12: assumes A1: f ∈ 𝒮  g ∈ 𝒮 and
  A2: f∼g ∨ (g + (-f)) ∈ 𝒮₊
  shows [f] ≤ [g]
proof -
  from A1 A2 have [f] = [g] ∨ [g]-[f] ∈ PositiveReals
    using Real˙ZF˙1˙1˙L5A Real˙ZF˙1˙2˙L2 Real˙ZF˙1˙1˙L4B
    by auto
  with A1 show [f] ≤ [g] using  Real˙ZF˙1˙1˙L3 Real˙ZF˙1˙2˙L11
    by simp
qed
```

Taking negative on both sides reverses the inequality, a case with an inverse on one side. Property of ordered groups.

```
lemma (in real1) Real˙ZF˙1˙2˙L13:
  assumes A1: a∈ℝ and A2: (-a) ≤ b
  shows (-b) ≤ a
  using assms Real˙ZF˙1˙2˙L10 group3.OrderedGroup˙ZF˙1˙L5AG
  by simp
```

Real order is antisymmetric.

```
lemma (in real1) real˙ord˙antisym:
  assumes A1: a≤b  b≤a shows a=b
proof -
  from A1 have
    group3(RealNumbers,RealAddition,OrderOnReals)
    ⟨a,b⟩ ∈ OrderOnReals  ⟨b,a⟩ ∈ OrderOnReals
    using Real˙ZF˙1˙2˙L10 by auto
  then show a=b by (rule group3.group˙order˙antisym)
qed
```

Real order is transitive.

```
lemma (in real1) real˙ord˙transitive: assumes A1: a≤b  b≤c
  shows a≤c
proof -
  from A1 have
    group3(RealNumbers,RealAddition,OrderOnReals)
    ⟨a,b⟩ ∈ OrderOnReals  ⟨b,c⟩ ∈ OrderOnReals
    using Real˙ZF˙1˙2˙L10 by auto
  then have ⟨a,c⟩ ∈ OrderOnReals
    by (rule group3.Group˙order˙transitive)
  then show a≤c by simp
qed
```

We can multiply both sides of an inequality by a nonnegative real number.

```
lemma (in real1) Real˙ZF˙1˙2˙L14:
  assumes a≤b and 0≤c
  shows
  a·c ≤ b·c
  c·a ≤ c·b
  using assms Real˙ZF˙1˙2˙L10 ring1.OrdRing˙ZF˙1˙L9
  by auto
```

A special case of Real˙ZF˙1˙2˙L14: we can multiply an inequality by a real number.

```
lemma (in real1) Real˙ZF˙1˙2˙L14A:
  assumes A1: a≤b and A2: c∈ℝ₊
  shows c·a ≤ c·b
  using assms Real˙ZF˙1˙2˙L10 ring1.OrdRing˙ZF˙1˙L9A
  by simp
```

In the real1 context notation $a \leq b$ implies that a and b are real numbers.

```
lemma (in real1) Real˙ZF˙1˙2˙L15: assumes a≤b shows a∈ℝ  b∈ℝ
  using assms Real˙ZF˙1˙2˙L10 group3.OrderedGroup˙ZF˙1˙L4
  by auto
```

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

```
lemma (in real1) Real˙ZF˙1˙2˙L16: assumes a≤b
  shows 0 ≤ b-a
```

```
  using assms Real˙ZF˙1˙2˙L10 group3.OrderedGroup˙ZF˙1˙L12A
  by simp
```

A sum of nonnegative elements is nonnegative.

```
lemma (in real1) Real˙ZF˙1˙2˙L17: assumes 0≤a 0≤b
  shows 0 ≤ a+b
  using assms Real˙ZF˙1˙2˙L10 group3.OrderedGroup˙ZF˙1˙L12
  by simp
```

We can add sides of two inequalities

```
lemma (in real1) Real˙ZF˙1˙2˙L18: assumes a≤b  c≤d
  shows a+c ≤ b+d
  using assms Real˙ZF˙1˙2˙L10 group3.OrderedGroup˙ZF˙1˙L5B
  by simp
```

The order on real is reflexive.

```
lemma (in real1) real˙ord˙refl: assumes a∈ℝ shows a≤a
  using assms Real˙ZF˙1˙2˙L10 group3.OrderedGroup˙ZF˙1˙L3
  by simp
```

We can add a real number to both sides of an inequality.

```
lemma (in real1) add˙num˙to˙ineq: assumes a≤b and c∈ℝ
  shows a+c ≤ b+c
  using assms Real˙ZF˙1˙2˙L10 IsAnOrdGroup˙def by simp
```

We can put a number on the other side of an inequality, changing its sign.

```
lemma (in real1) Real˙ZF˙1˙2˙L19:
  assumes a∈ℝ  b∈ℝ and c ≤ a+b
  shows c-b ≤ a
  using assms  Real˙ZF˙1˙2˙L10 group3.OrderedGroup˙ZF˙1˙L9C
  by simp
```

What happens when one real number is not greater or equal than another?

```
lemma (in real1) Real˙ZF˙1˙2˙L20: assumes a∈ℝ  b∈ℝ and ¬(a≤b)
  shows b ¡ a
proof -
  from assms have I:
    group3(ℝ,RealAddition,OrderOnReals)
    OrderOnReals –is total on˝ ℝ
    a∈ℝ  b∈ℝ  ¬(⟨a,b⟩ ∈ OrderOnReals)
    using Real˙ZF˙1˙2˙L10 by auto
  then have ⟨b,a⟩ ∈ OrderOnReals
    by (rule group3.OrderedGroup˙ZF˙1˙L8)
  then have b ≤ a by simp
  moreover from I have a≠b by (rule group3.OrderedGroup˙ZF˙1˙L8)
  ultimately show b ¡ a by auto
qed
```

We can put a number on the other side of an inequality, changing its sign, version with a minus.

```
lemma (in real1) Real˙ZF˙1˙2˙L21:
  assumes a∈ℝ  b∈ℝ and c ≤ a-b
  shows c+b ≤ a
  using assms Real˙ZF˙1˙2˙L10 group3.OrderedGroup˙ZF˙1˙L5J
  by simp
```

The order on reals is a relation on reals.

```
lemma (in real1) Real˙ZF˙1˙2˙L22: shows OrderOnReals ⊆ ℝ×ℝ
  using Real˙ZF˙1˙2˙L10 IsAnOrdGroup˙def
  by simp
```

A set that is bounded above in the sense defined by order on reals is a subset of real numbers.

```
lemma (in real1) Real˙ZF˙1˙2˙L23:
  assumes A1: IsBoundedAbove(A,OrderOnReals)
  shows A ⊆ ℝ
  using A1 Real˙ZF˙1˙2˙L22 Order˙ZF˙3˙L1A
  by blast
```

Properties of the maximum of three real numbers.

```
lemma (in real1) Real˙ZF˙1˙2˙L24:
  assumes A1: a∈ℝ  b∈ℝ  c∈ℝ
  shows
  Maximum(OrderOnReals,–a,b,c″) ∈ –a,b,c″
  Maximum(OrderOnReals,–a,b,c″) ∈ ℝ
  a ≤ Maximum(OrderOnReals,–a,b,c″)
  b ≤ Maximum(OrderOnReals,–a,b,c″)
  c ≤ Maximum(OrderOnReals,–a,b,c″)
proof -
  have IsLinOrder(ℝ,OrderOnReals)
    using Real˙ZF˙1˙2˙L10 group3.group˙ord˙total˙is˙lin
    by simp
  with A1 show
    Maximum(OrderOnReals,–a,b,c″) ∈ –a,b,c″
    Maximum(OrderOnReals,–a,b,c″) ∈ ℝ
    a ≤ Maximum(OrderOnReals,–a,b,c″)
    b ≤ Maximum(OrderOnReals,–a,b,c″)
    c ≤ Maximum(OrderOnReals,–a,b,c″)
    using Finite˙ZF˙1˙L2A by auto
qed
```

A form of transitivity for the order on reals.

```
lemma (in real1) real˙strict˙ord˙transit:
  assumes A1: a≤b and A2: b¡c
  shows a¡c
proof -
```

```
  from A1 A2 have I:
    group3(ℝ,RealAddition,OrderOnReals)
    ⟨a,b⟩ ∈ OrderOnReals  ⟨b,c⟩ ∈ OrderOnReals ∧ b≠c
    using Real˙ZF˙1˙2˙L10 by auto
  then have ⟨a,c⟩ ∈ OrderOnReals ∧ a≠c by (rule group3.group˙strict˙ord˙transit)
  then show a¡c by simp
qed
```

We can multiply a right hand side of an inequality between positive real numbers by a number that is greater than one.

```
lemma (in real1) Real˙ZF˙1˙2˙L25:
  assumes b ∈ ℝ₊ and a≤b and 1¡c
  shows a¡b·c
  using assms reals˙are˙ord˙ring Real˙ZF˙1˙2˙L10 ring1.OrdRing˙ZF˙3˙L17
  by simp
```

We can move a real number to the other side of a strict inequality, changing its sign.

```
lemma (in real1) Real˙ZF˙1˙2˙L26:
  assumes a∈ℝ  b∈ℝ and  a-b ¡ c
  shows a ¡ c+b
  using assms Real˙ZF˙1˙2˙L10 group3.OrderedGroup˙ZF˙1˙L12B
  by simp
```

Real order is translation invariant.

```
lemma (in real1) real˙ord˙transl˙inv:
  assumes a≤b and c∈ℝ
  shows c+a ≤ c+b
  using assms Real˙ZF˙1˙2˙L10 IsAnOrdGroup˙def
  by simp
```

It is convenient to have the transitivity of the order on integers in the notation specific to real1 context. This may be confusing for the presentation readers: even though ≤ and ≤ are printed in the same way, they are different symbols in the source. In the real1 context the former denotes inequality between integers, and the latter denotes inequality between real numbers (classes of slopes). The next lemma is about transitivity of the order relation on integers.

```
lemma (in real1) int˙order˙transitive:
  assumes A1: a≤b  b≤c
  shows a≤c
proof -
  from A1 have
    ⟨a,b⟩ ∈ IntegerOrder and ⟨b,c⟩ ∈ IntegerOrder
    by auto
  then have ⟨a,c⟩ ∈ IntegerOrder
    by (rule Int˙ZF˙2˙L5)
```

```
  then show a≤c by simp
qed
```

A property of nonempty subsets of real numbers that don't have a maximum: for any element we can find one that is (strictly) greater.

```
lemma (in real1) Real˙ZF˙1˙2˙L27:
  assumes A⊆ℝ and ¬HasAmaximum(OrderOnReals,A) and x∈A
  shows ∃y∈A. x¦y
  using assms Real˙ZF˙1˙2˙L10 group3.OrderedGroup˙ZF˙2˙L2B
  by simp
```

The next lemma shows what happens when one real number is not greater or equal than another.

```
lemma (in real1) Real˙ZF˙1˙2˙L28:
  assumes a∈ℝ  b∈ℝ and ¬(a≤b)
  shows b¦a
proof -
  from assms have
    group3(ℝ,RealAddition,OrderOnReals)
    OrderOnReals –is total on″ ℝ
    a∈ℝ  b∈ℝ  ⟨a,b⟩ ∉ OrderOnReals
    using Real˙ZF˙1˙2˙L10 by auto
  then have ⟨b,a⟩ ∈ OrderOnReals  ∧ b≠a
    by (rule group3.OrderedGroup˙ZF˙1˙L8)
  then show b¦a by simp
qed
```

If a real number is less than another, then the second one can not be less or equal that the first.

```
lemma (in real1) Real˙ZF˙1˙2˙L29:
  assumes a¦b shows ¬(b≤a)
proof -
  from assms have
    group3(ℝ,RealAddition,OrderOnReals)
    ⟨a,b⟩ ∈ OrderOnReals  a≠b
    using Real˙ZF˙1˙2˙L10 by auto
  then have ⟨b,a⟩ ∉ OrderOnReals
    by (rule group3.OrderedGroup˙ZF˙1˙L8AA)
  then show ¬(b≤a) by simp
qed
```

48.4 Inverting reals

In this section we tackle the issue of existence of (multiplicative) inverses of real numbers and show that real numbers form an ordered field. We also restate here some facts specific to ordered fields that we need for the construction. The actual proofs of most of these facts can be found in Field˙ZF.thy and OrderedField˙ZF.thy

We rewrite the theorem from Int˙ZF˙2.thy that shows that for every positive slope we can find one that is almost equal and has an inverse.

```
lemma (in real1) pos˙slopes˙have˙inv: assumes f ∈ 𝒮₊
  shows ∃g∈𝒮. f∼g ∧ (∃h∈𝒮. g∘h ∼ id(int))
  using assms PositiveSlopes˙def Slopes˙def PositiveIntegers˙def
    int1.pos˙slope˙has˙inv SlopeOp1˙def SlopeOp2˙def
    BoundedIntMaps˙def SlopeEquivalenceRel˙def
  by simp
```

The set of real numbers we are constructing is an ordered field.

```
theorem (in real1) reals˙are˙ord˙field: shows
  IsAnOrdField(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
proof -
  let R = RealNumbers
  let A = RealAddition
  let M = RealMultiplication
  let r = OrderOnReals
  have ring1(R,A,M,r) and 0 ≠ 1
    using reals˙are˙ord˙ring OrdRing˙ZF˙1˙L2 real˙zero˙not˙one
    by auto
  moreover have M –is commutative on″ R
    using real˙mult˙commutative by simp
  moreover have
    ∀a∈PositiveSet(R,A,r). ∃b∈R. a·b = 1
  proof
    fix a assume a ∈ PositiveSet(R,A,r)
    then obtain f where I: f∈𝒮₊ and II: a = [f]
      using reals˙are˙ord˙ring Real˙ZF˙1˙2˙L2
      by auto
    then have ∃g∈𝒮. f∼g ∧ (∃h∈𝒮. g∘h ∼ id(int))
      using pos˙slopes˙have˙inv by simp
    then obtain g where
      III: g∈𝒮 and IV: f∼g and V: ∃h∈𝒮. g∘h ∼ id(int)
      by auto
    from V obtain h where VII: h∈𝒮 and VIII: g∘h ∼ id(int)
      by auto
    from I III IV have [f] = [g]
      using Real˙ZF˙1˙2˙L1 Slopes˙def Real˙ZF˙1˙1˙L5
      by auto
    with II III VII VIII have a·[h] = 1
      using Real˙ZF˙1˙1˙L4  Real˙ZF˙1˙1˙L5A real˙one˙cl˙identity
      by simp
    with VII show ∃b∈R. a·b = 1 using Real˙ZF˙1˙1˙L3
      by auto
  qed
  ultimately show thesis using ring1.OrdField˙ZF˙1˙L4
    by simp
qed
```

Reals form a field.

```
lemma reals˙are˙field:
  shows IsAfield(RealNumbers,RealAddition,RealMultiplication)
  using real1.reals˙are˙ord˙field OrdField˙ZF˙1˙L1A
  by simp
```

Theorem proven in field0 and field1 contexts are valid as applied to real numbers.

```
lemma field˙cntxts˙ok: shows
  field0(RealNumbers,RealAddition,RealMultiplication)
  field1(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
  using reals˙are˙field real1.reals˙are˙ord˙field
     field˙field0 OrdField˙ZF˙1˙L2 by auto
```

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

```
lemma (in real1) Real˙ZF˙1˙3˙L1: assumes a ∈ ℝ+
  shows a⁻¹ ∈ ℝ+   a⁻¹ ∈ ℝ
  using assms field˙cntxts˙ok field1.OrdField˙ZF˙1˙L8 PositiveSet˙def
  by auto
```

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

```
lemma (in real1) Real˙ZF˙1˙3˙L2:
  assumes a ∈ ℝ+ and a⁻¹ ¡ b
  shows 1 ¡ b·a
  using assms field˙cntxts˙ok field1.OrdField˙ZF˙2˙L2
  by simp
```

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

```
lemma (in real1) Real˙ZF˙1˙3˙L3: assumes a¡b
  shows (b-a)⁻¹ ∈ ℝ+
  using assms field˙cntxts˙ok field1.OrdField˙ZF˙1˙L9
  by simp
```

We can put a positive factor on the other side of a strict inequality, changing it to its inverse.

```
lemma (in real1) Real˙ZF˙1˙3˙L4:
  assumes A1: a∈ℝ  b∈ℝ+ and A2: a·b ¡ c
  shows a ¡ c·b⁻¹
  using assms field˙cntxts˙ok field1.OrdField˙ZF˙2˙L6
  by simp
```

We can put a positive factor on the other side of a strict inequality, changing it to its inverse, version with the product initially on the right hand side.

```
lemma (in real1) Real˙ZF˙1˙3˙L4A:
  assumes A1: b∈ℝ  c∈ℝ+ and A2: a ¡ b·c
  shows a·c⁻¹ ¡ b
```

using assms field˙cntxts˙ok field1.OrdField˙ZF˙2˙L6A
by simp

We can put a positive factor on the other side of an inequality, changing it to its inverse, version with the product initially on the right hand side.

lemma (in real1) Real˙ZF˙1˙3˙L4B:
assumes A1: $b \in \mathbb{R}$ $c \in \mathbb{R}_+$ and A2: $a \leq b \cdot c$
shows $a \cdot c^{-1} \leq b$
using assms field˙cntxts˙ok field1.OrdField˙ZF˙2˙L5A
by simp

We can put a positive factor on the other side of an inequality, changing it to its inverse, version with the product initially on the left hand side.

lemma (in real1) Real˙ZF˙1˙3˙L4C:
assumes A1: $a \in \mathbb{R}$ $b \in \mathbb{R}_+$ and A2: $a \cdot b \leq c$
shows $a \leq c \cdot b^{-1}$
using assms field˙cntxts˙ok field1.OrdField˙ZF˙2˙L5
by simp

A technical lemma about solving a strict inequality with three real numbers and inverse of a difference.

lemma (in real1) Real˙ZF˙1˙3˙L5:
assumes a¡b and $(b\text{-}a)^{-1}$ ¡ c
shows $\mathbf{1} + a \cdot c$ ¡ $b \cdot c$
using assms field˙cntxts˙ok field1.OrdField˙ZF˙2˙L9
by simp

We can multiply an inequality by the inverse of a positive number.

lemma (in real1) Real˙ZF˙1˙3˙L6:
assumes $a \leq b$ and $c \in \mathbb{R}_+$ shows $a \cdot c^{-1} \leq b \cdot c^{-1}$
using assms field˙cntxts˙ok field1.OrdField˙ZF˙2˙L3
by simp

We can multiply a strict inequality by a positive number or its inverse.

lemma (in real1) Real˙ZF˙1˙3˙L7:
assumes a¡b and $c \in \mathbb{R}_+$ shows
$a \cdot c$ ¡ $b \cdot c$
$c \cdot a$ ¡ $c \cdot b$
$a \cdot c^{-1}$ ¡ $b \cdot c^{-1}$
using assms field˙cntxts˙ok field1.OrdField˙ZF˙2˙L4
by auto

An identity with three real numbers, inverse and cancelling.

lemma (in real1) Real˙ZF˙1˙3˙L8: assumes$a \in \mathbb{R}$ $b \in \mathbb{R}$ $b \neq \mathbf{0}$ $c \in \mathbb{R}$
shows $a \cdot b \cdot (c \cdot b^{-1}) = a \cdot c$
using assms field˙cntxts˙ok field0.Field˙ZF˙2˙L6
by simp

48.5 Completeness

This goal of this section is to show that the order on real numbers is complete, that is every subset of reals that is bounded above has a smallest upper bound.

If m is an integer, then m^R is a real number. Recall that in real1 context m^R denotes the class of the slope $n \mapsto m \cdot n$.

```
lemma (in real1) real˙int˙is˙real: assumes m ∈ int
  shows m^R ∈ ℝ
  using assms int1.Int˙ZF˙2˙5˙L1 Real˙ZF˙1˙1˙L3 by simp
```

The negative of the real embedding of an integer is the embedding of the negative of the integer.

```
lemma (in real1) Real˙ZF˙1˙4˙L1: assumes m ∈ int
  shows (-m)^R = -(m^R)
  using assms int1.Int˙ZF˙2˙5˙L3 int1.Int˙ZF˙2˙5˙L1 Real˙ZF˙1˙1˙L4A
  by simp
```

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

```
lemma (in real1) Real˙ZF˙1˙4˙L1A: assumes m ∈ int  k ∈ int
  shows m^R + k^R = ((m+k)^R)
  using assms int1.Int˙ZF˙2˙5˙L1 SlopeOp1˙def int1.Int˙ZF˙2˙5˙L3A
    Real˙ZF˙1˙1˙L4 by simp
```

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

```
lemma (in real1) Real˙ZF˙1˙4˙L1B: assumes A1: m ∈ int  k ∈ int
  shows m^R - k^R = (m-k)^R
proof -
  from A1 have (-k) ∈ int using int0.Int˙ZF˙1˙1˙L4
    by simp
  with A1 have (m-k)^R = m^R + (-k)^R
    using Real˙ZF˙1˙4˙L1A by simp
  with A1 show m^R - k^R = (m-k)^R
    using Real˙ZF˙1˙4˙L1 by simp
qed
```

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

```
lemma (in real1) Real˙ZF˙1˙4˙L1C: assumes m ∈ int  k ∈ int
  shows m^R · k^R = (m·k)^R
  using assms int1.Int˙ZF˙2˙5˙L1 SlopeOp2˙def int1.Int˙ZF˙2˙5˙L3B
    Real˙ZF˙1˙1˙L4 by simp
```

For any real numbers there is an integer whose real version is greater or equal.

```
lemma (in real1) Real˙ZF˙1˙4˙L2: assumes A1: a∈ℝ
  shows ∃m∈int. a ≤ m^R
```

```
proof -
  from A1 obtain f where I: f∈𝒮 and II: a = [f]
    using Real˙ZF˙1˙1˙L3A by auto
  then have ∃m∈int. ∃g∈𝒮.
    −⟨n,m·n⟩ . n ∈ int″ ∼ g ∧ (f∼g ∨ (g + (-f)) ∈ 𝒮₊)
    using int1.Int˙ZF˙2˙5˙L2 Slopes˙def SlopeOp1˙def
      BoundedIntMaps˙def SlopeEquivalenceRel˙def
      PositiveIntegers˙def PositiveSlopes˙def
    by simp
  then obtain m g where III: m∈int and IV: g∈𝒮 and
   −⟨n,m·n⟩ . n ∈ int″ ∼ g ∧ (f∼g ∨ (g + (-f)) ∈ 𝒮₊)
    by auto
  then have m^R = [g] and f ∼ g ∨ (g + (-f)) ∈ 𝒮₊
    using Real˙ZF˙1˙1˙L5A by auto
  with I II IV have a ≤ m^R using Real˙ZF˙1˙2˙L12
    by simp
  with III show ∃m∈int. a ≤ m^R by auto
qed
```

For any real numbers there is an integer whose real version (embedding) is less or equal.

```
lemma (in real1) Real˙ZF˙1˙4˙L3: assumes A1: a∈ℝ
  shows −m ∈ int. m^R ≤ a″ ≠ 0
proof -
  from A1 have (-a) ∈ ℝ using Real˙ZF˙1˙1˙L8
    by simp
  then obtain m where I: m∈int and II: (-a) ≤ m^R
    using Real˙ZF˙1˙4˙L2 by auto
  let k = GroupInv(int,IntegerAddition)(m)
  from A1 I II have k ∈ int and k^R ≤ a
    using Real˙ZF˙1˙2˙L13 Real˙ZF˙1˙4˙L1 int0.Int˙ZF˙1˙1˙L4
    by auto
  then show thesis by auto
qed
```

Embeddings of two integers are equal only if the integers are equal.

```
lemma (in real1) Real˙ZF˙1˙4˙L4:
  assumes A1: m ∈ int  k ∈ int and A2: m^R = k^R
  shows m=k
proof -
  let r = −⟨n, IntegerMultiplication  ⟨m, n⟩⟩ . n ∈ int″
  let s = −⟨n, IntegerMultiplication  ⟨k, n⟩⟩ . n ∈ int″
  from A1 A2 have r ∼ s
    using int1.Int˙ZF˙2˙5˙L1 AlmostHoms˙def Real˙ZF˙1˙1˙L5
    by simp
  with A1 have
    m ∈ int  k ∈ int
    ⟨r,s⟩ ∈ QuotientGroupRel(AlmostHoms(int, IntegerAddition),
    AlHomOp1(int, IntegerAddition),FinRangeFunctions(int, int))
```

```
    using SlopeEquivalenceRel˙def Slopes˙def SlopeOp1˙def
      BoundedIntMaps˙def by auto
  then show m=k by (rule int1.Int˙ZF˙2˙5˙L6)
qed
```

The embedding of integers preserves the order.

```
lemma (in real1) Real˙ZF˙1˙4˙L5: assumes A1: m≤k
  shows 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 is smaller than $M^R y$. This is Lemma 14 ii) in [2].

```
lemma (in real1) Arthan˙Lemma14ii: assumes A1: x∈ℝ  y ∈ ℝ+
  shows ∃M∈ℤ+. x ¡ M^R·y
proof -
  from A1 have
    ∃C∈ℤ+. x ¡ C^R and ∃D∈ℤ+. y^-1 ¡ D^R
    using Real˙ZF˙1˙3˙L1 Arthan˙Lemma14i by auto
  then obtain C D where
    I: C∈ℤ+ and II: x ¡ C^R and
    III: D∈ℤ+ and IV: y^-1 ¡ D^R
    by auto
  let M = C·D
  from I III have
    T: M ∈ ℤ+  C^R ∈ ℝ  D^R ∈ ℝ
    using int0.pos˙int˙closed˙mul˙unfold PositiveSet˙def real˙int˙is˙real
    by auto
  with A1 I III have C^R·(D^R·y) = M^R·y
    using PositiveSet˙def Real˙ZF˙1˙L6A Real˙ZF˙1˙4˙L1C
    by simp
  moreover from A1 I II IV have
    x ¡ C^R·(D^R·y)
    using int˙pos˙is˙real˙pos Real˙ZF˙1˙3˙L2 Real˙ZF˙1˙2˙L25
    by auto
  ultimately have x ¡ M^R·y
    by auto
  with T show thesis by auto
qed
```

Taking the floor function preserves the order.

```
lemma (in real1) Real˙ZF˙1˙4˙L9: assumes A1: a≤b
  shows ⌊a⌋ ≤ ⌊b⌋
proof -
  from A1 have T: a∈ℝ using Real˙ZF˙1˙2˙L15
    by simp
  with A1 have ⌊a⌋^R ≤ a and a≤b
    using Real˙ZF˙1˙4˙L7 by auto
  then have ⌊a⌋^R ≤ b by (rule real˙ord˙transitive)
  moreover from T have ⌊a⌋ ∈ int using Real˙ZF˙1˙4˙L7
    by simp
```

```
 ultimately show ⌊a⌋ ≤ ⌊b⌋ using Real˙ZF˙1˙4˙L8
   by simp
qed
```

If S is bounded above and p is a positive 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 $\forall$x$\in$A. $\langle$x,Maximum(IntegerOrder,A)$\rangle \in$ IntegerOrder
by (rule int0.int˙bounded˙above˙has˙max)
with A2 show $\lfloor p^R \cdot x \rfloor \leq \Gamma(S,p)$ 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$\neq$0 and
A2: g = $-\langle p,\Gamma(S,p)\rangle$. p$\in\mathbb{Z}_+$ ″
shows
g : $\mathbb{Z}_+ \rightarrow$int
$\forall$n$\in\mathbb{Z}_+$. g(n) = Γ(S,n)
proof -
from A1 have $\forall$n$\in\mathbb{Z}_+$. Γ(S,n) $\in$ int using Real˙ZF˙1˙4˙L10
by simp
with A2 show I: g : $\mathbb{Z}_+ \rightarrow$int using ZF˙fun˙from˙total by simp
{ fix n assume n$\in\mathbb{Z}_+$
with A2 I have g(n) = Γ(S,n) using ZF˙fun˙from˙tot˙val
by simp
} then show $\forall$n$\in\mathbb{Z}_+$. 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 $\in$ int
shows $\lfloor m^R \rfloor$ = m
proof -
let A = $-$n $\in$ int. $n^R \leq m^R$ ″
have antisym(IntegerOrder) using int0.Int˙ZF˙2˙L4
by simp
moreover from A1 have m $\in$ A
using real˙int˙is˙real real˙ord˙refl by auto
moreover from A1 have $\forall$n $\in$ A. $\langle$n,m$\rangle \in$ IntegerOrder
using Real˙ZF˙1˙4˙L6 by auto
ultimately show $\lfloor m^R \rfloor$ = 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
$\lfloor \mathbf{0} \rfloor = \mathbf{0}_Z \quad \lfloor \mathbf{1} \rfloor = \mathbf{1}_Z$
proof -
have $\lfloor (\mathbf{0}_Z)^R \rfloor = \mathbf{0}_Z$ and $\lfloor (\mathbf{1}_Z)^R \rfloor = \mathbf{1}_Z$
using int0.int˙zero˙one˙are˙int Real˙ZF˙1˙4˙L14
by auto
then show $\lfloor \mathbf{0} \rfloor = \mathbf{0}_Z$ and $\lfloor \mathbf{1} \rfloor = \mathbf{1}_Z$
using int˙0˙1˙are˙real˙zero˙one
by auto
qed

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

```
lemma (in real1) floor˙2˙is˙two: shows ⌊2⌋ = 2_Z
proof -
  have ⌊(2_Z)^R⌋ = 2_Z
    using int0.int˙two˙three˙are˙int Real˙ZF˙1˙4˙L14
    by simp
  then show ⌊2⌋ = 2_Z using int˙two˙is˙real˙two
    by simp
qed
```

Floor of a product of embeddings of integers is equal to the product of integers.

```
lemma (in real1) Real˙ZF˙1˙4˙L14A: assumes A1: m ∈ int  k ∈ int
  shows  ⌊m^R·k^R⌋ = m·k
proof -
  from A1 have T: m·k ∈ int
    using int0.Int˙ZF˙1˙1˙L5 by simp
  from A1 have ⌊m^R·k^R⌋ = ⌊(m·k)^R⌋ using Real˙ZF˙1˙4˙L1C
    by simp
  with T show ⌊m^R·k^R⌋ = m·k using Real˙ZF˙1˙4˙L14
    by simp
qed
```

Floor of the sum of a number and the embedding of an integer is the floor of the number plus the integer.

```
lemma (in real1) Real˙ZF˙1˙4˙L15: assumes A1: x∈ℝ and A2: p ∈ int
  shows ⌊x + p^R⌋ = ⌊x⌋ + p
proof -
  let A = –n ∈ int. n^R ≤ x + p^R"
  have antisym(IntegerOrder) using int0.Int˙ZF˙2˙L4
    by simp
  moreover have ⌊x⌋ + p ∈ A
  proof -
    from A1 A2 have ⌊x⌋^R ≤ x and p^R ∈ ℝ
      using Real˙ZF˙1˙4˙L7 real˙int˙is˙real by auto
    then have ⌊x⌋^R + p^R ≤ x + p^R
      using add˙num˙to˙ineq by simp
    moreover from A1 A2 have (⌊x⌋ + p)^R = ⌊x⌋^R + p^R
      using Real˙ZF˙1˙4˙L7 Real˙ZF˙1˙4˙L1A by simp
    ultimately have (⌊x⌋ + p)^R ≤ x + p^R
      by simp
    moreover from A1 A2 have ⌊x⌋ + p ∈ int
      using Real˙ZF˙1˙4˙L7 int0.Int˙ZF˙1˙1˙L5 by simp
    ultimately show ⌊x⌋ + p ∈ A by auto
  qed
  moreover have ∀n∈A. n ≤ ⌊x⌋ + p
  proof
    fix n assume n∈A
```

```
    then have I: n ∈ int and n^R ≤ x + p^R
      by auto
    with A1 A2 have n^R - p^R ≤ x
      using real˙int˙is˙real Real˙ZF˙1˙2˙L19
      by simp
    with A2 I have ⌊(n-p)^R⌋ ≤ ⌊x⌋
      using Real˙ZF˙1˙4˙L1B Real˙ZF˙1˙4˙L9
      by simp
    moreover
    from A2 I have n-p ∈ int
      using int0.Int˙ZF˙1˙1˙L5 by simp
    then have ⌊(n-p)^R⌋ = n-p
      using Real˙ZF˙1˙4˙L14 by simp
    ultimately have n-p ≤ ⌊x⌋
      by simp
    with A2 I show n ≤ ⌊x⌋ + p
      using int0.Int˙ZF˙2˙L9C by simp
  qed
  ultimately show ⌊x + p^R⌋ = ⌊x⌋ + p
    using Order˙ZF˙4˙L14 by auto
qed
```

Floor of the difference of a number and the embedding of an integer is the floor of the number minus the integer.

```
lemma (in real1) Real˙ZF˙1˙4˙L16: assumes A1: x∈ℝ and A2: p ∈ int
  shows ⌊x - p^R⌋ = ⌊x⌋ - p
proof -
  from A2 have ⌊x - p^R⌋ = ⌊x + (-p)^R⌋
     using Real˙ZF˙1˙4˙L1 by simp
  with A1 A2 show ⌊x - p^R⌋ = ⌊x⌋ - p
    using int0.Int˙ZF˙1˙1˙L4 Real˙ZF˙1˙4˙L15 by simp
qed
```

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

```
lemma (in real1) Real˙ZF˙1˙4˙L17: assumes m ∈ int  n ∈ int
  shows ⌊(m^R) + n^R⌋ = m + n
  using assms real˙int˙is˙real Real˙ZF˙1˙4˙L15 Real˙ZF˙1˙4˙L14
  by simp
```

A lemma about adding one to floor.

```
lemma (in real1) Real˙ZF˙1˙4˙L17A: assumes A1: a∈ℝ
  shows 1 + ⌊a⌋^R = (1_Z + ⌊a⌋)^R
proof -
  have 1 + ⌊a⌋^R = 1_Z^R + ⌊a⌋^R
    using int˙0˙1˙are˙real˙zero˙one by simp
  with A1 show 1 + ⌊a⌋^R = (1_Z + ⌊a⌋)^R
    using int0.int˙zero˙one˙are˙int Real˙ZF˙1˙4˙L7 Real˙ZF˙1˙4˙L1A
    by simp
qed
```

The difference between the a number and the embedding of its floor is (strictly) less than one.

```
lemma (in real1) Real˙ZF˙1˙4˙L17B: assumes A1: a∈ℝ
  shows
  a - ⌊a⌋^R ¡ 1
  a ¡ (1_Z + ⌊a⌋)^R
proof -
  from A1 have T1: ⌊a⌋ ∈ int  ⌊a⌋^R ∈ ℝ and
    T2: 1 ∈ ℝ  a -  ⌊a⌋^R ∈ ℝ
    using Real˙ZF˙1˙4˙L7 real˙int˙is˙real Real˙ZF˙1˙L6 Real˙ZF˙1˙L4
    by auto
  { assume 1 ≤ a -  ⌊a⌋^R
    with A1 T1 have ⌊1_Z^R + ⌊a⌋^R⌋ ≤ ⌊a⌋
      using Real˙ZF˙1˙2˙L21 Real˙ZF˙1˙4˙L9 int˙0˙1˙are˙real˙zero˙one
      by simp
    with T1 have False
      using int0.int˙zero˙one˙are˙int Real˙ZF˙1˙4˙L17
      int0.Int˙ZF˙1˙2˙L3AA by simp
  } then have I: ¬(1 ≤ a - ⌊a⌋^R) by auto
  with T2 show II: a - ⌊a⌋^R ¡ 1
    by (rule Real˙ZF˙1˙2˙L20)
   with A1 T1 I II have
    a ¡ 1 + ⌊a⌋^R
    using Real˙ZF˙1˙2˙L26 by simp
  with A1 show a ¡ (1_Z + ⌊a⌋)^R
    using Real˙ZF˙1˙4˙L17A by simp
qed
```

The next lemma corresponds to Lemma 14 iii) in [2]. It says that we can find a rational number between any two different real numbers.

```
lemma (in real1) Arthan˙Lemma14iii: assumes A1: x¡y
  shows ∃M∈int. ∃N∈ℤ_+.  x·N^R ¡ M^R ∧ M^R ¡ y·N^R
proof -
  from A1 have (y-x)^-1 ∈ ℝ_+ using Real˙ZF˙1˙3˙L3
    by simp
  then have
    ∃N∈ℤ_+. (y-x)^-1 ¡ N^R
    using Arthan˙Lemma14i PositiveSet˙def by simp
  then obtain N where I: N∈ℤ_+ and II: (y-x)^-1 ¡ N^R
    by auto
  let M = 1_Z + ⌊x·N^R⌋
  from A1 I have
    T1: x∈ℝ  N^R ∈ ℝ  N^R ∈ ℝ_+  x·N^R ∈ ℝ
    using Real˙ZF˙1˙2˙L15 PositiveSet˙def real˙int˙is˙real
      Real˙ZF˙1˙L6 int˙pos˙is˙real˙pos by auto
  then have T2: M ∈ int using
    int0.int˙zero˙one˙are˙int Real˙ZF˙1˙4˙L7 int0.Int˙ZF˙1˙1˙L5
    by simp
  from T1 have III: x·N^R ¡ M^R
```

using Real˙ZF˙1˙4˙L17B by simp
from T1 have $(\mathbf{1} + \lfloor x{\cdot}N^R \rfloor^R) \leq (\mathbf{1} + x{\cdot}N^R)$
using Real˙ZF˙1˙4˙L7 Real˙ZF˙1˙L4 real˙ord˙transl˙inv
by simp
with T1 have $M^R \leq (\mathbf{1} + x{\cdot}N^R)$
using Real˙ZF˙1˙4˙L17A by simp
moreover from A1 II have $(\mathbf{1} + x{\cdot}N^R) \text{ ¡ } y{\cdot}N^R$
using Real˙ZF˙1˙3˙L5 by simp
ultimately have $M^R \text{ ¡ } y{\cdot}N^R$
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: $x \in \mathbb{R}$ $y \in \mathbb{R}$
shows
$\text{abs}(\lfloor x+y \rfloor - \lfloor x \rfloor - \lfloor y \rfloor) \leq \mathbf{2}_Z$
proof -
from A1 have T:
$\lfloor x \rfloor^R \in \mathbb{R}$ $\lfloor y \rfloor^R \in \mathbb{R}$
$x+y - (\lfloor x \rfloor^R) \in \mathbb{R}$
using Real˙ZF˙1˙4˙L7 real˙int˙is˙real Real˙ZF˙1˙L6
by auto
from A1 have
$\mathbf{0} \leq x - (\lfloor x \rfloor^R) + (y - (\lfloor y \rfloor^R))$
$x - (\lfloor x \rfloor^R) + (y - (\lfloor y \rfloor^R)) \leq \mathbf{2}$
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
$x - (\lfloor x \rfloor^R) + (y - (\lfloor y \rfloor^R)) = x+y - (\lfloor x \rfloor^R) - (\lfloor y \rfloor^R)$
using Real˙ZF˙1˙L7A by simp
ultimately have
$\mathbf{0} \leq x+y - (\lfloor x \rfloor^R) - (\lfloor y \rfloor^R)$
$x+y - (\lfloor x \rfloor^R) - (\lfloor y \rfloor^R) \leq \mathbf{2}$
by auto
then have
$\lfloor \mathbf{0} \rfloor \leq \lfloor x+y - (\lfloor x \rfloor^R) - (\lfloor y \rfloor^R) \rfloor$
$\lfloor x+y - (\lfloor x \rfloor^R) - (\lfloor y \rfloor^R) \rfloor \leq \lfloor \mathbf{2} \rfloor$
using Real˙ZF˙1˙4˙L9 by auto
then have
$\mathbf{0}_Z \leq \lfloor x+y - (\lfloor x \rfloor^R) - (\lfloor y \rfloor^R) \rfloor$
$\lfloor x+y - (\lfloor x \rfloor^R) - (\lfloor y \rfloor^R) \rfloor \leq \mathbf{2}_Z$
using floor˙01˙is˙zero˙one floor˙2˙is˙two by auto
moreover from A1 have
$\lfloor x+y - (\lfloor x \rfloor^R) - (\lfloor y \rfloor^R) \rfloor = \lfloor x+y \rfloor - \lfloor x \rfloor - \lfloor y \rfloor$
using Real˙ZF˙1˙L6 Real˙ZF˙1˙4˙L7 real˙int˙is˙real Real˙ZF˙1˙4˙L16
by simp
ultimately have
$\mathbf{0}_Z \leq \lfloor x+y \rfloor - \lfloor x \rfloor - \lfloor y \rfloor$

```
    ⌊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)) ≤ $\mathbf{2}_Z$
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:$\mathbb{Z}_+$→int ∀a∈$\mathbb{Z}_+$. ∀b∈$\mathbb{Z}_+$. abs(δ(f,a,b)) ≤ L
shows OddExtension(int,IntegerAddition,IntegerOrder,f) ∈ $\mathcal{S}$
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)) ≤ 2Z
  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)) ≤  2Z
 by (rule Real˙ZF˙1˙4˙L21)
      ultimately have abs(δ(g,m,n)) ≤ 2Z
 by simp
    } then show ∀m∈ℤ+. ∀n∈ℤ+. abs(δ(g,m,n)) ≤ 2Z
      by simp
  qed
  ultimately show thesis by (rule Real˙ZF˙1˙4˙L21A)
qed
```

A technical lemma used in the proof that all elements of S are less or equal than the candidate for supremum of S.

```
lemma (in real1) Real˙ZF˙1˙4˙L23:
  assumes A1: f ∈ 𝒮 and A2: N ∈ int  M ∈ int and
  A3: ∀n∈ℤ+. M·n ≤ f(N·n)
  shows M^R ≤ [f]·(N^R)
proof -
  let M_S = −⟨n, M·n⟩ . n ∈ int″
  let N_S = −⟨n, N·n⟩ . n ∈ int″
  from A1 A2 have T: M_S ∈ 𝒮  N_S ∈ 𝒮  f∘N_S ∈ 𝒮
    using int1.Int˙ZF˙2˙5˙L1 int1.Int˙ZF˙2˙1˙L11 SlopeOp2˙def
    by auto
  moreover from A1 A2 A3 have M_S ∼ f∘N_S ∨ f∘N_S + (-M_S) ∈ 𝒮+
    using int1.Int˙ZF˙2˙5˙L8 SlopeOp2˙def SlopeOp1˙def Slopes˙def
      BoundedIntMaps˙def SlopeEquivalenceRel˙def PositiveIntegers˙def
      PositiveSlopes˙def by simp
  ultimately have [M_S] ≤ [f∘N_S] using Real˙ZF˙1˙2˙L12
    by simp
  with A1 T show M^R ≤ [f]·(N^R) using Real˙ZF˙1˙1˙L4
    by simp
qed
```

A technical lemma aimed used in the proof the candidate for supremum of S is less or equal than any upper bound for S.

```
lemma (in real1) Real˙ZF˙1˙4˙L23A:
  assumes A1: f ∈ 𝒮 and A2: N ∈ int  M ∈ int and
  A3: ∀n∈ℤ₊. f(N·n) ≤  M·n
  shows [f]·(N^R) ≤ M^R
proof -
  let M_S = −⟨n, M·n⟩ . n ∈ int″
  let N_S = −⟨n, N·n⟩ . n ∈ int″
  from A1 A2 have T: M_S ∈ 𝒮  N_S ∈ 𝒮  f∘N_S ∈ 𝒮
    using int1.Int˙ZF˙2˙5˙L1 int1.Int˙ZF˙2˙1˙L11 SlopeOp2˙def
    by auto
  moreover from A1 A2 A3 have
    f∘N_S ∼ M_S ∨  M_S + (-(f∘N_S)) ∈ 𝒮₊
    using int1.Int˙ZF˙2˙5˙L9 SlopeOp2˙def SlopeOp1˙def Slopes˙def
      BoundedIntMaps˙def SlopeEquivalenceRel˙def PositiveIntegers˙def
      PositiveSlopes˙def by simp
  ultimately have [f∘N_S] ≤ [M_S] using Real˙ZF˙1˙2˙L12
    by simp
  with A1 T show  [f]·(N^R)≤ M^R using Real˙ZF˙1˙1˙L4
    by simp
qed
```

The essential condition to claim that the candidate for supremum of S is greater or equal than all elements of S.

```
lemma (in real1) Real˙ZF˙1˙4˙L24:
  assumes A1: IsBoundedAbove(S,OrderOnReals) and
  A2: x¡y  y∈S  and
  A4: N ∈ ℤ₊  M ∈ int and
  A5: M^R ¡ y·N^R and A6: p ∈ ℤ₊
  shows p·M ≤ Γ(S,p·N)
proof -
  from A2 A4 A6 have T1:
    N^R ∈ ℝ₊   y∈ℝ   p^R ∈ ℝ₊
    p·N ∈ ℤ₊   (p·N)^R ∈ ℝ₊
    using int˙pos˙is˙real˙pos Real˙ZF˙1˙2˙L15
    int0.pos˙int˙closed˙mul˙unfold by auto
  with A4 A6 have T2:
    p ∈ int   p^R ∈ ℝ   N^R ∈ ℝ  N^R ≠ 0   M^R ∈ ℝ
    using real˙int˙is˙real PositiveSet˙def by auto
  from T1 A5 have ⌊(p·N)^R·(M^R·(N^R)^−1)⌋ ≤ ⌊(p·N)^R·y⌋
    using Real˙ZF˙1˙3˙L4A Real˙ZF˙1˙3˙L7 Real˙ZF˙1˙4˙L9
    by simp
  moreover from A1 A2 T1 have ⌊(p·N)^R·y⌋ ≤ Γ(S,p·N)
    using Real˙ZF˙1˙4˙L11 by simp
  ultimately have I: ⌊(p·N)^R·(M^R·(N^R)^−1)⌋ ≤ Γ(S,p·N)
    by (rule int˙order˙transitive)
  from A4 A6 have (p·N)^R·(M^R·(N^R)^−1) = p^R·N^R·(M^R·(N^R)^−1)
    using PositiveSet˙def Real˙ZF˙1˙4˙L1C by simp
  with A4 T2 have ⌊(p·N)^R·(M^R·(N^R)^−1)⌋ = p·M
    using Real˙ZF˙1˙3˙L8 Real˙ZF˙1˙4˙L14A by simp
```

```
  with I show p·M ≤ Γ(S,p·N) by simp
qed
```

An obvious fact about odd extension of a function $p \mapsto \Gamma(s,p)$ that is used a couple of times in proofs.

```
lemma (in real1) Real˙ZF˙1˙4˙L24A:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0 and A2: p ∈ ℤ+
  and A3:
  h = OddExtension(int,IntegerAddition,IntegerOrder,−⟨p,Γ(S,p)⟩. p∈ℤ+˝)
  shows h(p) = Γ(S,p)
proof -
  let g = −⟨p,Γ(S,p)⟩. p∈ℤ+˝
  from A1 have I: g : ℤ+→int using  Real˙ZF˙1˙4˙L12
    by blast
  with A2 A3 show h(p) = Γ(S,p)
    using int0.Int˙ZF˙1˙5˙L11 ZF˙fun˙from˙tot˙val
    by simp
qed
```

The candidate for the supremum of S is not smaller than any element of S.

```
lemma (in real1) Real˙ZF˙1˙4˙L25:
  assumes A1: IsBoundedAbove(S,OrderOnReals) and
  A2: ¬HasAmaximum(OrderOnReals,S) and
  A3: x∈S and A4:
  h = OddExtension(int,IntegerAddition,IntegerOrder,−⟨p,Γ(S,p)⟩. p∈ℤ+˝)
  shows x ≤ [h]
proof -
  from A1 A2 A3 have
    S ⊆ ℝ  ¬HasAmaximum(OrderOnReals,S)  x∈S
    using Real˙ZF˙1˙2˙L23 by auto
  then have ∃y∈S. x¡y by (rule Real˙ZF˙1˙2˙L27)
  then obtain y where I: y∈S and  II: x¡y
    by auto
  from II have
    ∃M∈int. ∃N∈ℤ+.  x·N^R ¡ M^R ∧ M^R ¡ y·N^R
    using Arthan˙Lemma14iii by simp
  then obtain M N where III: M ∈ int  N∈ℤ+ and
    IV: x·N^R ¡ M^R  M^R ¡ y·N^R
    by auto
  from II III IV have V: x ≤ M^R·(N^R)^−1
    using int˙pos˙is˙real˙pos Real˙ZF˙1˙2˙L15 Real˙ZF˙1˙3˙L4
    by auto
  from A3 have VI: S≠0 by auto
  with A1 A4 have T1: h ∈ 𝒮 using Real˙ZF˙1˙4˙L22
    by simp
  moreover from III have N ∈ int  M ∈ int
    using PositiveSet˙def by auto
  moreover have ∀n∈ℤ+. M·n ≤ h(N·n)
  proof
```

```
    let g = −⟨p,Γ(S,p)⟩. p∈ℤ₊″
    fix n assume A5: n∈ℤ₊
    with III have T2: N·n ∈ ℤ₊
      using int0.pos˙int˙closed˙mul˙unfold by simp
    from III A5 have
      N·n = n·N  and n·M = M·n
      using PositiveSet˙def int0.Int˙ZF˙1˙1˙L5 by auto
    moreover
    from A1 I II III IV A5 have
      IsBoundedAbove(S,OrderOnReals)
      x¡y  y∈S
      N ∈ ℤ₊  M ∈ int
      M^R ¡ y·N^R  n ∈ ℤ₊
      by auto
    then have n·M ≤ Γ(S,n·N) by (rule Real˙ZF˙1˙4˙L24)
    moreover from A1 A4 VI T2 have h(N·n) = Γ(S,N·n)
      using Real˙ZF˙1˙4˙L24A by simp
    ultimately show M·n ≤ h(N·n) by auto
  qed
  ultimately have M^R ≤ [h]·N^R using Real˙ZF˙1˙4˙L23
    by simp
  with III T1 have M^R·(N^R)^−1 ≤ [h]
    using int˙pos˙is˙real˙pos Real˙ZF˙1˙1˙L3 Real˙ZF˙1˙3˙L4B
    by simp
  with V show x ≤ [h] by (rule real˙ord˙transitive)
qed
```

The essential condition to claim that the candidate for supremum of S is less or equal than any upper bound of S.

```
lemma (in real1) Real˙ZF˙1˙4˙L26:
  assumes A1: IsBoundedAbove(S,OrderOnReals) and
  A2: x≤y  x∈S  and
  A4: N ∈ ℤ₊  M ∈ int and
  A5: y·N^R ¡ M^R  and A6: p ∈ ℤ₊
  shows ⌊(N·p)^R·x⌋ ≤ M·p
proof -
  from A2 A4 A6 have T:
    p·N ∈ ℤ₊  p ∈ int  N ∈ int
    p^R ∈ ℝ₊ p^R ∈ ℝ  N^R ∈ ℝ  x ∈ ℝ  y ∈ ℝ
    using int0.pos˙int˙closed˙mul˙unfold PositiveSet˙def
      real˙int˙is˙real Real˙ZF˙1˙2˙L15 int˙pos˙is˙real˙pos
    by auto
  with A2 have (p·N)^R·x ≤ (p·N)^R·y
    using int˙pos˙is˙real˙pos Real˙ZF˙1˙2˙L14A
    by simp
  moreover from A4 T have I:
    (p·N)^R = p^R·N^R
    (p·M)^R = p^R·M^R
    using Real˙ZF˙1˙4˙L1C by auto
```

```
  ultimately have (p·N)^R·x ≤ p^R·N^R·y
    by simp
  moreover
  from A5 T I have p^R·(y·N^R) ¡ (p·M)^R
    using Real˙ZF˙1˙3˙L7 by simp
  with T have p^R·N^R·y ¡ (p·M)^R using Real˙ZF˙1˙1˙L9
    by simp
  ultimately have (p·N)^R·x ¡ (p·M)^R
    by (rule real˙strict˙ord˙transit)
  then have ⌊(p·N)^R·x⌋ ≤ ⌊(p·M)^R⌋
    using Real˙ZF˙1˙4˙L9 by simp
  moreover
  from A4 T have p·M ∈ int using int0.Int˙ZF˙1˙1˙L5
    by simp
  then have ⌊(p·M)^R⌋ = p·M using Real˙ZF˙1˙4˙L14
    by simp
   moreover from A4 A6 have p·N = N·p and p·M = M·p
    using PositiveSet˙def int0.Int˙ZF˙1˙1˙L5 by auto
  ultimately show ⌊(N·p)^R·x⌋ ≤ M·p by simp
qed
```

A piece of the proof of the fact that the candidate for the supremum of S is not greater than any upper bound of S, done separately for clarity (of mind).

```
lemma (in real1) Real˙ZF˙1˙4˙L27:
  assumes IsBoundedAbove(S,OrderOnReals)  S≠0 and
  h = OddExtension(int,IntegerAddition,IntegerOrder,−⟨p,Γ(S,p)⟩. p∈ℤ₊˝)
  and p ∈ ℤ₊
  shows ∃x∈S. h(p) = ⌊p^R·x⌋
  using assms Real˙ZF˙1˙4˙L10 Real˙ZF˙1˙4˙L24A by auto
```

The candidate for the supremum of S is not greater than any upper bound of S.

```
lemma (in real1) Real˙ZF˙1˙4˙L28:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0
  and A2: ∀x∈S. x≤y and A3:
  h = OddExtension(int,IntegerAddition,IntegerOrder,−⟨p,Γ(S,p)⟩. p∈ℤ₊˝)
  shows [h] ≤ y
proof -
  from A1 obtain a where a∈S by auto
  with A1 A2 A3 have T: y∈ℝ  h ∈ 𝒮  [h] ∈ ℝ
    using Real˙ZF˙1˙2˙L15 Real˙ZF˙1˙4˙L22 Real˙ZF˙1˙1˙L3
    by auto
  { assume ¬([h] ≤ y)
    with T have y ¡ [h] using Real˙ZF˙1˙2˙L28
      by blast
    then have ∃M∈int. ∃N∈ℤ₊.  y·N^R ¡ M^R ∧ M^R ¡ [h]·N^R
      using Arthan˙Lemma14iii by simp
    then obtain M N where I: M∈int  N∈ℤ₊ and
```

```
    II: y·N^R ¡ M^R  M^R ¡ [h]·N^R
      by auto
    from I have III: N^R ∈ ℝ+ using int˙pos˙is˙real˙pos
      by simp
    have ∀p∈ℤ+. h(N·p) ≤  M·p
    proof
      fix p assume A4: p∈ℤ+
      with A1 A3 I have ∃x∈S. h(N·p) = ⌊(N·p)^R·x⌋
 using int0.pos˙int˙closed˙mul˙unfold Real˙ZF˙1˙4˙L27
 by simp
      with A1 A2 I II A4 show h(N·p) ≤  M·p
 using Real˙ZF˙1˙4˙L26 by auto
    qed
    with T I have [h]·N^R ≤ M^R
      using PositiveSet˙def Real˙ZF˙1˙4˙L23A
      by simp
    with T III have [h] ≤  M^R·(N^R)^−1
      using Real˙ZF˙1˙3˙L4C by simp
    moreover from T II III have M^R·(N^R)^−1 ¡ [h]
      using Real˙ZF˙1˙3˙L4A by simp
    ultimately have False using Real˙ZF˙1˙2˙L29 by blast
  } then show [h] ≤ y by auto
qed
```

Now we can prove that every nonempty subset of reals that is bounded above has a supremum. Proof by considering two cases: when the set has a maximum and when it does not.

```
lemma (in real1) real˙order˙complete:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0
  shows HasAminimum(OrderOnReals,⋂a∈S. OrderOnReals–a″)
proof -
  { assume HasAmaximum(OrderOnReals,S)
    with A1 have HasAminimum(OrderOnReals,⋂a∈S. OrderOnReals–a″)
      using Real˙ZF˙1˙2˙L10 IsAnOrdGroup˙def IsPartOrder˙def
 Order˙ZF˙5˙L6 by simp }
  moreover
  { assume A2: ¬HasAmaximum(OrderOnReals,S)
    let h = OddExtension(int,IntegerAddition,IntegerOrder,–⟨p,Γ(S,p)⟩. p∈ℤ+″)
    let r = OrderOnReals
    from A1 have antisym(OrderOnReals)  S≠0
      using Real˙ZF˙1˙2˙L10 IsAnOrdGroup˙def IsPartOrder˙def by auto
    moreover from A1 A2 have ∀x∈S. ⟨x,[h]⟩ ∈ r
      using Real˙ZF˙1˙4˙L25 by simp
    moreover from A1 have ∀y. (∀x∈S. ⟨x,y⟩ ∈ r) ⟶ ⟨[h],y⟩ ∈ r
      using Real˙ZF˙1˙4˙L28 by simp
    ultimately have HasAminimum(OrderOnReals,⋂a∈S. OrderOnReals–a″)
      by (rule Order˙ZF˙5˙L5) }
  ultimately show thesis by blast
qed
```

Finally, we are ready to formulate the main result: that the construction of real numbers from the additive group of integers results in a complete ordered field. This theorem completes the construction. It was fun.

```
theorem eudoxus˙reals˙are˙reals: shows
  IsAmodelOfReals(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
  using real1.reals˙are˙ord˙field real1.real˙order˙complete
    IsComplete˙def IsAmodelOfReals˙def by simp

end
```

49 Complex numbers

```
theory Complex˙ZF imports func˙ZF˙1 OrderedField˙ZF

begin
```

The goal of this theory is to define complex numbers and prove that the Metamath complex numbers axioms hold.

49.1 From complete ordered fields to complex numbers

This section consists mostly of definitions and a proof context for talking about complex numbers. Suppose we have a set R with binary operations A and M and a relation r such that the quadruple (R, A, M, r) forms a complete ordered field. The next definitions take (R, A, M, r) and construct the sets that represent the structure of complex numbers: the carrier ($\mathbb{C} = R \times R$), binary operations of addition and multiplication of complex numbers and the order relation on $\mathbb{R} = R \times 0$. The ImCxAdd, ReCxAdd, ImCxMul, ReCxMul are helper meta-functions representing the imaginary part of a sum of complex numbers, the real part of a sum of real numbers, the imaginary part of a product of complex numbers and the real part of a product of real numbers, respectively. The actual operations (subsets of $(R \times R) \times R$ are named CplxAdd and CplxMul.

When R is an ordered field, it comes with an order relation. This induces a natural strict order relation on $\{\langle x, 0\rangle : x \in R\} \subseteq R \times R$. We call the set $\{\langle x, 0\rangle : x \in R\}$ ComplexReals(R,A) and the strict order relation CplxROrder(R,A,r). The order on the real axis of complex numbers is defined as the relation induced on it by the canonical projection on the first coordinate and the order we have on the real numbers. OK, lets repeat this slower. We start with the order relation r on a (model of) real numbers R. We want to define an order relation on a subset of complex numbers, namely on $R \times \{0\}$. To do that we use the notion of a relation induced by a mapping. The mapping here is $f : R \times \{0\} \to R, f\langle x, 0\rangle = x$ which is defined under a name of SliceProjection in func˙ZF.thy. This defines a relation r_1 (called

InducedRelation(f,r), see func˙ZF) on $R \times \{0\}$ such that $\langle\langle x, 0\rangle, \langle y, 0\rangle \in r_1$ iff $\langle x, y\rangle \in r$. This way we get what we call CplxROrder(R,A,r). However, this is not the end of the story, because Metamath uses strict inequalities in its axioms, rather than weak ones like IsarMathLib (mostly). So we need to take the strict version of this order relation. This is done in the syntax definition of $<_{\mathbb{R}}$ in the definition of complex0 context. Since Metamath proves a lot of theorems about the real numbers extended with $+\infty$ and $-\infty$, we define the notation for inequalites on the extended real line as well.

A helper expression representing the real part of the sum of two complex numbers.

definition
ReCxAdd(R,A,a,b) ≡ A⟨fst(a),fst(b)⟩

An expression representing the imaginary part of the sum of two complex numbers.

definition
ImCxAdd(R,A,a,b) ≡ A⟨snd(a),snd(b)⟩

The set (function) that is the binary operation that adds complex numbers.

definition
CplxAdd(R,A) ≡
–⟨p, ⟨ ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p)) ⟩ ⟩.
p∈(R×R)×(R×R)″

The expression representing the imaginary part of the product of complex numbers.

definition
ImCxMul(R,A,M,a,b) ≡ A⟨M⟨fst(a),snd(b)⟩, M⟨snd(a),fst(b)⟩ ⟩

The expression representing the real part of the product of complex numbers.

definition
ReCxMul(R,A,M,a,b) ≡
A⟨M⟨fst(a),fst(b)⟩,GroupInv(R,A)(M⟨snd(a),snd(b)⟩)⟩

The function (set) that represents the binary operation of multiplication of complex numbers.

definition
CplxMul(R,A,M) ≡
– ⟨p, ⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩ ⟩.
p ∈ (R×R)×(R×R)″

The definition real numbers embedded in the complex plane.

definition
ComplexReals(R,A) ≡ R×–TheNeutralElement(R,A)″

Definition of order relation on the real line.

definition
CplxROrder(R,A,r) ≡
InducedRelation(SliceProjection(ComplexReals(R,A)),r)

The next locale defines proof context and notation that will be used for complex numbers.

locale complex0 =
fixes R and A and M and r
assumes R˙are˙reals: IsAmodelOfReals(R,A,M,r)

fixes complex (ℂ)
defines complex˙def[simp]: ℂ ≡ R×R

fixes rone ($\mathbf{1}_R$)
defines rone˙def[simp]: $\mathbf{1}_R$ ≡ TheNeutralElement(R,M)

fixes rzero ($\mathbf{0}_R$)
defines rzero˙def[simp]: $\mathbf{0}_R$ ≡ TheNeutralElement(R,A)

fixes one ($\mathbf{1}$)
defines one˙def[simp]: $\mathbf{1}$ ≡ ⟨$\mathbf{1}_R$, $\mathbf{0}_R$⟩

fixes zero ($\mathbf{0}$)
defines zero˙def[simp]: $\mathbf{0}$ ≡ ⟨$\mathbf{0}_R$, $\mathbf{0}_R$⟩

fixes iunit (i)
defines iunit˙def[simp]: i ≡ ⟨$\mathbf{0}_R$,$\mathbf{1}_R$⟩

fixes creal (ℝ)
defines creal˙def[simp]: ℝ ≡ −⟨r,$\mathbf{0}_R$⟩. r∈R″

fixes rmul (infixl · 71)
defines rmul˙def[simp]: a · b ≡ M⟨a,b⟩

fixes radd (infixl + 69)
defines radd˙def[simp]: a + b ≡ A⟨a,b⟩

fixes rneg (- ˙ 70)
defines rneg˙def[simp]: - a ≡ GroupInv(R,A)(a)

fixes ca (infixl + 69)
defines ca˙def[simp]: a + b ≡ CplxAdd(R,A)⟨a,b⟩

fixes cm (infixl · 71)
defines cm˙def[simp]: a · b ≡ CplxMul(R,A,M)⟨a,b⟩

fixes cdiv (infixl / 70)
defines cdiv˙def[simp]: a / b ≡ ⋃ − x ∈ ℂ. b · x = a ″

fixes sub (infixl - 69)
defines sub˙def[simp]: a - b ≡ $\bigcup$ − x ∈ ℂ. b + x = a ″

fixes cneg (-˙ 95)
defines cneg˙def[simp]: - a ≡ **0** - a

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

fixes cpnf (+∞)
defines cpnf˙def[simp]: +∞ ≡ ℂ

fixes cmnf (−∞)
defines cmnf˙def[simp]: −∞ ≡ −ℂ ″

fixes cxr ($\mathbb{R}^*$)
defines cxr˙def[simp]: $\mathbb{R}^*$ ≡ ℝ ∪ −+∞,−∞″

fixes cxn (ℕ)
defines cxn˙def[simp]:
ℕ ≡ $\bigcap$ −N ∈ Pow(ℝ). **1** ∈ N ∧ (∀n. n∈N ⟶ n+**1** ∈ N)″

fixes cltrrset (¡)
defines cltrrset˙def[simp]:
¡ ≡ StrictVersion(CplxROrder(R,A,r)) ∩ ℝ×ℝ ∪
−⟨−∞,+∞⟩″ ∪ (ℝ×−+∞″) ∪ (−−∞″×ℝ)

fixes cltrr (infix ¡ 68)
defines cltrr˙def[simp]: a ¡ b ≡ ⟨a,b⟩ ∈ ¡

fixes lsq (infix ≤ 68)
defines lsq˙def[simp]: a ≤ b ≡ ¬ (b ¡ a)

fixes two (**2**)
defines two˙def[simp]: **2** ≡ **1** + **1**

fixes three (**3**)
defines three˙def[simp]: **3** ≡ **2**+**1**

fixes four (**4**)
defines four˙def[simp]: **4** ≡ **3**+**1**

fixes five (**5**)
defines five˙def[simp]: **5** ≡ **4**+**1**

fixes six (**6**)
defines six˙def[simp]: **6** ≡ **5**+**1**

```
  fixes seven (7)
  defines seven˙def[simp]: 7 ≡ 6+1

  fixes eight (8)
  defines eight˙def[simp]: 8 ≡ 7+1

  fixes nine (9)
  defines nine˙def[simp]: 9 ≡ 8+1
```

49.2 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,z_1,z_2) ∈ R
ReCxMul(R,A,M,z_1,z_2) ∈ R
using cplx˙mul˙add˙defs by auto
qed

Complex reals are complex. Recall the definition of ℝ in the complex0 locale.

lemma (in complex0) axresscn: shows ℝ ⊆ ℂ
using valid˙cntxts group0.group0˙2˙L2 by auto

Complex 1 is not complex 0.

lemma (in complex0) ax1ne0: shows **1** ≠ **0**
proof -
have IsAfield(R,A,M) using valid˙cntxts field0.Field˙ZF˙1˙L1
by simp
then show **1** ≠ **0** using IsAfield˙def by auto
qed

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

lemma (in complex0) axaddopr: shows CplxAdd(R,A): ℂ × ℂ → ℂ
proof -
have ∀p ∈ ℂ×ℂ.
⟨ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p))⟩ ∈ ℂ
using cplx˙mul˙add˙types by simp
then have
−⟨p,⟨ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p))⟩ ⟩.
p ∈ ℂ×ℂ ″: ℂ×ℂ → ℂ
by (rule ZF˙fun˙from˙total)
then show CplxAdd(R,A): ℂ × ℂ → ℂ using CplxAdd˙def by simp
qed

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

lemma (in complex0) axmulopr: shows CplxMul(R,A,M): ℂ × ℂ → ℂ
proof -
have ∀p ∈ ℂ×ℂ.
⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩ ∈ ℂ
using cplx˙mul˙add˙types by simp
then have
−⟨p,⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩⟩.
p ∈ ℂ×ℂ ″: ℂ×ℂ → ℂ by (rule ZF˙fun˙from˙total)
then show CplxMul(R,A,M): ℂ × ℂ → ℂ using CplxMul˙def by simp
qed

What are the values of omplex addition and multiplication in terms of their real and imaginary parts?

lemma (in complex0) cplx˙mul˙add˙vals:
assumes A1: a∈R b∈R c∈R d∈R
shows

```
  ⟨a,b⟩ + ⟨c,d⟩ = ⟨a + c, b + d⟩
  ⟨a,b⟩ · ⟨c,d⟩ = ⟨a·c + (-b·d), a·d + b·c⟩
proof -
  let S = CplxAdd(R,A)
  let P = CplxMul(R,A,M)
  let p = ⟨ ⟨a,b⟩, ⟨c,d⟩ ⟩
  from A1 have S : ℂ × ℂ → ℂ and p ∈ ℂ × ℂ
    using axaddopr by auto
  moreover have
    S = −⟨p, ¡ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p))¿⟩.
    p ∈ ℂ × ℂ ″
    using CplxAdd˙def by simp
 ultimately have S(p) = ⟨ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p))⟩
    by (rule ZF˙fun˙from˙tot˙val)
  then show ⟨a,b⟩ + ⟨c,d⟩ = ⟨a + c, b + d⟩
    using cplx˙mul˙add˙defs by simp
  from A1 have P : ℂ × ℂ → ℂ and p ∈ ℂ × ℂ
    using axmulopr by auto
  moreover have
    P = −⟨p, ⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩ ⟩.
    p ∈ ℂ × ℂ ″
    using CplxMul˙def by simp
  ultimately have
    P(p) = ⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩
    by (rule ZF˙fun˙from˙tot˙val)
  then show ⟨a,b⟩ · ⟨c,d⟩ = ⟨a·c + (-b·d), a·d + b·c⟩
    using cplx˙mul˙add˙defs by simp
qed
```

Complex multiplication is commutative.

```
lemma (in complex0) axmulcom: assumes A1: a ∈ ℂ  b ∈ ℂ
  shows a·b = b·a
  using assms cplx˙mul˙add˙vals valid˙cntxts ring0.Ring˙ZF˙1˙L4
      field0.field˙mult˙comm by auto
```

A sum of complex numbers is complex.

```
lemma (in complex0) axaddcl: assumes a ∈ ℂ  b ∈ ℂ
  shows a+b ∈ ℂ
  using assms axaddopr apply˙funtype by simp
```

A product of complex numbers is complex.

```
lemma (in complex0) axmulcl: assumes a ∈ ℂ  b ∈ ℂ
  shows  a·b ∈ ℂ
  using assms axmulopr apply˙funtype by simp
```

Multiplication is distributive with respect to addition.

```
lemma (in complex0) axdistr:
  assumes A1: a ∈ ℂ  b ∈ ℂ  c ∈ ℂ
```

shows a·(b + c) = a·b + a·c
proof -
let a_r = fst(a)
let a_i = snd(a)
let b_r = fst(b)
let b_i = snd(b)
let c_r = fst(c)
let c_i = snd(c)
from A1 have T:
$a_r \in R$ $a_i \in R$ $b_r \in R$ $b_i \in R$ $c_r \in R$ $c_i \in R$
$b_r+c_r \in R$ $b_i+c_i \in R$
$a_r \cdot b_r + (-a_i \cdot b_i) \in R$
$a_r \cdot c_r + (-a_i \cdot c_i) \in R$
$a_r \cdot b_i + a_i \cdot b_r \in R$
$a_r \cdot c_i + a_i \cdot c_r \in R$
using valid˙cntxts ring0.Ring˙ZF˙1˙L4 by auto
with A1 have a·(b + c) =
$\langle a_r \cdot (b_r+c_r) + (-a_i \cdot (b_i+c_i)), a_r \cdot (b_i+c_i) + a_i \cdot (b_r+c_r)\rangle$
using cplx˙mul˙add˙vals by auto
moreover from T have
$a_r \cdot (b_r+c_r) + (-a_i \cdot (b_i+c_i)) =$
$a_r \cdot b_r + (-a_i \cdot b_i) + (a_r \cdot c_r + (-a_i \cdot c_i))$
and
$a_r \cdot (b_i+c_i) + a_i \cdot (b_r+c_r) =$
$a_r \cdot b_i + a_i \cdot b_r + (a_r \cdot c_i + a_i \cdot c_r)$
using valid˙cntxts ring0.Ring˙ZF˙2˙L6 by auto
moreover from A1 T have
$\langle a_r \cdot b_r + (-a_i \cdot b_i) + (a_r \cdot c_r + (-a_i \cdot c_i)),$
$a_r \cdot b_i + a_i \cdot b_r + (a_r \cdot c_i + a_i \cdot c_r)\rangle =$
a·b + a·c
using cplx˙mul˙add˙vals by auto
ultimately show a·(b + c) = a·b + a·c
by simp
qed

Complex addition is commutative.

lemma (in complex0) axaddcom: assumes a ∈ ℂ b ∈ ℂ
shows a+b = b+a
using assms cplx˙mul˙add˙vals valid˙cntxts ring0.Ring˙ZF˙1˙L4
by auto

Complex addition is associative.

lemma (in complex0) axaddass: assumes A1: a ∈ ℂ b ∈ ℂ c ∈ ℂ
shows a + b + c = a + (b + c)
proof -
let a_r = fst(a)
let a_i = snd(a)
let b_r = fst(b)
let b_i = snd(b)

let c_r = fst(c)
let c_i = snd(c)
from A1 have T:
$a_r \in R$ $a_i \in R$ $b_r \in R$ $b_i \in R$ $c_r \in R$ $c_i \in R$
$a_r+b_r \in R$ $a_i+b_i \in R$
$b_r+c_r \in R$ $b_i+c_i \in R$
using valid˙cntxts ring0.Ring˙ZF˙1˙L4 by auto
with A1 have a + b + c = $\langle a_r+b_r+c_r,a_i+b_i+c_i\rangle$
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:
$a_r \in R$ $a_i \in R$ $b_r \in R$ $b_i \in R$ $c_r \in R$ $c_i \in R$
$a_r \cdot b_r + (-a_i \cdot b_i) \in R$
$a_r \cdot b_i + a_i \cdot b_r \in R$
$b_r \cdot c_r + (-b_i \cdot c_i) \in R$
$b_r \cdot c_i + b_i \cdot c_r \in R$
using valid˙cntxts ring0.Ring˙ZF˙1˙L4 by auto
with A1 have a · b · c =
$\langle (a_r \cdot b_r + (-a_i \cdot b_i)) \cdot c_r + (-(a_r \cdot b_i + a_i \cdot b_r) \cdot c_i),$
$(a_r \cdot b_r + (-a_i \cdot b_i)) \cdot c_i + (a_r \cdot b_i + a_i \cdot b_r) \cdot c_r\rangle$
using cplx˙mul˙add˙vals by auto
moreover from A1 T have
$\langle a_r \cdot (b_r \cdot c_r + (-b_i \cdot c_i)) + (-a_i \cdot (b_r \cdot c_i + b_i \cdot c_r)),$
$a_r \cdot (b_r \cdot c_i + b_i \cdot c_r) + a_i \cdot (b_r \cdot c_r + (-b_i \cdot c_i))\rangle =$
a · (b · c)
using cplx˙mul˙add˙vals by auto
moreover from T have
$a_r \cdot (b_r \cdot c_r + (-b_i \cdot c_i)) + (-a_i \cdot (b_r \cdot c_i + b_i \cdot c_r)) =$
$(a_r \cdot b_r + (-a_i \cdot b_i)) \cdot c_r + (-(a_r \cdot b_i + a_i \cdot b_r) \cdot c_i)$
and
$a_r \cdot (b_r \cdot c_i + b_i \cdot c_r) + a_i \cdot (b_r \cdot c_r + (-b_i \cdot c_i)) =$
$(a_r \cdot b_r + (-a_i \cdot b_i)) \cdot c_i + (a_r \cdot b_i + a_i \cdot b_r) \cdot c_r$
using valid˙cntxts ring0.Ring˙ZF˙2˙L6 by auto

```
  ultimately show a · b · c = a · (b · c)
    by auto
qed
```

Complex 1 is real. This really means that the pair $\langle 1, 0\rangle$ is on the real axis.

```
lemma (in complex0) ax1re: shows 1 ∈ ℝ
  using valid˙cntxts ring0.Ring˙ZF˙1˙L2 by simp
```

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

```
lemma (in complex0) axi2m1: shows i·i + 1 = 0
  using valid˙cntxts ring0.Ring˙ZF˙1˙L2 ring0.Ring˙ZF˙1˙L3
  cplx˙mul˙add˙vals ring0.Ring˙ZF˙1˙L6 group0.group0˙2˙L6
  by simp
```

0 is the neutral element of complex addition.

```
lemma (in complex0) ax0id: assumes a ∈ ℂ
  shows a + 0 = a
  using assms cplx˙mul˙add˙vals valid˙cntxts
    ring0.Ring˙ZF˙1˙L2 ring0.Ring˙ZF˙1˙L3
  by auto
```

The imaginary unit is a complex number.

```
lemma (in complex0) axicn: shows i ∈ ℂ
  using valid˙cntxts ring0.Ring˙ZF˙1˙L2 by auto
```

All complex numbers have additive inverses.

```
lemma (in complex0) axnegex: assumes A1: a ∈ ℂ
  shows ∃x∈ℂ. a + x  = 0
proof -
  let a_r = fst(a)
  let a_i = snd(a)
  let x = ⟨-a_r, -a_i⟩
  from A1 have T:
    a_r ∈ R   a_i ∈ R   (-a_r) ∈ R   (-a_r) ∈ R
    using valid˙cntxts ring0.Ring˙ZF˙1˙L3 by auto
  then have x ∈ ℂ using valid˙cntxts ring0.Ring˙ZF˙1˙L3
    by auto
  moreover from A1 T have a + x = 0
    using cplx˙mul˙add˙vals valid˙cntxts ring0.Ring˙ZF˙1˙L3
    by auto
  ultimately show ∃x∈ℂ. a + x  = 0
    by auto
qed
```

A non-zero complex number has a multiplicative inverse.

```
lemma (in complex0) axrecex: assumes A1: a ∈ ℂ and A2: a≠0
  shows ∃x∈ℂ. a·x = 1
proof -
```

```
  let a_r = fst(a)
  let a_i = snd(a)
  let m = a_r·a_r + a_i·a_i
  from A1 have T1: a_r ∈ R   a_i ∈ R by auto
  moreover from A1 A2 have a_r ≠ 0_R ∨ a_i ≠ 0_R
    by auto
  ultimately have ∃c∈R. m·c = 1_R
    using valid'cntxts field1.OrdField'ZF'1'L10
    by auto
  then obtain c where I: c∈R and II: m·c = 1_R
    by auto
  let x = ⟨a_r·c, -a_i·c⟩
  from T1 I have T2: a_r·c ∈ R  (-a_i·c) ∈ R
    using valid'cntxts ring0.Ring'ZF'1'L4 ring0.Ring'ZF'1'L3
    by auto
  then have x ∈ ℂ by auto
  moreover from A1 T1 T2 I II have a·x = 1
    using cplx'mul'add'vals valid'cntxts ring0.ring'rearr'3'elemA
    by auto
  ultimately show ∃x∈ℂ. a·x = 1 by auto
qed
```

Complex 1 is a right neutral element for multiplication.

```
lemma (in complex0) ax1id: assumes A1: a ∈ ℂ
  shows a·1 = a
  using assms valid'cntxts ring0.Ring'ZF'1'L2 cplx'mul'add'vals
    ring0.Ring'ZF'1'L3 ring0.Ring'ZF'1'L6 by auto
```

A formula for sum of (complex) real numbers.

```
lemma (in complex0) sum'of'reals: assumes a∈ℝ  b∈ℝ
  shows
  a + b = ⟨fst(a) + fst(b),0_R⟩
  using assms valid'cntxts ring0.Ring'ZF'1'L2 cplx'mul'add'vals
    ring0.Ring'ZF'1'L3 by auto
```

The sum of real numbers is real.

```
lemma (in complex0) axaddrcl: assumes A1: a∈ℝ  b∈ℝ
  shows a + b ∈ ℝ
  using assms sum'of'reals valid'cntxts ring0.Ring'ZF'1'L4
  by auto
```

The formula for the product of (complex) real numbers.

```
lemma (in complex0) prod'of'reals: assumes A1: a∈ℝ  b∈ℝ
  shows a · b = ⟨fst(a)·fst(b),0_R⟩
proof -
  let a_r = fst(a)
  let b_r = fst(b)
  from A1 have T:
```

```
    a_r ∈ R b_r ∈ R  0_R ∈ R  a_r·b_r ∈ R
    using valid˙cntxts ring0.Ring˙ZF˙1˙L2 ring0.Ring˙ZF˙1˙L4
    by auto
  with A1 show a · b = ⟨a_r·b_r,0_R⟩
    using cplx˙mul˙add˙vals valid˙cntxts ring0.Ring˙ZF˙1˙L2
      ring0.Ring˙ZF˙1˙L6 ring0.Ring˙ZF˙1˙L3 by auto
qed
```

The product of (complex) real numbers is real.

```
lemma (in complex0) axmulrcl: assumes a∈ℝ  b∈ℝ
  shows a · b ∈ ℝ
  using assms prod˙of˙reals valid˙cntxts ring0.Ring˙ZF˙1˙L4
  by auto
```

The existence of a real negative of a real number.

```
lemma (in complex0) axrnegex: assumes A1: a∈ℝ
  shows ∃ x ∈ ℝ. a + x = 0
proof -
  let a_r = fst(a)
  let x = ⟨-a_r,0_R⟩
  from A1 have T:
    a_r ∈ R  (-a_r) ∈ R  0_R ∈ R
    using valid˙cntxts ring0.Ring˙ZF˙1˙L3 ring0.Ring˙ZF˙1˙L2
    by auto
  then have x∈ ℝ by auto
  moreover from A1 T have a + x = 0
    using cplx˙mul˙add˙vals valid˙cntxts ring0.Ring˙ZF˙1˙L3
    by auto
  ultimately show ∃x∈ℝ. a + x = 0 by auto
qed
```

Each nonzero real number has a real inverse

```
lemma (in complex0) axrrecex:
  assumes A1:  a ∈ ℝ   a ≠ 0
  shows ∃x∈ℝ. a · x = 1
proof -
  let R_0 = R-–0_R″
  let a_r = fst(a)
  let y = GroupInv(R_0,restrict(M,R_0×R_0))(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) $\subseteq \mathbb{R}\times\mathbb{R}$
using ComplexReals˙def slice˙proj˙bij real˙means˙real˙axis
CplxROrder˙def InducedRelation˙def by auto

The strict version of the complex relation is a relation on complex reals.

lemma (in complex0) cplx˙strict˙ord˙on˙cplx˙reals:
shows StrictVersion(CplxROrder(R,A,r)) $\subseteq \mathbb{R}\times\mathbb{R}$
using cplx˙ord˙on˙cplx˙reals strict˙ver˙rel by simp

The CplxROrder thing is a relation on the complex reals. Here this is formulated as a statement that in complex0 context $a < b$ implies that a, b are complex reals

lemma (in complex0) strict˙cplx˙ord˙type: assumes a $<_{\mathbb{R}}$ b
shows a$\in\mathbb{R}$ b$\in\mathbb{R}$
using assms CplxROrder˙def def˙of˙strict˙ver InducedRelation˙def
slice˙proj˙bij ComplexReals˙def real˙means˙real˙axis
by auto

A more readable version of the definition of the strict order relation on the real axis. Recall that in the complex0 context r denotes the (non-strict) order relation on the underlying model of real numbers.

lemma (in complex0) def˙of˙real˙axis˙order: shows
$\langle$x,$\mathbf{0}_R\rangle <_{\mathbb{R}} \langle$y,$\mathbf{0}_R\rangle \longleftrightarrow \langle$x,y$\rangle \in$ r $\wedge$ x$\neq$y
proof
let f = SliceProjection(ComplexReals(R,A))
assume A1: $\langle$x,$\mathbf{0}_R\rangle <_{\mathbb{R}} \langle$y,$\mathbf{0}_R\rangle$
then have $\langle$ f$\langle$x,$\mathbf{0}_R\rangle$, f$\langle$y,$\mathbf{0}_R\rangle$ $\rangle \in$ r $\wedge$ x $\neq$ y
using CplxROrder˙def def˙of˙strict˙ver def˙of˙ind˙relA
by simp
moreover from A1 have $\langle$x,$\mathbf{0}_R\rangle \in \mathbb{R}$ $\langle$y,$\mathbf{0}_R\rangle \in \mathbb{R}$
using strict˙cplx˙ord˙type by auto
ultimately show $\langle$x,y$\rangle \in$ r $\wedge$ x$\neq$y
using slice˙proj˙bij ComplexReals˙def by simp
next assume A1: $\langle$x,y$\rangle \in$ r $\wedge$ x$\neq$y
let f = SliceProjection(ComplexReals(R,A))
have f : $\mathbb{R} \to$ R
using ComplexReals˙def slice˙proj˙bij real˙means˙real˙axis
by simp
moreover from A1 have T: $\langle$x,$\mathbf{0}_R\rangle \in \mathbb{R}$ $\langle$y,$\mathbf{0}_R\rangle \in \mathbb{R}$
using valid˙cntxts ring1.OrdRing˙ZF˙1˙L3 by auto
moreover from A1 T have $\langle$ f$\langle$x,$\mathbf{0}_R\rangle$, f$\langle$y,$\mathbf{0}_R\rangle$ $\rangle \in$ r
using slice˙proj˙bij ComplexReals˙def by simp
ultimately have $\langle$ $\langle$x,$\mathbf{0}_R\rangle$, $\langle$y,$\mathbf{0}_R\rangle$ $\rangle \in$ InducedRelation(f,r)
using def˙of˙ind˙relB by simp

```
  with A1 show ⟨x,0_R⟩ <_ℝ ⟨y,0_R⟩
    using CplxROrder˙def def˙of˙strict˙ver
    by simp
qed
```

The (non strict) order on complex reals is antisymmetric, transitive and total.

```
lemma (in complex0) cplx˙ord˙antsym˙trans˙tot: shows
  antisym(CplxROrder(R,A,r))
  trans(CplxROrder(R,A,r))
  CplxROrder(R,A,r) –is total on″ ℝ
proof -
  let f = SliceProjection(ComplexReals(R,A))
  have f ∈ ord˙iso(ℝ,CplxROrder(R,A,r),R,r)
    using ComplexReals˙def slice˙proj˙bij real˙means˙real˙axis
      bij˙is˙ord˙iso CplxROrder˙def by simp
  moreover have CplxROrder(R,A,r) ⊆ ℝ×ℝ
    using cplx˙ord˙on˙cplx˙reals by simp
  moreover have I:
    antisym(r)   r –is total on″ R   trans(r)
    using valid˙cntxts ring1.OrdRing˙ZF˙1˙L1 IsAnOrdRing˙def
      IsLinOrder˙def by auto
  ultimately show
    antisym(CplxROrder(R,A,r))
    trans(CplxROrder(R,A,r))
    CplxROrder(R,A,r) –is total on″ ℝ
    using ord˙iso˙pres˙antsym ord˙iso˙pres˙tot ord˙iso˙pres˙trans
    by auto
qed
```

The trichotomy law for the strict order on the complex reals.

```
lemma (in complex0) cplx˙strict˙ord˙trich:
  assumes a ∈ ℝ  b ∈ ℝ
  shows Exactly˙1˙of˙3˙holds(a<_ℝb, a=b, b<_ℝa)
  using assms cplx˙ord˙antsym˙trans˙tot strict˙ans˙tot˙trich
  by simp
```

The strict order on the complex reals is kind of antisymetric.

```
lemma (in complex0) pre˙axlttri: assumes A1: a ∈ ℝ   b ∈ ℝ
  shows a <_ℝ b ⟷ ¬(a=b ∨ b <_ℝ a)
proof -
  from A1 have Exactly˙1˙of˙3˙holds(a<_ℝb, a=b, b<_ℝa)
    by (rule cplx˙strict˙ord˙trich)
  then show a <_ℝ b ⟷ ¬(a=b ∨ b <_ℝ a)
    by (rule Fol1˙L8A)
qed
```

The strict order on complex reals is transitive.

```
lemma (in complex0) cplx˙strict˙ord˙trans:
```

```
  shows trans(StrictVersion(CplxROrder(R,A,r)))
  using cplx˙ord˙antsym˙trans˙tot strict˙of˙transB by simp
```

The strict order on complex reals is transitive - the explicit version of cplx˙strict˙ord˙trans.

```
lemma (in complex0) pre˙axlttrn:
  assumes A1: a <ℝ b   b <ℝ c
  shows a <ℝ c
proof -
  let s = StrictVersion(CplxROrder(R,A,r))
  from A1 have
    trans(s)   ⟨a,b⟩ ∈ s ∧ ⟨b,c⟩ ∈ s
    using cplx˙strict˙ord˙trans by auto
  then have ⟨a,c⟩ ∈ s by (rule Fol1˙L3)
  then show a <ℝ c by simp
qed
```

The strict order on complex reals is preserved by translations.

```
lemma (in complex0) pre˙axltadd:
  assumes A1: a <ℝ b and A2: c ∈ ℝ
  shows c+a <ℝ c+b
proof -
  from A1 have T: a∈ℝ  b∈ℝ using strict˙cplx˙ord˙type
    by auto
  with A1 A2 show c+a <ℝ c+b
    using def˙of˙real˙axis˙order valid˙cntxts
      group3.group˙strict˙ord˙transl˙inv sum˙of˙reals
    by auto
qed
```

The set of positive complex reals is closed with respect to multiplication.

```
lemma (in complex0) pre˙axmulgt0: assumes A1: 0 <ℝ a   0 <ℝ b
  shows 0 <ℝ a·b
proof -
  from A1 have T: a∈ℝ  b∈ℝ using strict˙cplx˙ord˙type
    by auto
  with A1 show 0 <ℝ a·b
    using def˙of˙real˙axis˙order valid˙cntxts field1.pos˙mul˙closed
      def˙of˙real˙axis˙order prod˙of˙reals
    by auto
qed
```

The order on complex reals is linear and complete.

```
lemma (in complex0) cmplx˙reals˙ord˙lin˙compl: shows
  CplxROrder(R,A,r) –is complete″
  IsLinOrder(ℝ,CplxROrder(R,A,r))
proof -
  have SliceProjection(ℝ) ∈ bij(ℝ,R)
```

```
    using slice˙proj˙bij ComplexReals˙def real˙means˙real˙axis
    by simp
  moreover have r ⊆ R×R using valid˙cntxts ring1.OrdRing˙ZF˙1˙L1
    IsAnOrdRing˙def by simp
  moreover from R˙are˙reals have
    r –is complete″ and IsLinOrder(R,r)
    using IsAmodelOfReals˙def valid˙cntxts ring1.OrdRing˙ZF˙1˙L1
    IsAnOrdRing˙def by auto
  ultimately show
    CplxROrder(R,A,r) –is complete″
    IsLinOrder(ℝ,CplxROrder(R,A,r))
    using CplxROrder˙def real˙means˙real˙axis ind˙rel˙pres˙compl
      ind˙rel˙pres˙lin by auto
qed
```

The property of the strict order on complex reals that corresponds to completeness.

```
lemma (in complex0) pre˙axsup: assumes A1: X ⊆ ℝ   X ≠ 0 and
  A2: ∃x∈ℝ. ∀y∈X. y <ℝ x
  shows
  ∃x∈ℝ. (∀y∈X. ¬(x <ℝ y)) ∧ (∀y∈ℝ. (y <ℝ x ⟶ (∃z∈X. y <ℝ z)))
proof -
  let s = StrictVersion(CplxROrder(R,A,r))
  have
    CplxROrder(R,A,r) ⊆ ℝ×ℝ
    IsLinOrder(ℝ,CplxROrder(R,A,r))
    CplxROrder(R,A,r) –is complete″
    using cplx˙ord˙on˙cplx˙reals cmplx˙reals˙ord˙lin˙compl
    by auto
  moreover note A1
  moreover have s = StrictVersion(CplxROrder(R,A,r))
    by simp
  moreover from A2 have ∃u∈ℝ. ∀y∈X. ⟨y,u⟩ ∈ s
    by simp
  ultimately have
    ∃x∈ℝ. ( ∀y∈X. ⟨x,y⟩ ∉ s ) ∧
    (∀y∈ℝ. ⟨y,x⟩ ∈ s ⟶ (∃z∈X. ⟨y,z⟩ ∈ s))
    by (rule strict˙of˙compl)
  then show (∃x∈ℝ. (∀y∈X. ¬(x <ℝ y)) ∧
    (∀y∈ℝ. (y <ℝ x ⟶ (∃z∈X. y <ℝ z))))
    by simp
qed

end
```

50 Topology - introduction

```
theory Topology˙ZF imports ZF1 Finite˙ZF Fol1
```

begin

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

50.1 Basic definitions and properties

A typical textbook defines a topology on a set X as a collection T of subsets of X such that $X \in T$, $\emptyset \in T$ and T is closed with respect to arbitrary unions and intersection of two sets. One can notice here that since we always have $\bigcup T = X$, the set on which the topology is defined (the "carrier" of the topology) can always be constructed from the topology itself and is superfluous in the definition. Moreover, as Marnix Klooster pointed out to me, the fact that the empty set is open can also be proven from other axioms. Hence, we define a topology as a collection of sets that is closed under arbitrary unions and intersections of two sets, without any mention of the set on which the topology is defined. Recall that Pow(T) is the powerset of T, so that if $M \in$ Pow(T) then M is a subset of T. The sets that belong to a topology T will be sometimes called "open in" T or just "open" if the topology is clear from the context.

Topology is a collection of sets that is closed under arbitrary unions and intersections of two sets.

definition
IsATopology (˙ –is a topology″ [90] 91) where
T –is a topology″ ≡ (∀M ∈ Pow(T). ⋃M ∈ T) ∧
(∀U∈T. ∀ V∈T. U∩V ∈ T)

We define interior of a set A as the union of all open sets contained in A. We use Interior(A,T) to denote the interior of A.

definition
Interior(A,T) ≡ ⋃ –U∈T. U ⊆ A″

A set is closed if it is contained in the carrier of topology and its complement is open.

definition
IsClosed (infixl –is closed in″ 90) where
D –is closed in″ T ≡ (D ⊆ ⋃T ∧ ⋃T - D ∈ T)

To prove various properties of closure we will often use the collection of closed sets that contain a given set A. Such collection does not have a separate name in informal math. We will call it ClosedCovers(A,T).

definition
ClosedCovers(A,T) ≡ –D ∈ Pow(⋃T). D –is closed in″ T ∧ A⊆D″

The closure of a set A is defined as the intersection of the collection of closed sets that contain A.

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

We also define boundary of a set as the intersection of its closure with the closure of the complement (with respect to the carrier).

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

A set K is compact if for every collection of open sets that covers K we can choose a finite one that still covers the set. Recall that FinPow(M) is the collection of finite subsets of M (finite powerset of M), defined in IsarMathLib's Finite˙ZF theory.

```
definition
  IsCompact (infixl –is compact in″ 90) where
  K –is compact in″ T ≡ (K ⊆ ⋃T ∧
  (∀ M∈Pow(T). K ⊆ ⋃M ⟶ (∃ N ∈ FinPow(M). K ⊆ ⋃N)))
```

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

```
lemma Pow˙is˙top: shows Pow(X) –is a topology″
proof -
  have ∀A∈Pow(Pow(X)). ⋃A ∈ Pow(X) by fast
  moreover have ∀U∈Pow(X). ∀V∈Pow(X). U∩V ∈ Pow(X) by fast
  ultimately show Pow(X) –is a topology″ using IsATopology˙def
    by auto
qed
```

Empty set is open.

```
lemma empty˙open:
  assumes T –is a topology″ shows 0 ∈ T
proof -
  have 0 ∈ Pow(T) by simp
  with assms have ⋃0 ∈ T using IsATopology˙def by blast
  thus 0 ∈ T by simp
qed
```

The carrier is open.

```
lemma carr˙open: assumes T –is a topology″ shows (⋃T) ∈ T
  using assms IsATopology˙def by auto
```

Union of a collection of open sets is open.

```
lemma union˙open: assumes T –is a topology″ and ∀A∈𝒜. A ∈ T
  shows (⋃𝒜) ∈ T using assms IsATopology˙def by auto
```

Union of a indexed family of open sets is open.

```
lemma union˙indexed˙open: assumes A1: T –is a topology″ and A2: ∀i∈I. P(i) ∈ T
```

```
  shows (⋃i∈I. P(i)) ∈ T  using assms union˙open by simp
```

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

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

We will now introduce some notation. In Isar, this is done by definining a "locale". Locale is kind of a context that holds some assumptions and notation used in all theorems proven in it. In the locale (context) below called topology0 we assume that T is a topology. The interior of the set A (with respect to the topology in the context) is denoted int(A). The closure of a set $A \subseteq \bigcup T$ is denoted cl(A) and the boundary is ∂A.

```
locale topology0 =
  fixes T
  assumes topSpaceAssum: T –is a topology″

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

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

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

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

```
lemma (in topology0) fin˙inter˙open˙open: assumes N≠0 N ∈ FinPow(T)
  shows ⋂N ∈ T
  using topSpaceAssum assms IsATopology˙def inter˙two˙inter˙fin
```

```
  by simp
```

Having a topology T and a set X we can define the induced topology as the one consisting of the intersections of X with sets from T. The notion of a collection restricted to a set is defined in ZF1.thy.

```
lemma (in topology0) Top˙1˙L4:
  shows (T –restricted to″ X) –is a topology″
proof -
  let S = T –restricted to″ X
  have ∀A∈Pow(S). ⋃A ∈ S
  proof
    fix A assume A1: A∈Pow(S)
    have ∀V∈A. ⋃ –U ∈ T. V = U∩X″ ∈ T
    proof -
      { fix V
 let M = –U ∈ T. V = U∩X″
 have M ∈ Pow(T) by auto
 with topSpaceAssum have ⋃M ∈ T using IsATopology˙def by simp
      } thus thesis by simp
    qed
    hence –⋃–U∈T. V = U∩X″.V∈ A″ ⊆ T by auto
    with topSpaceAssum have (⋃V∈A. ⋃–U∈T. V = U∩X″) ∈ T
      using IsATopology˙def by auto
    then have (⋃V∈A. ⋃–U∈T. V = U∩X″)∩ X ∈ S
      using RestrictedTo˙def by auto
    moreover
    from A1 have ∀V∈A. ∃U∈T. V = U∩X
      using RestrictedTo˙def by auto
    hence (⋃V∈A. ⋃–U∈T. V = U∩X″)∩X = ⋃A by blast
    ultimately show ⋃A ∈ S by simp
  qed
  moreover have  ∀U∈S. ∀V∈S. U∩V ∈ S
  proof -
    { fix U V assume U∈S  V∈S
      then obtain U1 V1 where
 U1 ∈ T ∧ U = U1∩X and V1 ∈ T ∧ V = V1∩X
 using RestrictedTo˙def by auto
      with topSpaceAssum have U1∩V1 ∈ T and U∩V = (U1∩V1)∩X
 using IsATopology˙def by auto
      then have  U∩V ∈ S using RestrictedTo˙def by auto
    } thus ∀U∈S. ∀  V∈S. U∩V ∈ S
      by simp
  qed
  ultimately show S –is a topology″ using IsATopology˙def
    by simp
qed
```

50.2 Interior of a set

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

Interior of a set A is contained in A.

```
lemma (in topology0) Top˙2˙L1: shows int(A) ⊆ A
  using Interior˙def by auto
```

Interior is open.

```
lemma (in topology0) Top˙2˙L2: shows int(A) ∈ T
proof -
  have –U∈T. U⊆A″ ∈ Pow(T) by auto
  with topSpaceAssum show int(A) ∈ T
    using IsATopology˙def Interior˙def by auto
qed
```

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

```
lemma (in topology0) Top˙2˙L3: shows U∈T ⟷ int(U) = U
proof
  assume U∈T then show int(U) = U
    using Interior˙def by auto
next assume A1: int(U) = U
  have int(U) ∈ T using Top˙2˙L2 by simp
  with A1 show U∈T by simp
qed
```

Interior of the interior is the interior.

```
lemma (in topology0) Top˙2˙L4: shows int(int(A)) = int(A)
proof -
  let U = int(A)
  from topSpaceAssum have U∈T using Top˙2˙L2 by simp
  then show int(int(A)) = int(A) using Top˙2˙L3 by simp
qed
```

Interior of a bigger set is bigger.

```
lemma (in topology0) interior˙mono:
  assumes A1: A⊆B shows int(A) ⊆ int(B)
proof -
  from A1 have ∀ U∈T. (U⊆A ⟶ U⊆B) by auto
  then show int(A) ⊆ int(B) using Interior˙def by auto
qed
```

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

```
lemma (in topology0) Top˙2˙L5: assumes U⊆A and U∈T
  shows U ⊆ int(A)
  using assms Interior˙def by auto
```

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

lemma (in topology0) Top˙2˙L6: assumes ∃U∈T. (x∈U ∧ U⊆A)
shows x ∈ int(A)
using assms Interior˙def by auto

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

lemma (in topology0) open˙open˙neigh:
assumes A1: V∈T
shows ∀x∈V. ∃U∈T. (x∈U ∧ U⊆V)
proof -
from A1 have ∀x∈V. V∈T ∧ x ∈ V ∧ V ⊆ V by simp
thus thesis by auto
qed

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

lemma (in topology0) open˙neigh˙open:
assumes A1: ∀x∈V. ∃U∈T. (x∈U ∧ U⊆V)
shows V∈T
proof -
from A1 have V = int(V) using Top˙2˙L1 Top˙2˙L6
by blast
then show V∈T using Top˙2˙L3 by simp
qed

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

lemma (in topology0) int˙inter˙int: shows int(A) ∩ int(B) = int(A∩B)
proof
have int(A) ∩ int(B) ⊆ A∩B using Top˙2˙L1 by auto
moreover have int(A) ∩ int(B) ∈ T using Top˙2˙L2 IsATopology˙def topSpaceAssum
by auto
ultimately show int(A) ∩ int(B) ⊆ int(A∩B) using Top˙2˙L5 by simp
have A∩B ⊆ A and A∩B ⊆ B by auto
then have int(A∩B) ⊆ int(A) and int(A∩B) ⊆ int(B) using interior˙mono by auto
thus int(A∩B) ⊆ int(A) ∩ int(B) by auto
qed

50.3 Closed sets, closure, boundary.

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

The carrier of the space is closed.

lemma (in topology0) Top˙3˙L1: shows (⋃T) –is closed in˝ T
proof -

```
  have ⋃T - ⋃T = 0 by auto
  with topSpaceAssum have ⋃T - ⋃T ∈ T using IsATopology˙def by auto
  then show thesis using IsClosed˙def by simp
qed
```

Empty set is closed.

```
lemma (in topology0) Top˙3˙L2: shows 0 –is closed in″ T
  using topSpaceAssum  IsATopology˙def IsClosed˙def by simp
```

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

```
lemma (in topology0) Top˙3˙L3:
  assumes A1: A ⊆ ⋃T shows ClosedCovers(A,T) ≠ 0
proof -
  from A1 have ⋃T ∈ ClosedCovers(A,T) using ClosedCovers˙def Top˙3˙L1
    by auto
  thus thesis by auto
qed
```

Intersection of a nonempty family of closed sets is closed.

```
lemma (in topology0) Top˙3˙L4: assumes A1: K≠0 and
  A2: ∀D∈K. D –is closed in″ T
  shows (⋂K) –is closed in″ T
proof -
  from A2 have I: ∀D∈K. (D ⊆ ⋃T ∧ (⋃T - D)∈ T)
    using IsClosed˙def by simp
  then have –⋃T - D. D∈ K″ ⊆ T by auto
  with topSpaceAssum have (⋃ –⋃T - D. D∈ K″) ∈ T
    using IsATopology˙def by auto
  moreover from A1 have ⋃ –⋃T - D. D∈ K″ = ⋃T - ⋂K by fast
  moreover from A1 I have ⋂K ⊆ ⋃T by blast
  ultimately show (⋂K) –is closed in″ T using  IsClosed˙def
    by simp
qed
```

The union and intersection of two closed sets are closed.

```
lemma (in topology0) Top˙3˙L5:
  assumes A1: D1 –is closed in″ T   D2 –is closed in″ T
  shows
  (D1∩D2) –is closed in″ T
  (D1∪D2) –is closed in″ T
proof -
  have –D1,D2″ ≠ 0 by simp
  with A1 have (⋂ –D1,D2″) –is closed in″ T using Top˙3˙L4
    by fast
  thus (D1∩D2) –is closed in″ T by simp
  from topSpaceAssum A1 have (⋃T - D1) ∩ (⋃T - D2) ∈ T
```

```
    using IsClosed˙def IsATopology˙def by simp
  moreover have (⋃T - D₁) ∩ (⋃T - D₂) = ⋃T - (D₁ ∪ D₂)
    by auto
  moreover from A1 have D₁ ∪ D₂ ⊆ ⋃T using IsClosed˙def
    by auto
  ultimately show (D₁∪D₂) –is closed in″ T using IsClosed˙def
    by simp
qed
```

Finite union of closed sets is closed. To understand the proof recall that $D \in$Pow($\bigcup$T) means that D is a subset of the carrier of the topology.

```
lemma (in topology0) fin˙union˙cl˙is˙cl:
  assumes
  A1: N ∈ FinPow(–D∈Pow(⋃T). D –is closed in″ T″)
  shows (⋃N) –is closed in″ T
proof -
  let C = –D∈Pow(⋃T). D –is closed in″ T″
  have 0∈C using Top˙3˙L2 by simp
  moreover have ∀A∈C. ∀B∈C. A∪B ∈ C
    using Top˙3˙L5 by auto
  moreover note A1
  ultimately have ⋃N ∈ C by (rule union˙two˙union˙fin)
  thus (⋃N) –is closed in″ T by simp
qed
```

Closure of a set is closed.

```
lemma (in topology0) cl˙is˙closed: assumes A ⊆ ⋃T
  shows cl(A) –is closed in″ T
  using assms 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 cl˙is˙closed Top˙3˙L5 Boundary˙def by auto
```

qed

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

```
lemma (in topology0) Top˙3˙L8: assumes A1: A ⊆ ⋃T
  shows A –is closed in″ T ⟷ cl(A) = A
proof
  assume A –is closed in″ T
  with A1 show cl(A) = A
    using Closure˙def ClosedCovers˙def by auto
next assume cl(A) = A
  then have ⋃T - A = ⋃T - cl(A) by simp
  with A1 show A –is closed in″ T using cl˙is˙closed IsClosed˙def
    by simp
qed
```

Complement of an open set is closed.

```
lemma (in topology0) Top˙3˙L9:
  assumes A1: A∈T
  shows (⋃T - A) –is closed in″ T
proof -
  from topSpaceAssum A1 have ⋃T - (⋃T - A) = A and ⋃T - A ⊆ ⋃T
    using IsATopology˙def by auto
  with A1 show (⋃T - A) –is closed in″ T using IsClosed˙def by simp
qed
```

A set is contained in its closure.

```
lemma (in topology0) cl˙contains˙set: assumes A ⊆ ⋃T shows A ⊆ cl(A)
  using assms Top˙3˙L1 ClosedCovers˙def Top˙3˙L3 Closure˙def by auto
```

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

```
lemma (in topology0) Top˙3˙L11: assumes A1: A ⊆ ⋃T
  shows
  cl(A) ⊆ ⋃T
  cl(⋃T - A) = ⋃T - int(A)
proof -
  from A1 show cl(A) ⊆ ⋃T using Top˙3˙L1 Closure˙def ClosedCovers˙def
    by auto
  from A1 have ⋃T - A ⊆ ⋃T - int(A) using Top˙2˙L1
    by auto
  moreover have I: ⋃T - int(A) ⊆ ⋃T   ⋃T - A ⊆ ⋃T by auto
  ultimately have cl(⋃T - A) ⊆ cl(⋃T - int(A))
    using top˙closure˙mono by simp
  moreover
  from I have (⋃T - int(A)) –is closed in″ T
    using Top˙2˙L2 Top˙3˙L9 by simp
  with I have cl((⋃T) - int(A)) = ⋃T - int(A)
    using Top˙3˙L8 by simp
```

```
  ultimately have cl(⋃T - A) ⊆ ⋃T - int(A) by simp
  moreover
  from I have ⋃T - A ⊆ cl(⋃T - A) using cl˙contains˙set by simp
  hence ⋃T - cl(⋃T - A) ⊆ A and ⋃T - A ⊆ ⋃T  by auto
  then have ⋃T - cl(⋃T - A) ⊆ int(A)
    using cl˙is˙closed IsClosed˙def Top˙2˙L5 by simp
  hence ⋃T - int(A) ⊆  cl(⋃T - A) by auto
  ultimately show cl(⋃T - A) = ⋃T - int(A) by auto
qed
```

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

```
lemma (in topology0) Top˙3˙L12: assumes A1: A ⊆ ⋃T
  shows ∂A = cl(A) - int(A)
proof -
  from A1 have ∂A = cl(A) ∩ (⋃T - int(A))
    using Boundary˙def Top˙3˙L11 by simp
  moreover from A1 have
    cl(A) ∩ (⋃T - int(A)) = cl(A) - int(A)
    using Top˙3˙L11 by blast
  ultimately show ∂A = cl(A) - int(A) by simp
qed
```

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

```
lemma (in topology0) Top˙3˙L13:
  assumes A1: B –is closed in″ T   A⊆B
  shows cl(A) ⊆ B
proof -
  from A1 have B ⊆ ⋃T using IsClosed˙def by simp
  with A1 show cl(A) ⊆ B using ClosedCovers˙def Closure˙def by auto
qed
```

If a set is disjoint with an open set, then we can close it and it will still be disjoint.

```
lemma (in topology0) disj˙open˙cl˙disj:
  assumes A1: A ⊆ ⋃T  V∈T and  A2: A∩V = 0
  shows cl(A) ∩ V = 0
proof -
  from assms have A ⊆ ⋃T - V by auto
  moreover from A1 have (⋃T - V) –is closed in″ T using Top˙3˙L9 by simp
  ultimately have cl(A) - (⋃T - V) = 0
    using Top˙3˙L13 by blast
  moreover from A1 have cl(A) ⊆ ⋃T using cl˙is˙closed IsClosed˙def by simp
  then have cl(A) -(⋃T - V) = cl(A) ∩ V by auto
  ultimately show thesis by simp
qed
```

A reformulation of disj˙open˙cl˙disj: If a point belongs to the closure of a set, then we can find a point from the set in any open neighboorhood of the

point.

lemma (in topology0) cl˙inter˙neigh:
  assumes A ⊆ ⋃T and U∈T and x ∈ cl(A) ∩ U
  shows A∩U ≠ 0 using assms disj˙open˙cl˙disj by auto

A reverse of cl˙inter˙neigh: if every open neiboorhood of a point has a nonempty intersection with a set, then that point belongs to the closure of the set.

lemma (in topology0) inter˙neigh˙cl:
  assumes A1: A ⊆ ⋃T and A2: x∈⋃T and A3: ∀U∈T. x∈U ⟶ U∩A ≠ 0
  shows x ∈ cl(A)
proof -
  { assume x ∉ cl(A)
    with A1 obtain D where D –is closed in″ T and A⊆D and x∉D
      using Top˙3˙L3 Closure˙def ClosedCovers˙def by auto
    let U = (⋃T) - D
    from A2 ‹D –is closed in″ T› ‹x∉D› ‹A⊆D› have U∈T x∈U and U∩A = 0
      unfolding IsClosed˙def by auto
    with A3 have False by auto
  } thus thesis by auto
qed

end

51 Topology 1

theory Topology˙ZF˙1 imports Topology˙ZF

begin

In this theory file we study separation axioms and the notion of base and subbase. Using the products of open sets as a subbase we define a natural topology on a product of two topological spaces.

51.1 Separation axioms.

Topological spaces cas be classified according to certain properties called "separation axioms". In this section we define what it means that a topological space is T_0, T_1 or T_2.

A topology on X is T_0 if for every pair of distinct points of X there is an open set that contains only one of them.

definition
  isT0 (˙ –is T_0″ [90] 91) where
  T –is T_0″ ≡ ∀ x y. ((x ∈ ⋃T ∧ y ∈ ⋃T ∧ x≠y) ⟶
  (∃U∈T. (x∈U ∧ y∉U) ∨ (y∈U ∧ x∉U)))

A topology is T_1 if for every such pair there exist an open set that contains the first point but not the second.

```
definition
  isT1 (˙ –is T₁″ [90] 91) where
  T –is T₁″ ≡ ∀ x y. ((x ∈ ⋃T ∧ y ∈ ⋃T ∧  x≠y) ⟶
  (∃U∈T. (x∈U ∧ y∉U)))
```

A topology is T_2 (Hausdorff) if for every pair of points there exist a pair of disjoint open sets each containing one of the points. This is an important class of topological spaces. In particular, metric spaces are Hausdorff.

```
definition
  isT2 (˙ –is T₂″ [90] 91) where
  T –is T₂″ ≡ ∀ x y. ((x ∈ ⋃T ∧ y ∈ ⋃T ∧  x≠y) ⟶
  (∃U∈T. ∃V∈T. x∈U ∧ y∈V ∧ U∩V=0))
```

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
```

51.2 Bases and subbases.

Sometimes it is convenient to talk about topologies in terms of their bases and subbases. These are certain collections of open sets that define the whole topology.

A base of topology is a collection of open sets such that every open set is a union of the sets from the base.

```
definition
  IsAbaseFor (infixl –is a base for″ 65) where
  B –is a base for″ T ≡ B⊆T ∧ T = –⋃A. A∈Pow(B)″
```

A subbase is a collection of open sets such that finite intersection of those sets form a base.

```
definition
  IsAsubBaseFor (infixl –is a subbase for″ 65) where
  B –is a subbase for″ T ≡
  B ⊆ T ∧ –⋂A. A ∈ FinPow(B)″ –is a base for″ T
```

Below we formulate a condition that we will prove to be necessary and sufficient for a collection B of open sets to form a base. It says that for any two sets U, V from the collection B we can find a point $x \in U \cap V$ with a neighboorhod from B contained in $U \cap V$.

```
definition
  SatisfiesBaseCondition (˙ –satisfies the base condition″ [50] 50)
  where
  B –satisfies the base condition″ ≡
  ∀U V. ((U∈B ∧ V∈B) ⟶ (∀x ∈ U∩V. ∃W∈B. x∈W ∧ W ⊆ U∩V))
```

A collection that is closed with respect to intersection satisfies the base condition.

```
lemma inter˙closed˙base: assumes ∀U∈B.(∀V∈B. U∩V ∈ B)
  shows  B –satisfies the base condition″
proof -
    { fix U V x assume U∈B and V∈B and x ∈ U∩V
      with assms have ∃W∈B. x∈W ∧ W ⊆ U∩V by blast
    } then show thesis using SatisfiesBaseCondition˙def by simp
qed
```

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

```
lemma Top˙1˙2˙L1: assumes B –is a base for″ T  and U∈T
  shows ∃A∈Pow(B). U = ⋃A
  using assms IsAbaseFor˙def by simp
```

Elements of base are open.

```
lemma base˙sets˙open:
  assumes B –is a base for″ T and U ∈ B
  shows U ∈ T
  using assms IsAbaseFor˙def by auto
```

A base defines topology uniquely.

```
lemma same˙base˙same˙top:
  assumes B –is a base for″ T and B –is a base for″ S
  shows T = S
  using assms IsAbaseFor˙def by simp
```

Every point from an open set has a neighboorhood from the base that is contained in the set.

```
lemma point˙open˙base˙neigh:
  assumes A1: B –is a base for″ T and A2: U∈T and A3: x∈U
  shows ∃V∈B. V⊆U ∧ x∈V
proof -
  from A1 A2 obtain A where A ∈ Pow(B) and U = ⋃A
    using Top˙1˙2˙L1 by blast
  with A3 obtain V where V∈A and x∈V by auto
  with ‹A ∈ Pow(B)› ‹U = ⋃A› show thesis by auto
qed
```

A criterion for a collection to be a base for a topology that is a slight reformulation of the definition. The only thing different that in the definition is that we assume only that every open set is a union of some sets from the base. The definition requires also the opposite inclusion that every union of the sets from the base is open, but that we can prove if we assume that T is a topology.

```
lemma is˙a˙base˙criterion: assumes A1: T –is a topology″
  and A2: B ⊆ T and A3: ∀V ∈ T. ∃A ∈ Pow(B). V = ⋃A
  shows B –is a base for″ T
proof -
  from A3 have T ⊆ –⋃A. A∈Pow(B)″ by auto
  moreover have –⋃A. A∈Pow(B)″ ⊆ T
  proof
    fix U assume U ∈ –⋃A. A∈Pow(B)″
    then obtain A where A ∈ Pow(B) and U = ⋃A
      by auto
    with ‹B ⊆ T› have A ∈ Pow(T) by auto
    with A1 ‹U = ⋃A› show U ∈ T
      unfolding IsATopology˙def by simp
  qed
```

ultimately have T = –⋃A. A∈Pow(B)″ by auto
with A2 show B –is a base for″ T
unfolding IsAbaseFor˙def by simp
qed

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

lemma Top˙1˙2˙L2:
assumes A1:∃T. T –is a topology″ ∧ B –is a base for″ T
and A2: V∈B W∈B
shows ∀ x ∈ V∩W. ∃U∈B. x∈U ∧ U ⊆ V ∩ W
proof -
from A1 obtain T where
D1: T –is a topology″ B –is a base for″ T
by auto
then have B ⊆ T using IsAbaseFor˙def by auto
with A2 have V∈T and W∈T using IsAbaseFor˙def by auto
with D1 have ∃A∈Pow(B). V∩W = ⋃A using IsATopology˙def Top˙1˙2˙L1
by auto
then obtain A where A ⊆ B and V ∩ W = ⋃A by auto
then show ∀ x ∈ V∩W. ∃U∈B. (x∈U ∧ U ⊆ V ∩ W) by auto
qed

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

lemma Top˙1˙2˙L3:
assumes A1: ∀x∈ V∩W . ∃U∈B. x∈U ∧ U ⊆ V∩W
shows V∩W ∈ –⋃A. A∈Pow(B)″
proof
let A = ⋃x∈V∩W. –U∈B. x∈U ∧ U ⊆ V∩W″
show A∈Pow(B) by auto
from A1 show V∩W = ⋃A by blast
qed

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

lemma Top˙1˙2˙L4:
assumes A1: U_1 ∈ –⋃A. A∈Pow(B)″ U_2 ∈ –⋃A. A∈Pow(B)″
and A2: B –satisfies the base condition″
shows ∃C. C ⊆ –⋃A. A∈Pow(B)″ ∧ U_1∩U_2 = ⋃C
proof -
from A1 A2 obtain A_1 A_2 where

```
    D1: A₁∈ Pow(B)  U₁ = ⋃A₁  A₂ ∈ Pow(B)  U₂ = ⋃A₂
    by auto
  let C = ⋃U∈A₁.–U∩V. V∈A₂″
  from D1 have (∀U∈A₁. U∈B) ∧ (∀V∈A₂. V∈B) by auto
  with A2 have C ⊆ –⋃A . A ∈ Pow(B)″
    using Top˙1˙2˙L3 SatisfiesBaseCondition˙def by auto
  moreover from D1 have U₁ ∩ U₂ = ⋃C by auto
  ultimately show thesis by auto
qed
```

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

```
theorem Top˙1˙2˙T1:
  assumes A1: B –satisfies the base condition″
  and A2: T = –⋃A. A∈Pow(B)″
  shows T –is a topology″ and B –is a base for″ T
proof -
  show T –is a topology″
  proof -
    have I: ∀C∈Pow(T). ⋃C ∈ T
    proof -
      { fix C assume A3: C ∈ Pow(T)
        let Q = ⋃ –⋃–A∈Pow(B). U = ⋃A″. U∈C″
        from A2 A3 have ∀U∈C. ∃A∈Pow(B). U = ⋃A by auto
        then have ⋃Q = ⋃C using ZF1˙1˙L10 by simp
        moreover from A2 have ⋃Q ∈ T by auto
        ultimately have ⋃C ∈ T by simp
      } thus ∀C∈Pow(T). ⋃C ∈ T by auto
    qed
    moreover have ∀U∈T. ∀  V∈T. U∩V ∈ T
    proof -
      { fix U V assume  U ∈ T  V ∈ T
        with A1 A2 have ∃C.(C ⊆ T ∧ U∩V = ⋃C)
        using Top˙1˙2˙L4 by simp
        then obtain C where C ⊆ T and  U∩V = ⋃C
          by auto
          with I have U∩V ∈ T by simp
      } then show ∀U∈T. ∀  V∈T. U∩V ∈ T by simp
    qed
    ultimately show T –is a topology″ using IsATopology˙def
      by simp
  qed
  from A2 have B⊆T by auto
  with A2 show B –is a base for″ T using IsAbaseFor˙def
    by simp
qed
```

The carrier of the base and topology are the same.

```
lemma Top˙1˙2˙L5: assumes B –is a base for″ T
```

shows $\bigcup$T = $\bigcup$B
using assms IsAbaseFor˙def by auto

If B is a base for T, then T is the smallest topology containing B.

lemma base˙smallest˙top:
assumes A1: B –is a base for″ T and A2: S –is a topology″ and A3: B⊆S
shows T⊆S
proof
fix U assume U∈T
with A1 obtain B_U where $B_U \subseteq$ B and U = $\bigcup B_U$ using IsAbaseFor˙def by auto
with A3 have $B_U \subseteq$ S by auto
with A2 ‹U = $\bigcup B_U$› show U∈S using IsATopology˙def by simp
qed

If B is a base for T and B is a topology, then $B = T$.

lemma base˙topology: assumes B –is a topology″ and B –is a base for″ T
shows B=T using assms base˙sets˙open base˙smallest˙top by blast

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

definition
ProductTopology(T,S) ≡ –$\bigcup$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 U_1 V_1 U_2 V_2 where
D1: U_1∈T V_1∈S A=U_1×V_1 U_2∈T V_2∈S B=U_2×V_2
using ProductCollection˙def by auto
let W = (U_1∩U_2) × (V_1∩V_2)
from A1 D1 have U_1∩U_2 ∈ T and V_1∩V_2 ∈ 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
```

Each point of a set open in the product topology has a neighborhood which is a cartesian product of open sets.

```
lemma prod˙top˙point˙neighb:
  assumes A1: T –is a topology″  S –is a topology″ and
  A2: U ∈ ProductTopology(T,S) and A3: x ∈ U
  shows ∃V W. V∈T ∧ W∈S ∧ V×W ⊆ U ∧ x ∈ V×W
proof -
  from A1 have
    ProductCollection(T,S) –is a base for″ ProductTopology(T,S)
    using Top˙1˙4˙T1 by simp
  with A2 A3 obtain Z where
    Z ∈ ProductCollection(T,S) and Z ⊆ U ∧ x∈Z
    using point˙open˙base˙neigh by blast
  then obtain V W where V ∈ T and W∈S and V×W ⊆ U ∧ x ∈ V×W
    using ProductCollection˙def by auto
  thus thesis by auto
qed
```

Products of open sets are open in the product topology.

```
lemma prod˙open˙open˙prod:
  assumes A1: T –is a topology″  S –is a topology″ and
  A2: U∈T V∈S
  shows U×V ∈ ProductTopology(T,S)
proof -
  from A1 have
    ProductCollection(T,S) –is a base for″ ProductTopology(T,S)
    using Top˙1˙4˙T1 by simp
  moreover from A2 have U×V ∈ ProductCollection(T,S)
    unfolding ProductCollection˙def by auto
```

```
  ultimately show U×V ∈ ProductTopology(T,S)
    using base˙sets˙open by simp
qed
```

Sets that are open in th product topology are contained in the product of the carrier.

```
lemma prod˙open˙type: assumes A1: T –is a topology″  S –is a topology″ and
  A2: V ∈ ProductTopology(T,S)
  shows V ⊆ ⋃T × ⋃S
proof -
  from A2 have V ⊆ ⋃ ProductTopology(T,S) by auto
  with A1 show thesis using Top˙1˙4˙T1 by simp
qed
```

Suppose we have subsets $A \subseteq X, B \subseteq Y$, where X, Y are topological spaces with topologies T, S. We can the consider relative topologies on T_A, S_B on sets A, B and the collection of cartesian products of sets open in T_A, S_B, (namely $\{U \times V : U \in T_A, V \in S_B\}$. The next lemma states that this collection is a base of the product topology on $X \times Y$ restricted to the product $A \times B$.

```
lemma prod˙restr˙base˙restr:
  assumes A1: T –is a topology″  S –is a topology″
  shows
  ProductCollection(T –restricted to″ A, S –restricted to″ B)
  –is a base for″ (ProductTopology(T,S) –restricted to″ A×B)
proof -
  let ℬ = ProductCollection(T –restricted to″ A, S –restricted to″ B)
  let τ = ProductTopology(T,S)
  from A1 have (τ –restricted to″ A×B) –is a topology″
    using Top˙1˙4˙T1 topology0˙def topology0.Top˙1˙L4
    by simp
  moreover have ℬ ⊆ (τ –restricted to″ A×B)
  proof
    fix U assume U ∈ ℬ
    then obtain U_A U_B where U = U_A × U_B and
      U_A ∈ (T –restricted to″ A) and U_B ∈ (S –restricted to″ B)
      using ProductCollection˙def by auto
    then obtain W_A W_B where
      W_A ∈ T  U_A = W_A ∩ A and W_B ∈ S  U_B = W_B ∩ B
      using RestrictedTo˙def by auto
    with ‹U = U_A × U_B› have U = W_A×W_B ∩ (A×B) by auto
    moreover from A1 ‹W_A ∈ T› and ‹W_B ∈ S› have W_A×W_B ∈ τ
      using prod˙open˙open˙prod by simp
    ultimately show U ∈ τ –restricted to″ A×B
      using RestrictedTo˙def by auto
  qed
  moreover have ∀U ∈ τ –restricted to″ A×B.
    ∃C ∈ Pow(ℬ). U = ⋃C
```

```
  proof
    fix U assume U ∈ τ –restricted to″ A×B
    then obtain W where W ∈ τ and U = W ∩ (A×B)
      using RestrictedTo˙def by auto
    from A1 ‹W ∈ τ› obtain A_W  where
      A_W ∈ Pow(ProductCollection(T,S)) and W = ⋃A_W
       using Top˙1˙4˙T1 IsAbaseFor˙def by auto
    let C = –V ∩ A×B. V ∈ A_W″
    have C ∈ Pow(𝓑) and U = ⋃C
    proof -
      { fix R assume R ∈ C
 then obtain V where V ∈ A_W and R = V ∩ A×B
   by auto
 with ‹A_W ∈ Pow(ProductCollection(T,S))› obtain V_T V_S where
   V_T ∈ T and V_S ∈ S and V = V_T × V_S
   using ProductCollection˙def by auto
 with ‹R = V ∩ A×B› have R ∈ 𝓑
   using ProductCollection˙def RestrictedTo˙def
   by auto
      } then show C ∈ Pow(𝓑) by auto
      from ‹U = W ∩ (A×B)› and ‹W = ⋃A_W›
      show U = ⋃C by auto
    qed
    thus ∃C ∈ Pow(𝓑). U = ⋃C by blast
  qed
  ultimately show thesis by (rule is˙a˙base˙criterion)
qed
```

We can commute taking restriction (relative topology) and product topology. The reason the two topologies are the same is that they have the same base.

```
lemma prod˙top˙restr˙comm:
  assumes A1: T –is a topology″  S –is a topology″
  shows
  ProductTopology(T –restricted to″ A,S –restricted to″ B) =
  ProductTopology(T,S) –restricted to″ (A×B)
proof -
  let 𝓑 = ProductCollection(T –restricted to″ A, S –restricted to″ B)
  from A1 have
    𝓑 –is a base for″ ProductTopology(T –restricted to″ A,S –restricted to″ B)
    using topology0˙def topology0.Top˙1˙L4 Top˙1˙4˙T1 by simp
  moreover from A1 have
    𝓑 –is a base for″ ProductTopology(T,S) –restricted to″ (A×B)
    using prod˙restr˙base˙restr by simp
  ultimately show thesis by (rule same˙base˙same˙top)
qed
```

Projection of a section of an open set is open.

```
lemma prod˙sec˙open1: assumes A1: T –is a topology″  S –is a topology″ and
  A2: V ∈ ProductTopology(T,S) and A3: x ∈ ⋃T
```

```
  shows –y ∈ ⋃S. ⟨x,y⟩ ∈ V″ ∈ S
proof -
  let A = –y ∈ ⋃S. ⟨x,y⟩ ∈ V″
  from A1 have topology0(S) using topology0˙def by simp
  moreover have ∀y∈A.∃W∈S. (y∈W ∧ W⊆A)
    proof
      fix y assume y ∈ A
      then have ⟨x,y⟩ ∈ V by simp
      with A1 A2 have ⟨x,y⟩ ∈ ⋃T × ⋃S using prod˙open˙type by blast
      hence x ∈ ⋃T and y ∈ ⋃S by auto
       from A1 A2 ‹⟨x,y⟩ ∈ V› have ∃U W. U∈T ∧ W∈S ∧ U×W ⊆ V ∧ ⟨x,y⟩ ∈
U×W
        by (rule prod˙top˙point˙neighb)
      then obtain U W where  U∈T W∈S U×W ⊆ V ⟨x,y⟩ ∈ U×W
        by auto
      with A1 A2 show ∃W∈S. (y∈W ∧ W⊆A) using prod˙open˙type section˙proj
        by auto
    qed
  ultimately show thesis by (rule topology0.open˙neigh˙open)
qed
```

Projection of a section of an open set is open. This is dual of prod˙sec˙open1 with a very similar proof.

```
lemma prod˙sec˙open2: assumes A1: T –is a topology″  S –is a topology″ and
  A2: V ∈ ProductTopology(T,S) and A3: y ∈ ⋃S
  shows –x ∈ ⋃T. ⟨x,y⟩ ∈ V″ ∈ T
proof -
  let A = –x ∈ ⋃T. ⟨x,y⟩ ∈ V″
  from A1 have topology0(T) using topology0˙def by simp
  moreover have ∀x∈A.∃W∈T. (x∈W ∧ W⊆A)
    proof
      fix x assume x ∈ A
      then have ⟨x,y⟩ ∈ V by simp
      with A1 A2 have ⟨x,y⟩ ∈ ⋃T × ⋃S using prod˙open˙type by blast
      hence x ∈ ⋃T and y ∈ ⋃S by auto
       from A1 A2 ‹⟨x,y⟩ ∈ V› have ∃U W. U∈T ∧ W∈S ∧ U×W ⊆ V ∧ ⟨x,y⟩ ∈
U×W
        by (rule prod˙top˙point˙neighb)
      then obtain U W where  U∈T W∈S U×W ⊆ V ⟨x,y⟩ ∈ U×W
        by auto
      with A1 A2 show ∃W∈T. (x∈W ∧ W⊆A) using prod˙open˙type section˙proj
        by auto
    qed
  ultimately show thesis by (rule topology0.open˙neigh˙open)
qed


end
```

52 Topology 1b

theory Topology˙ZF˙1b imports Topology˙ZF˙1

begin

One of the facts demonstrated in every class on General Topology is that in a T_2 (Hausdorff) topological space compact sets are closed. Formalizing the proof of this fact gave me an interesting insight into the role of the Axiom of Choice (AC) in many informal proofs.

A typical informal proof of this fact goes like this: we want to show that the complement of K is open. To do this, choose an arbitrary point $y \in K^c$. Since X is T_2, for every point $x \in K$ we can find an open set U_x such that $y \notin \overline{U_x}$. Obviously $\{U_x\}_{x\in K}$ covers K, so select a finite subcollection that covers K, and so on. I had never realized that such reasoning requires the Axiom of Choice. Namely, suppose we have a lemma that states "In T_2 spaces, if $x \neq y$, then there is an open set U such that $x \in U$ and $y \notin \overline{U}$" (like our lemma T2˙cl˙open˙sep below). This only states that the set of such open sets U is not empty. To get the collection $\{U_x\}_{x\in K}$ in this proof we have to select one such set among many for every $x \in K$ and this is where we use the Axiom of Choice. Probably in 99/100 cases when an informal calculus proof states something like $\forall\varepsilon\exists\delta_\varepsilon \cdots$ the proof uses AC. Most of the time the use of AC in such proofs can be avoided. This is also the case for the fact that in a T_2 space compact sets are closed.

52.1 Compact sets are closed - no need for AC

In this section we show that in a T_2 topological space compact sets are closed.

First we prove a lemma that in a T_2 space two points can be separated by the closure of an open set.

```
lemma (in topology0) T2˙cl˙open˙sep:
  assumes T –is T₂″  and x ∈ ⋃T  y ∈ ⋃T   x≠y
  shows ∃U∈T. (x∈U ∧ y ∉ cl(U))
proof -
  from assms have ∃U∈T. ∃V∈T. x∈U ∧ y∈V ∧ U∩V=0
    using isT2˙def by simp
  then obtain U V where U∈T  V∈T  x∈U  y∈V  U∩V=0
    by auto
  then have U∈T ∧ x∈U ∧ y∈ V ∧ cl(U) ∩ V = 0
    using  disj˙open˙cl˙disj by auto
  thus ∃U∈T. (x∈U ∧ y ∉ cl(U)) by auto
qed
```

AC-free proof that in a Hausdorff space compact sets are closed. To understand the notation recall that in Isabelle/ZF Pow(A) is the powerset (the

set of subsets) of A and FinPow(A) denotes the set of finite subsets of A in IsarMathLib.

```
theorem (in topology0) in˙t2˙compact˙is˙cl:
  assumes A1: T –is T₂″ and A2: K –is compact in″ T
  shows K –is closed in″ T
proof -
  let X = ⋃T
  have ∀y ∈ X - K. ∃U∈T. y∈U ∧ U ⊆ X - K
  proof -
    { fix y assume y ∈ X  y∉K
      have ∃U∈T. y∈U ∧ U ⊆ X - K
      proof -
 let B = ⋃x∈K. –V∈T. x∈V ∧ y ∉ cl(V)″
 have I: B ∈ Pow(T)  FinPow(B) ⊆ Pow(B)
   using FinPow˙def by auto
 from ‹K –is compact in″ T› ‹y ∈ X›  ‹y∉K› have
   ∀x∈K. x ∈ X ∧ y ∈ X ∧ x≠y
   using IsCompact˙def by auto
 with ‹T –is T₂″› have ∀x∈K. –V∈T. x∈V ∧ y ∉ cl(V)″ ≠ 0
   using T2˙cl˙open˙sep by auto
 hence K ⊆ ⋃B by blast
 with ‹K –is compact in″ T› I have
   ∃N ∈ FinPow(B). K ⊆ ⋃N
   using IsCompact˙def by auto
 then obtain N where N ∈ FinPow(B)  K ⊆ ⋃N
   by auto
 with I have N ⊆ B by auto
 hence ∀V∈N. V∈B by auto
 let M = –cl(V). V∈N″
 let C = –D ∈ Pow(X). D –is closed in″ T″
 from ‹N ∈ FinPow(B)› have ∀V∈B. cl(V) ∈ C  N ∈ FinPow(B)
   using cl˙is˙closed IsClosed˙def by auto
 then have M ∈ FinPow(C) by (rule fin˙image˙fin)
 then have X - ⋃M ∈ T using fin˙union˙cl˙is˙cl IsClosed˙def
   by simp
 moreover from ‹y ∈ X›  ‹y∉K›  ‹∀V∈N. V∈B› have
   y ∈ X - ⋃M by simp
 moreover have X - ⋃M ⊆ X - K
 proof -
   from ‹∀V∈N. V∈B› have ⋃N ⊆ ⋃M using cl˙contains˙set by auto
   with ‹K ⊆ ⋃N› show X - ⋃M ⊆ X - K by auto
 qed
 ultimately have ∃U. U∈T ∧ y ∈ U ∧ U ⊆ X - K
   by auto
 thus ∃U∈T. y∈U ∧ U ⊆ X - K by auto
      qed
    } thus ∀y ∈ X - K. ∃U∈T. y∈U ∧ U ⊆ X - K
      by auto
  qed
```

```
  with A2 show K –is closed in″ T
    using open˙neigh˙open IsCompact˙def IsClosed˙def by auto
qed
```

end

53 Topology 2

theory Topology˙ZF˙2 imports Topology˙ZF˙1 func1 Fol1

begin

This theory continues the series on general topology and covers the definition and basic properties of continuous functions. We also introduce the notion of homeomorphism an prove the pasting lemma.

53.1 Continuous functions.

In this section we define continuous functions and prove that certain conditions are equivalent to a function being continuous.

In standard math we say that a function is 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.

The notation f-(A) means the inverse image of (a set) A with respect to (a function) f.

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

A trivial example of a continuous function - identity is continuous.

```
lemma id˙cont: shows IsContinuous(τ,τ,id(⋃τ))
proof -
  { fix U assume U∈τ
    then have id(⋃τ)-(U) = U using vimage˙id˙same by auto
    with ‹U∈τ› have id(⋃τ)-(U) ∈ τ by simp
  } then show IsContinuous(τ,τ,id(⋃τ)) using IsContinuous˙def
    by simp
qed
```

We will work with a pair of topological spaces. The following locale sets up our context that consists of two topologies τ_1, τ_2 and a continuous function $f : X_1 \to X_2$, where X_i is defined as $\bigcup \tau_i$ for $i = 1, 2$. We also define notation cl_1(A) and cl_2(A) for closure of a set A in topologies τ_1 and τ_2, respectively.

locale two˙top˙spaces0 =

fixes τ_1
assumes tau1˙is˙top: τ_1 –is a topology″

fixes τ_2
assumes tau2˙is˙top: τ_2 –is a topology″

fixes X_1
defines X1˙def [simp]: $X_1 \equiv \bigcup \tau_1$

fixes X_2
defines X2˙def [simp]: $X_2 \equiv \bigcup \tau_2$

fixes f
assumes fmapAssum: f: $X_1 \to X_2$

fixes isContinuous (˙ –is continuous″ [50] 50)
defines isContinuous˙def [simp]: g –is continuous″ ≡ IsContinuous(τ_1,τ_2,g)

fixes cl_1
defines cl1˙def [simp]: cl_1(A) ≡ Closure(A,τ_1)

fixes cl_2
defines cl2˙def [simp]: cl_2(A) ≡ Closure(A,τ_2)

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

lemma (in two˙top˙spaces0) topol˙cntxs˙valid:
shows topology0(τ_1) and topology0(τ_2)
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″ τ_2
shows f-(D) –is closed in″ τ_1
proof -
from fmapAssum have f-(D) $\subseteq X_1$ using func1˙1˙L3 by simp
moreover from fmapAssum have f-(X_2 - D) = X_1 - f-(D)
using Pi˙iff function˙vimage˙Diff func1˙1˙L4 by auto
ultimately have X_1 - f-(X_2 - D) = f-(D) by auto
moreover from A1 A2 have (X_1 - f-(X_2 - D)) –is closed in″ τ_1
using IsClosed˙def IsContinuous˙def topol˙cntxs˙valid topology0.Top˙3˙L9
by simp
ultimately show f-(D) –is closed in″ τ_1 by simp
qed

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

```
lemma (in two˙top˙spaces0) Top˙ZF˙2˙1˙L2:
  assumes A1: ∀D. ((D –is closed in˝ τ2) ⟶ f-(D) –is closed in˝ τ1)
  and A2: A ⊆ X1
  shows f(cl1(A)) ⊆ cl2(f(A))
proof -
  from fmapAssum have f(A) ⊆ cl2(f(A))
    using func1˙1˙L6 topol˙cntxs˙valid topology0.cl˙contains˙set
    by simp
  with fmapAssum have f-(f(A)) ⊆ f-(cl2(f(A)))
    by auto
  moreover from fmapAssum A2 have A ⊆ f-(f(A))
    using func1˙1˙L9 by simp
  ultimately have A ⊆ f-(cl2(f(A))) by auto
  with fmapAssum A1 have f(cl1(A)) ⊆ f(f-(cl2(f(A))))
    using func1˙1˙L6 func1˙1˙L8 IsClosed˙def
      topol˙cntxs˙valid topology0.cl˙is˙closed topology0.Top˙3˙L13
    by simp
  moreover from fmapAssum have f(f-(cl2(f(A)))) ⊆ cl2(f(A))
    using fun˙is˙function function˙image˙vimage by simp
  ultimately show f(cl1(A)) ⊆ cl2(f(A))
    by auto
qed
```

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

```
lemma (in two˙top˙spaces0) Top˙ZF˙2˙1˙L3:
  assumes A1: ∀ A. ( A ⊆ X1 ⟶ f(cl1(A)) ⊆ cl2(f(A)))
  shows ∀B. ( B ⊆ X2 ⟶ cl1(f-(B)) ⊆ f-(cl2(B)) )
proof -
  { fix B assume B ⊆ X2
    from fmapAssum A1 have f(cl1(f-(B))) ⊆ cl2(f(f-(B)))
      using func1˙1˙L3 by simp
    moreover from fmapAssum ‹B ⊆ X2› have cl2(f(f-(B))) ⊆ cl2(B)
      using fun˙is˙function function˙image˙vimage func1˙1˙L6
 topol˙cntxs˙valid topology0.top˙closure˙mono
      by simp
    ultimately have f-(f(cl1(f-(B)))) ⊆ f-(cl2(B))
      using fmapAssum fun˙is˙function by auto
    moreover from fmapAssum ‹B ⊆ X2› have
      cl1(f-(B)) ⊆ f-(f(cl1(f-(B))))
      using func1˙1˙L3 func1˙1˙L9 IsClosed˙def
 topol˙cntxs˙valid topology0.cl˙is˙closed by simp
    ultimately have cl1(f-(B)) ⊆ f-(cl2(B)) by auto
  } then show thesis by simp
qed
```

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

ries of implications in lemmas Top˙ZF˙2˙1˙L1, Top˙ZF˙2˙1˙L2 and Top˙ZF˙2˙1˙L3 showing equivalence of four definitions of continuity.

```
lemma (in two˙top˙spaces0) Top˙ZF˙2˙1˙L4:
  assumes A1: ∀B. ( B ⊆ X₂ ⟶ cl₁(f-(B)) ⊆ f-(cl₂(B)) )
  shows f –is continuous″
proof -
  { fix U assume U ∈ τ₂
    then have (X₂ - U) –is closed in″ τ₂
      using topol˙cntxs˙valid topology0.Top˙3˙L9 by simp
    moreover have X₂ - U ⊆ ⋃τ₂ by auto
    ultimately have cl₂(X₂ - U) = X₂ - U
      using topol˙cntxs˙valid topology0.Top˙3˙L8 by simp
    moreover from A1 have cl₁(f-(X₂ - U)) ⊆ f-(cl₂(X₂ - U))
      by auto
    ultimately have cl₁(f-(X₂ - U)) ⊆ f-(X₂ - U) by simp
    moreover from fmapAssum have f-(X₂ - U) ⊆ cl₁(f-(X₂ - U))
      using func1˙1˙L3 topol˙cntxs˙valid topology0.cl˙contains˙set
      by simp
    ultimately have f-(X₂ - U) –is closed in″ τ₁
      using fmapAssum func1˙1˙L3 topol˙cntxs˙valid topology0.Top˙3˙L8
      by auto
    with fmapAssum have f-(U) ∈ τ₁
      using fun˙is˙function function˙vimage˙Diff func1˙1˙L4
 func1˙1˙L3 IsClosed˙def double˙complement by simp
  } then have ∀U∈τ₂. f-(U) ∈ τ₁ by simp
  then show thesis using IsContinuous˙def by simp
qed
```

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

```
lemma (in two˙top˙spaces0) Top˙ZF˙2˙1˙L5:
  assumes A1: B –is a base for″ τ₂ and A2: ∀U∈B. f-(U) ∈ τ₁
  shows f –is continuous″
proof -
  { fix V assume A3: V ∈ τ₂
    with A1 obtain A where A ⊆ B  V = ⋃A
      using IsAbaseFor˙def by auto
    with A2 have –f-(U). U∈A″ ⊆ τ₁ by auto
    with tau1˙is˙top have ⋃ –f-(U). U∈A″ ∈ τ₁
      using IsATopology˙def by simp
    moreover from ‹A ⊆ B› ‹V = ⋃A› have f-(V) = ⋃–f-(U). U∈A″
      by auto
    ultimately have f-(V) ∈  τ₁ by simp
  } then show f –is continuous″ using IsContinuous˙def
    by simp
qed
```

We can strenghten the previous lemma: it is sufficient to check if the inverse image of every set in a subbase is open. The proof is rather awkward, as

usual when we deal with general intersections. We have to keep track of the case when the collection is empty.

```
lemma (in two˙top˙spaces0) Top˙ZF˙2˙1˙L6:
  assumes A1: B –is a subbase for˝ τ2 and A2: ∀U∈B. f-(U) ∈ τ1
  shows f –is continuous˝
proof -
  let C = –⋂A. A ∈ FinPow(B)˝
  from A1 have C –is a base for˝ τ2
    using IsAsubBaseFor˙def by simp
  moreover have ∀U∈C. f-(U) ∈ τ1
  proof
    fix U assume U∈C
    { assume f-(U) = 0
      with tau1˙is˙top have f-(U) ∈ τ1
 using empty˙open by simp }
    moreover
    { assume f-(U) ≠ 0
      then have U≠0 by (rule func1˙1˙L13)
      moreover from ‹U∈C› obtain A where
 A ∈ FinPow(B) and U = ⋂A
 by auto
      ultimately have ⋂A≠0 by simp
      then have A≠0 by (rule inter˙nempty˙nempty)
      then have –f-(W). W∈A˝ ≠ 0 by simp
      moreover from A2 ‹A ∈ FinPow(B)› have –f-(W). W∈A˝ ∈ FinPow(τ1)
 by (rule fin˙image˙fin)
      ultimately have ⋂–f-(W). W∈A˝ ∈ τ1
 using topol˙cntxs˙valid topology0.fin˙inter˙open˙open by simp
      moreover
      from ‹A ∈ FinPow(B)› have A ⊆ B using FinPow˙def by simp
      with tau2˙is˙top A1 have A ⊆ Pow(X2)
 using IsAsubBaseFor˙def IsATopology˙def by auto
      with fmapAssum ‹A≠0› ‹U = ⋂A› have f-(U) = ⋂–f-(W). W∈A˝
 using func1˙1˙L12 by simp
      ultimately have f-(U) ∈ τ1 by simp }
    ultimately show f-(U) ∈ τ1 by blast
  qed
  ultimately show f –is continuous˝
    using Top˙ZF˙2˙1˙L5 by simp
qed
```

A dual of Top˙ZF˙2˙1˙L5: a function that maps base sets to open sets is open.

```
lemma (in two˙top˙spaces0) base˙image˙open:
  assumes A1: ℬ –is a base for˝ τ1 and A2: ∀B∈ℬ. f(B) ∈ τ2 and A3: U∈τ1
  shows f(U) ∈ τ2
proof -
  from A1 A3 obtain ℰ where ℰ ∈ Pow(ℬ) and U = ⋃ℰ using Top˙1˙2˙L1 by blast
  with A1 have f(U) = ⋃–f(E). E ∈ ℰ˝ using Top˙1˙2˙L5 fmapAssum image˙of˙Union
    by auto
```

moreover
from A2 ‹$\mathcal{E} \in$ Pow($\mathcal{B}$)› have –f(E). E $\in \mathcal{E}'' \in$ Pow(τ_2) by auto
then have $\bigcup$–f(E). E $\in \mathcal{E}'' \in \tau_2$ using tau2˙is˙top IsATopology˙def by simp
ultimately show thesis using tau2˙is˙top IsATopology˙def by auto
qed

A composition of two continuous functions is continuous.

lemma comp˙cont: assumes IsContinuous(T,S,f) and IsContinuous(S,R,g)
shows IsContinuous(T,R,g O f)
using assms IsContinuous˙def vimage˙comp by simp

A composition of three continuous functions is continuous.

lemma comp˙cont3:
assumes IsContinuous(T,S,f) and IsContinuous(S,R,g) and IsContinuous(R,P,h)
shows IsContinuous(T,P,h O g O f)
using assms IsContinuous˙def vimage˙comp by simp

53.2 Homeomorphisms

This section studies "homeomorphisms" - continous bijections whose inverses are also continuous. Notions that are preserved by (commute with) homeomorphisms are called "topological invariants".

Homeomorphism is a bijection that preserves open sets.

definition IsAhomeomorphism(T,S,f) $\equiv$
f $\in$ bij($\bigcup$T,$\bigcup$S) $\wedge$ IsContinuous(T,S,f) $\wedge$ IsContinuous(S,T,converse(f))

Inverse (converse) of a homeomorphism is a homeomorphism.

lemma homeo˙inv: assumes IsAhomeomorphism(T,S,f)
shows IsAhomeomorphism(S,T,converse(f))
using assms IsAhomeomorphism˙def bij˙converse˙bij bij˙converse˙converse
by auto

Homeomorphisms are open maps.

lemma homeo˙open: assumes IsAhomeomorphism(T,S,f) and U$\in$T
shows f(U) $\in$ S
using assms image˙converse IsAhomeomorphism˙def IsContinuous˙def by simp

A continuous bijection that is an open map is a homeomorphism.

lemma bij˙cont˙open˙homeo:
assumes f $\in$ bij($\bigcup$T,$\bigcup$S) and IsContinuous(T,S,f) and $\forall$U$\in$T. f(U) $\in$ S
shows IsAhomeomorphism(T,S,f)
using assms image˙converse IsAhomeomorphism˙def IsContinuous˙def by auto

A continuous bijection that maps base to open sets is a homeomorphism.

lemma (in two˙top˙spaces0) bij˙base˙open˙homeo:

assumes A1: f ∈ bij(X_1,X_2) and A2: $\mathcal{B}$ –is a base for″ τ_1 and A3: $\mathcal{C}$ –is a base for″ τ_2 and
A4: ∀U∈$\mathcal{C}$. f-(U) ∈ τ_1 and A5: ∀V∈$\mathcal{B}$. f(V) ∈ τ_2
shows IsAhomeomorphism(τ_1,τ_2,f)
using assms tau2˙is˙top tau1˙is˙top bij˙converse˙bij bij˙is˙fun two˙top˙spaces0˙def image˙converse two˙top˙spaces0.Top˙ZF˙2˙1˙L5 IsAhomeomorphism˙def by simp

A bijection that maps base to base is a homeomorphism.

lemma (in two˙top˙spaces0) bij˙base˙homeo:
assumes A1: f ∈ bij(X_1,X_2) and A2: $\mathcal{B}$ –is a base for″ τ_1 and
A3: –f(B). B∈$\mathcal{B}$″ –is a base for″ τ_2
shows IsAhomeomorphism(τ_1,τ_2,f)
proof -
note A1
moreover have f –is continuous″
proof -
{ fix C assume C ∈ –f(B). B∈$\mathcal{B}$″
then obtain B where B∈$\mathcal{B}$ and I: C = f(B) by auto
with A2 have B ⊆ X_1 using Top˙1˙2˙L5 by auto
with A1 A2 ‹B∈$\mathcal{B}$› I have f-(C) ∈ τ_1
using bij˙def inj˙vimage˙image base˙sets˙open by auto
} hence ∀C ∈ –f(B). B∈$\mathcal{B}$″. f-(C) ∈ τ_1 by auto
with A3 show thesis by (rule Top˙ZF˙2˙1˙L5)
qed
moreover
from A3 have ∀B∈$\mathcal{B}$. f(B) ∈ τ_2 using base˙sets˙open by auto
with A2 have ∀U∈τ_1. f(U) ∈ τ_2 using base˙image˙open by simp
ultimately show thesis using bij˙cont˙open˙homeo by simp
qed

Interior is a topological invariant.

theorem int˙top˙invariant: assumes A1: A⊆⋃T and A2: IsAhomeomorphism(T,S,f)
shows f(Interior(A,T)) = Interior(f(A),S)
proof -
let $\mathcal{A}$ = –U∈T. U⊆A″
have I: –f(U). U∈$\mathcal{A}$″ = –V∈S. V ⊆ f(A)″
proof
from A2 show –f(U). U∈$\mathcal{A}$″ ⊆ –V∈S. V ⊆ f(A)″
using homeo˙open by auto
{ fix V assume V ∈ –V∈S. V ⊆ f(A)″
hence V∈S and II: V ⊆ f(A) by auto
let U = f-(V)
from II have U ⊆ f-(f(A)) by auto
moreover from assms have f-(f(A)) = A
using IsAhomeomorphism˙def bij˙def inj˙vimage˙image by auto
moreover from A2 ‹V∈S› have U∈T
using IsAhomeomorphism˙def IsContinuous˙def by simp
moreover
from ‹V∈S› have V ⊆ ⋃S by auto

```
    with A2 have V = f(U)
      using IsAhomeomorphism˙def bij˙def surj˙image˙vimage by auto
    ultimately have V ∈ –f(U). U∈𝒜″ by auto
  } thus –V∈S. V ⊆ f(A)″ ⊆ –f(U). U∈𝒜″ by auto
 qed
 have f(Interior(A,T)) =  f(⋃𝒜) unfolding Interior˙def by simp
 also from A2 have . . . = ⋃–f(U). U∈𝒜″
   using IsAhomeomorphism˙def bij˙def inj˙def image˙of˙Union by auto
 also from I have . . . = Interior(f(A),S) unfolding Interior˙def by simp
 finally show thesis by simp
qed
```

53.3 Topologies induced by mappings

In this section we consider various ways a topology may be defined on a set that is the range (or the domain) of a function whose domain (or range) is a topological space.

A bijection from a topological space induces a topology on the range.

```
theorem bij˙induced˙top: assumes A1: T –is a topology″ and A2: f ∈ bij(⋃T,Y)
 shows
 –f(U). U∈T″ –is a topology″ and
 – –f(x).x∈U″. U∈T″ –is a topology″ and
 (⋃–f(U). U∈T″) = Y and
 IsAhomeomorphism(T, –f(U). U∈T″,f)
proof -
 from A2 have f ∈ inj(⋃T,Y) using bij˙def by simp
 then have f:⋃T→Y using inj˙def by simp
 let S = –f(U). U∈T″
 { fix M assume M ∈ Pow(S)
   let M_T = –f-(V). V∈M″
   have M_T ⊆ T
   proof
     fix W assume W∈M_T
     then obtain V where V∈M and I: W = f-(V) by auto
     with ‹M ∈ Pow(S)› have V∈S by auto
     then obtain U where U∈T and  V = f(U) by auto
     with I have W = f-(f(U)) by simp
     with ‹f ∈ inj(⋃T,Y)›  ‹U∈T› have W = U using inj˙vimage˙image by blast
     with ‹U∈T› show W∈T by simp
   qed
   with A1 have (⋃M_T) ∈ T using IsATopology˙def by simp
   hence f(⋃M_T) ∈  S by auto
   moreover have f(⋃M_T) = ⋃M
   proof -
     from ‹f:⋃T→Y› ‹M_T ⊆ T›  have f(⋃M_T) = ⋃–f(U). U∈M_T″
       using image˙of˙Union by auto
     moreover have –f(U). U∈M_T″ = M
     proof -
```

```
      from ‹f:⋃T→Y› have ∀U∈T. f(U) ⊆ Y using  func1˙1˙L6 by simp
      with ‹M ∈ Pow(S)› have M ⊆ Pow(Y) by auto
      with A2 show –f(U). U∈M_T″ = M using bij˙def surj˙subsets by auto
    qed
    ultimately show f(⋃M_T) = ⋃M by simp
  qed
  ultimately have ⋃M ∈ S by auto
} then have ∀M∈Pow(S). ⋃M ∈ S by auto
moreover
{ fix U V assume U∈S V∈S
  then obtain U_T V_T where U_T ∈ T   V_T ∈ T and
    I: U = f(U_T)  V = f(V_T)
    by auto
  with A1 have U_T∩V_T ∈ T using IsATopology˙def by simp
  hence f(U_T∩V_T) ∈ S by auto
  moreover have f(U_T∩V_T) = U∩V
  proof -
    from ‹U_T ∈ T›  ‹V_T ∈ T› have U_T ⊆ ⋃T  V_T ⊆ ⋃T
      using bij˙def by auto
    with ‹f ∈ inj(⋃T,Y)› I show f(U_T∩V_T) = U∩V using inj˙image˙inter
    by simp
  qed
  ultimately have U∩V ∈ S by simp
} then have ∀U∈S. ∀V∈S. U∩V ∈ S by auto
ultimately show S –is a topology″ using IsATopology˙def by simp
moreover from ‹f:⋃T→Y› have ∀U∈T. f(U) = –f(x).x∈U″
  using func˙imagedef by blast
ultimately show – –f(x).x∈U″. U∈T″ –is a topology″ by simp
show ⋃S =  Y
proof
  from ‹f:⋃T→Y› have ∀U∈T. f(U) ⊆ Y using func1˙1˙L6 by simp
  thus ⋃S ⊆ Y by auto
  from A1 have f(⋃T) ⊆ ⋃S using IsATopology˙def by auto
  with A2 show Y ⊆ ⋃S using bij˙def surj˙range˙image˙domain
    by auto
qed
show IsAhomeomorphism(T,S,f)
proof -
  from A2  ‹⋃S =  Y› have f ∈ bij(⋃T,⋃S) by simp
  moreover have IsContinuous(T,S,f)
  proof -
    { fix V assume V∈S
      then obtain U where U∈T and V = f(U) by auto
      hence U ⊆ ⋃T and f-(V) = f-(f(U))  by auto
      with ‹f ∈ inj(⋃T,Y)›  ‹U∈T› have f-(V) ∈ T  using inj˙vimage˙image
        by simp
    } then show IsContinuous(T,S,f) unfolding IsContinuous˙def by auto
  qed
  ultimately showIsAhomeomorphism(T,S,f) using bij˙cont˙open˙homeo
```

```
    by auto
  qed
qed
```

53.4 Partial functions and continuity

Suppose we have two topologies τ_1, τ_2 on sets $X_i = \bigcup \tau_i, i = 1, 2$. Consider some function $f : A \to X_2$, where $A \subseteq X_1$ (we will call such function "partial"). In such situation we have two natural possibilities for the pairs of topologies with respect to which this function may be continuous. One is obviously the original τ_1, τ_2 and in the second one the first element of the pair is the topology relative to the domain of the function: $\{A \cap U | U \in \tau_1\}$. These two possibilities are not exactly the same and the goal of this section is to explore the differences.

If a function is continuous, then its restriction is continous in relative topology.

```
lemma (in two˙top˙spaces0) restr˙cont:
  assumes A1: A ⊆ X1 and A2: f –is continuous″
  shows IsContinuous(τ1 –restricted to″ A, τ2,restrict(f,A))
proof -
  let g = restrict(f,A)
  { fix U assume U ∈ τ2
    with A2 have f-(U) ∈ τ1 using IsContinuous˙def by simp
    moreover from A1 have g-(U) = f-(U) ∩ A
      using fmapAssum func1˙2˙L1 by simp
    ultimately have g-(U) ∈ (τ1 –restricted to″ A)
      using RestrictedTo˙def by auto
  } then show thesis using IsContinuous˙def by simp
qed
```

If a function is continuous, then it is continuous when we restrict the topology on the range to the image of the domain.

```
lemma (in two˙top˙spaces0) restr˙image˙cont:
  assumes A1: f –is continuous″
  shows IsContinuous(τ1, τ2 –restricted to″ f(X1),f)
proof -
  have ∀U ∈ τ2 –restricted to″ f(X1). f-(U) ∈ τ1
  proof
    fix U assume U ∈ τ2 –restricted to″ f(X1)
    then obtain V where V ∈ τ2 and U = V ∩ f(X1)
      using RestrictedTo˙def by auto
    with A1 show  f-(U) ∈ τ1
      using fmapAssum inv˙im˙inter˙im IsContinuous˙def
      by simp
  qed
  then show thesis using IsContinuous˙def by simp
qed
```

A combination of restr˙cont and restr˙image˙cont.

```
lemma (in two˙top˙spaces0) restr˙restr˙image˙cont:
  assumes A1: A ⊆ X1 and A2: f –is continuous″ and
  A3: g = restrict(f,A) and
  A4: τ3 = τ1 –restricted to″ A
  shows IsContinuous(τ3, τ2 –restricted to″ g(A),g)
proof -
  from A1 A4 have ⋃τ3 = A
    using union˙restrict by auto
  have two˙top˙spaces0(τ3, τ2, g)
  proof -
    from A4 have
      τ3 –is a topology″ and τ2 –is a topology″
      using tau1˙is˙top tau2˙is˙top
 topology0˙def topology0.Top˙1˙L4 by auto
    moreover from A1 A3 ‹⋃τ3 = A› have g: ⋃τ3 → ⋃τ2
      using fmapAssum restrict˙type2 by simp
    ultimately show thesis using two˙top˙spaces0˙def
      by simp
  qed
  moreover from assms have IsContinuous(τ3, τ2, g)
    using restr˙cont by simp
  ultimately have IsContinuous(τ3, τ2 –restricted to″ g(⋃τ3),g)
    by (rule two˙top˙spaces0.restr˙image˙cont)
  moreover note ‹⋃τ3 = A›
  ultimately show thesis by simp
qed
```

We need a context similar to two˙top˙spaces0 but without the global function $f : X_1 \to X_2$.

```
locale two˙top˙spaces1 =

  fixes τ1
  assumes tau1˙is˙top: τ1 –is a topology″

  fixes τ2
  assumes tau2˙is˙top: τ2 –is a topology″

  fixes X1
  defines X1˙def [simp]: X1 ≡ ⋃τ1

  fixes X2
  defines X2˙def [simp]: X2 ≡ ⋃τ2
```

If a partial function $g : X_1 \supseteq A \to X_2$ is continuous with respect to (τ_1, τ_2), then A is open (in τ_1) and the function is continuous in the relative topology.

```
lemma (in two˙top˙spaces1) partial˙fun˙cont:
  assumes A1: g:A→X2 and A2: IsContinuous(τ1,τ2,g)
```

```
  shows A ∈ τ1 and IsContinuous(τ1 –restricted to″ A, τ2, g)
proof -
  from A2 have g-(X2) ∈ τ1
    using tau2˙is˙top IsATopology˙def IsContinuous˙def by simp
  with A1 show A ∈ τ1 using func1˙1˙L4 by simp
  { fix V assume V ∈ τ2
    with A2 have g-(V) ∈ τ1 using IsContinuous˙def by simp
    moreover
    from A1 have g-(V) ⊆ A using func1˙1˙L3 by simp
    hence g-(V) = A ∩ g-(V) by auto
    ultimately have g-(V) ∈ (τ1 –restricted to″ A)
      using RestrictedTo˙def by auto
  } then show IsContinuous(τ1 –restricted to″ A, τ2, g)
    using IsContinuous˙def by simp
qed
```

For partial function defined on open sets continuity in the whole and relative topologies are the same.

```
lemma (in two˙top˙spaces1) part˙fun˙on˙open˙cont:
  assumes A1: g:A→X2 and A2: A ∈ τ1
  shows IsContinuous(τ1,τ2,g) ⟷
        IsContinuous(τ1 –restricted to″ A, τ2, g)
proof
  assume IsContinuous(τ1,τ2,g)
  with A1 show IsContinuous(τ1 –restricted to″ A, τ2, g)
    using partial˙fun˙cont by simp
  next
    assume I: IsContinuous(τ1 –restricted to″ A, τ2, g)
    { fix V assume V ∈ τ2
      with I have g-(V) ∈ (τ1 –restricted to″ A)
        using IsContinuous˙def by simp
      then obtain W where W ∈ τ1 and g-(V) = A∩W
        using RestrictedTo˙def by auto
      with A2 have g-(V) ∈ τ1 using tau1˙is˙top IsATopology˙def
        by simp
    } then show IsContinuous(τ1,τ2,g) using IsContinuous˙def
      by simp
qed
```

53.5 Product topology and continuity

We start with three topological spaces $(\tau_1, X_1), (\tau_2, X_2)$ and (τ_3, X_3) and a function $f : X_1 \times X_2 \to X_3$. We will study the properties of f with respect to the product topology $\tau_1 \times \tau_2$ and τ_3. This situation is similar as in locale two˙top˙spaces0 but the first topological space is assumed to be a product of two topological spaces.

First we define a locale with three topological spaces.

locale prod˙top˙spaces0 =

fixes τ_1
assumes tau1˙is˙top: τ_1 –is a topology″

fixes τ_2
assumes tau2˙is˙top: τ_2 –is a topology″

fixes τ_3
assumes tau3˙is˙top: τ_3 –is a topology″

fixes X_1
defines X1˙def [simp]: $X_1 \equiv \bigcup \tau_1$

fixes X_2
defines X2˙def [simp]: $X_2 \equiv \bigcup \tau_2$

fixes X_3
defines X3˙def [simp]: $X_3 \equiv \bigcup \tau_3$

fixes η
defines eta˙def [simp]: $\eta \equiv$ ProductTopology(τ_1,τ_2)

Fixing the first variable in a two-variable continuous function results in a continuous function.

lemma (in prod˙top˙spaces0) fix˙1st˙var˙cont:
assumes f: $X_1 \times X_2 \rightarrow X_3$ and IsContinuous(η,τ_3,f)
and x$\in X_1$
shows IsContinuous(τ_2,τ_3,Fix1stVar(f,x))
using assms fix˙1st˙var˙vimage IsContinuous˙def tau1˙is˙top tau2˙is˙top
prod˙sec˙open1 by simp

Fixing the second variable in a two-variable continuous function results in a continuous function.

lemma (in prod˙top˙spaces0) fix˙2nd˙var˙cont:
assumes f: $X_1 \times X_2 \rightarrow X_3$ and IsContinuous(η,τ_3,f)
and y$\in X_2$
shows IsContinuous(τ_1,τ_3,Fix2ndVar(f,y))
using assms fix˙2nd˙var˙vimage IsContinuous˙def tau1˙is˙top tau2˙is˙top
prod˙sec˙open2 by simp

Having two constinuous mappings we can construct a third one on the cartesian product of the domains.

lemma cart˙prod˙cont:
assumes A1: τ_1 –is a topology″ τ_2 –is a topology″ and
A2: η_1 –is a topology″ η_2 –is a topology″ and
A3a: f_1:$\bigcup \tau_1 \rightarrow \bigcup \eta_1$ and A3b: f_2:$\bigcup \tau_2 \rightarrow \bigcup \eta_2$ and
A4: IsContinuous(τ_1,η_1,f_1) IsContinuous(τ_2,η_2,f_2) and

```
  A5: g = –⟨p,⟨f1(fst(p)),f2(snd(p))⟩⟩. p ∈ ⋃τ1×⋃τ2″
  shows IsContinuous(ProductTopology(τ1,τ2),ProductTopology(η1,η2),g)
proof -
  let τ = ProductTopology(τ1,τ2)
  let η = ProductTopology(η1,η2)
  let X1 = ⋃τ1
  let X2 = ⋃τ2
  let Y1 = ⋃η1
  let Y2 = ⋃η2
  let B = ProductCollection(η1,η2)
  from A1 A2 have τ –is a topology″ and η –is a topology″
    using Top˙1˙4˙T1 by auto
  moreover have g: X1×X2 → Y1×Y2
  proof -
    { fix p assume p ∈ X1×X2
      hence fst(p) ∈ X1 and snd(p) ∈ X2 by auto
      from A3a ‹fst(p) ∈ X1› have f1(fst(p)) ∈ Y1
        by (rule apply˙funtype)
      moreover from A3b ‹snd(p) ∈ X2› have f2(snd(p)) ∈ Y2
        by (rule apply˙funtype)
      ultimately have ⟨f1(fst(p)),f2(snd(p))⟩ ∈ ⋃η1×⋃η2 by auto
    } hence ∀p ∈ X1×X2. ⟨f1(fst(p)),f2(snd(p))⟩ ∈ Y1×Y2
      by simp
    with A5 show g: X1×X2 → Y1×Y2 using ZF˙fun˙from˙total
      by simp
  qed
  moreover from A1 A2 have ⋃τ = X1×X2 and ⋃η = Y1×Y2
    using Top˙1˙4˙T1 by auto
  ultimately have two˙top˙spaces0(τ,η,g) using two˙top˙spaces0˙def
    by simp
  moreover from A2 have B –is a base for″ η using Top˙1˙4˙T1
    by simp
  moreover have ∀U∈B. g-(U) ∈ τ
  proof
    fix U assume U∈B
    then obtain V W where V ∈ η1 W ∈ η2 and U = V×W
      using ProductCollection˙def by auto
    with A3a A3b A5 have g-(U) = f1-(V) × f2-(W)
      using cart˙prod˙fun˙vimage by simp
    moreover from A1 A4 ‹V ∈ η1› ‹W ∈ η2› have f1-(V) × f2-(W) ∈ τ
      using IsContinuous˙def prod˙open˙open˙prod by simp
    ultimately show g-(U) ∈ τ by simp
  qed
  ultimately show thesis using two˙top˙spaces0.Top˙ZF˙2˙1˙L5
    by simp
qed
```

A reformulation of the cart˙prod˙cont lemma above in slightly different notation.

```
theorem (in two˙top˙spaces0) product˙cont˙functions:
  assumes f:X₁→X₂ g:⋃τ₃→⋃τ₄
    IsContinuous(τ₁,τ₂,f) IsContinuous(τ₃,τ₄,g)
    τ₄–is a topology″ τ₃–is a topology″
 shows IsContinuous(ProductTopology(τ₁,τ₃),ProductTopology(τ₂,τ₄),–⟨⟨x,y⟩,⟨fx,gy⟩⟩.
⟨x,y⟩∈X₁×⋃τ₃″)
proof -
  have –⟨⟨x,y⟩,⟨fx,gy⟩⟩. ⟨x,y⟩∈X₁×⋃τ₃″ = –⟨p,⟨f(fst(p)),g(snd(p))⟩⟩. p ∈ X₁×⋃τ₃″
    by force
  with tau1˙is˙top tau2˙is˙top assms show thesis using cart˙prod˙cont by simp
qed
```

A special case of cart˙prod˙cont when the function acting on the second axis is the identity.

```
lemma cart˙prod˙cont1:
 assumes A1: τ₁ –is a topology″ and A1a: τ₂ –is a topology″ and
  A2: η₁ –is a topology″  and
  A3: f₁:⋃τ₁→⋃η₁ and A4: IsContinuous(τ₁,η₁,f₁) and
  A5: g = –⟨p, ⟨f₁(fst(p)),snd(p)⟩⟩. p ∈ ⋃τ₁×⋃τ₂″
  shows IsContinuous(ProductTopology(τ₁,τ₂),ProductTopology(η₁,τ₂),g)
proof -
  let f₂ = id(⋃τ₂)
  have ∀x∈⋃τ₂. f₂(x) = x using id˙conv by blast
  hence I: ∀p ∈ ⋃τ₁×⋃τ₂. snd(p) = f₂(snd(p)) by simp
  note A1 A1a A2 A1a A3
  moreover have f₂:⋃τ₂→⋃τ₂  using id˙type by simp
  moreover note A4
  moreover have IsContinuous(τ₂,τ₂,f₂) using id˙cont by simp
  moreover have g = –⟨p, ⟨f₁(fst(p)),f₂(snd(p))⟩ ⟩. p ∈ ⋃τ₁×⋃τ₂″
  proof
    from A5 I show  g ⊆ –⟨p, ⟨f₁(fst(p)),f₂(snd(p))⟩⟩. p ∈ ⋃τ₁×⋃τ₂″
      by auto
    from A5 I show –⟨p, ⟨f₁(fst(p)),f₂(snd(p))⟩⟩. p ∈ ⋃τ₁×⋃τ₂″ ⊆ g
      by auto
  qed
  ultimately show thesis by (rule cart˙prod˙cont)
qed
```

53.6 Pasting lemma

The classical pasting lemma states that if U_1, U_2 are both open (or closed) and a function is continuous when restricted to both U_1 and U_2 then it is continuous when restricted to $U_1 \cup U_2$. In this section we prove a generalization statement stating that the set $\{U \in \tau_1 | f|_U$ is continuous $\}$ is a topology.

A typical statement of the pasting lemma uses the notion of a function restricted to a set being continuous without specifying the topologies with

respect to which this continuity holds. In two˙top˙spaces0 context the notation g –is continuous″ means continuity wth respect to topologies τ_1, τ_2. The next lemma is a special case of partial˙fun˙cont and states that if for some set $A \subseteq X_1 = \bigcup \tau_1$ the function $f|_A$ is continuous (with respect to (τ_1, τ_2)), then A has to be open. This clears up terminology and indicates why we need to pay attention to the issue of which topologies we talk about when we say that the restricted (to some closed set for example) function is continuos.

```
lemma (in two˙top˙spaces0) restriction˙continuous1:
  assumes A1: A ⊆ X1 and A2: restrict(f,A) –is continuous″
  shows A ∈ τ1
proof -
  from assms have two˙top˙spaces1(τ1,τ2) and
    restrict(f,A):A→X2 and restrict(f,A) –is continuous″
    using tau1˙is˙top tau2˙is˙top two˙top˙spaces1˙def fmapAssum restrict˙fun
      by auto
  then show thesis using two˙top˙spaces1.partial˙fun˙cont by simp
qed
```

If a fuction is continuous on each set of a collection of open sets, then it is continuous on the union of them. We could use continuity with respect to the relative topology here, but we know that on open sets this is the same as the original topology.

```
lemma (in two˙top˙spaces0) pasting˙lemma1:
  assumes A1: M ⊆ τ1 and A2: ∀U∈M. restrict(f,U)  –is continuous″
  shows restrict(f,⋃M) –is continuous″
proof -
  { fix V assume V∈τ2
    from A1 have ⋃M ⊆ X1 by auto
    then have restrict(f,⋃M)-(V) = f-(V) ∩ (⋃M)
      using func1˙2˙L1 fmapAssum by simp
    also have ... = ⋃ –f-(V) ∩ U. U∈M″ by auto
    finally have restrict(f,⋃M)-(V) = ⋃ –f-(V) ∩ U. U∈M″ by simp
    moreover
    have –f-(V) ∩ U. U∈M″ ∈ Pow(τ1)
    proof -
      { fix W assume W ∈ –f-(V) ∩ U. U∈M″
        then obtain U where U∈M and I: W = f-(V) ∩ U by auto
        with A2 have restrict(f,U) –is continuous″ by simp
        with ‹V∈τ2› have restrict(f,U)-(V) ∈ τ1
          using IsContinuous˙def by simp
        moreover from ‹⋃M ⊆ X1› and ‹U∈M›
        have restrict(f,U)-(V) = f-(V) ∩ U
          using fmapAssum func1˙2˙L1 by blast
        ultimately have f-(V) ∩ U ∈ τ1 by simp
        with I have W ∈ τ1 by simp
      } then show thesis by auto
    qed
    then have  ⋃–f-(V) ∩ U. U∈M″ ∈ τ1
```

```
      using tau1˙is˙top IsATopology˙def by auto
    ultimately have restrict(f,⋃M)-(V) ∈ τ1
      by simp
  } then show thesis using IsContinuous˙def by simp
qed
```

If a function is continuous on two sets, then it is continuous on intersection.

```
lemma (in two˙top˙spaces0) cont˙inter˙cont:
  assumes A1: A ⊆ X1 B ⊆ X1 and
  A2: restrict(f,A) –is continuous″ restrict(f,B) –is continuous″
  shows restrict(f,A∩B) –is continuous″
proof -
  { fix V assume V∈τ2
    with assms have
      restrict(f,A)-(V) = f-(V) ∩ A  restrict(f,B)-(V) = f-(V) ∩ B and
      restrict(f,A)-(V) ∈ τ1 and restrict(f,B)-(V) ∈ τ1
        using func1˙2˙L1 fmapAssum IsContinuous˙def by auto
    then have (restrict(f,A)-(V)) ∩ (restrict(f,B)-(V)) = f-(V) ∩ (A∩B)
      by auto
    moreover
    from A2 ‹V∈τ2› have
      restrict(f,A)-(V) ∈ τ1 and restrict(f,B)-(V) ∈ τ1
      using IsContinuous˙def by auto
    then have (restrict(f,A)-(V)) ∩ (restrict(f,B)-(V)) ∈ τ1
      using tau1˙is˙top IsATopology˙def by simp
    moreover
    from A1 have (A∩B) ⊆ X1 by auto
    then have restrict(f,A∩B)-(V) = f-(V) ∩ (A∩B)
      using func1˙2˙L1 fmapAssum by simp
  ultimately have restrict(f,A∩B)-(V) ∈ τ1 by simp
  } then show thesis using  IsContinuous˙def by auto
qed
```

The collection of open sets U such that f restricted to U is continuous, is a topology.

```
theorem (in two˙top˙spaces0) pasting˙theorem:
  shows –U ∈ τ1. restrict(f,U) –is continuous″″ –is a topology″
proof -
  let T = –U ∈ τ1. restrict(f,U) –is continuous″″
  have ∀M∈Pow(T). ⋃M ∈ T
  proof
    fix M assume M ∈ Pow(T)
    then have restrict(f,⋃M) –is continuous″
      using pasting˙lemma1 by auto
    with ‹M ∈ Pow(T)› show ⋃M ∈ T
      using tau1˙is˙top IsATopology˙def by auto
  qed
  moreover have ∀U∈T.∀V∈T. U∩V ∈ T
    using cont˙inter˙cont tau1˙is˙top IsATopology˙def by auto
```

ultimately show thesis using IsATopology˙def by simp
qed

0 is continuous.

corollary (in two˙top˙spaces0) zero˙continuous: shows 0 –is continuous″
proof -
let T = –U ∈ τ_1. restrict(f,U) –is continuous″″
have T –is a topology″ by (rule pasting˙theorem)
then have 0∈T by (rule empty˙open)
hence restrict(f,0) –is continuous″ by simp
moreover have restrict(f,0) = 0 by simp
ultimately show thesis by simp
qed

end

54 Topology 3

theory Topology˙ZF˙3 imports Topology˙ZF˙2 FiniteSeq˙ZF

begin

Topology˙ZF˙1 theory describes how we can define a topology on a product of two topological spaces. One way to generalize that is to construct topology for a cartesian product of n topological spaces. The cartesian product approach is somewhat inconvenient though. Another way to approach product topology on X^n is to model cartesian product as sets of sequences (of length n) of elements of X. This means that having a topology on X we want to define a toplogy on the space $n \to X$, where n is a natural number (recall that $n = \{0, 1, ..., n-1\}$ in ZF). However, this in turn can be done more generally by defining a topology on any function space $I \to X$, where I is any set of indices. This is what we do in this theory.

54.1 The base of the product topology

In this section we define the base of the product topology.

Suppose $\mathcal{X} = I \to \bigcup T$ is a space of functions from some index set I to the carrier of a topology T. Then take a finite collection of open sets $W : N \to T$ indexed by $N \subseteq I$. We can define a subset of $\mathcal{X}$ that models the cartesian product of W.

definition
FinProd($\mathcal{X}$,W) ≡ –x∈$\mathcal{X}$. ∀ i∈domain(W). x(i) ∈ W(i)″

Now we define the base of the product topology as the collection of all finite products (in the sense defined above) of open sets.

```
definition
  ProductTopBase(I,T) ≡ ⋃N∈FinPow(I).–FinProd(I→⋃T,W). W∈N→T″
```

Finally, we define the product topology on sequences. We use the "Seq" prefix although the definition is good for any index sets, not only natural numbers.

```
definition
  SeqProductTopology(I,T) ≡ –⋃B. B ∈ Pow(ProductTopBase(I,T))″
```

Product topology base is closed with respect to intersections.

```
lemma prod˙top˙base˙inter:
  assumes A1: T –is a topology″ and
  A2: U ∈ ProductTopBase(I,T)  V ∈ ProductTopBase(I,T)
  shows U∩V ∈ ProductTopBase(I,T)
proof -
  let 𝒳 = I→⋃T
  from A2 obtain N1  W1 N2 W2 where
    I: N1 ∈ FinPow(I)  W1∈N1→T  U = FinProd(𝒳,W1) and
    II: N2 ∈ FinPow(I)  W2∈N2→T  V = FinProd(𝒳,W2)
    using ProductTopBase˙def by auto
  let N3 = N1 ∪ N2
  let W3 = –⟨i,if i ∈ N1-N2 then W1(i)
       else if i ∈ N2-N1 then  W2(i)
       else (W1(i)) ∩ (W2(i))⟩. i ∈ N3″
  from A1 I II have ∀i ∈ N1 ∩ N2.  (W1(i) ∩ W2(i)) ∈ T
     using apply˙funtype IsATopology˙def by auto
  moreover from I II have ∀i∈N1-N2. W1(i) ∈ T and ∀i∈N2-N1. W2(i) ∈ T
     using apply˙funtype by auto
  ultimately have  W3:N3→T by (rule fun˙union˙overlap)
  with I II have FinProd(𝒳,W3) ∈ ProductTopBase(I,T) using union˙finpow Pro-
ductTopBase˙def
    by auto
  moreover have U∩V = FinProd(𝒳,W3)
  proof
    { fix x assume x∈U and x∈V
        with ‹U = FinProd(𝒳,W1)›  ‹W1∈N1→T› and  ‹V = FinProd(𝒳,W2)›
‹W2∈N2→T›
      have x∈𝒳 and ∀i∈N1. x(i) ∈ W1(i) and ∀i∈N2. x(i) ∈ W2(i)
        using func1˙1˙L1 FinProd˙def by auto
      with ‹W3:N3→T› ‹x∈𝒳› have x ∈ FinProd(𝒳,W3)
        using ZF˙fun˙from˙tot˙val func1˙1˙L1 FinProd˙def by auto
    } thus U∩V ⊆ FinProd(𝒳,W3) by auto
    { fix x assume x ∈ FinProd(𝒳,W3)
      with ‹W3:N3→T› have x:I→⋃T and III: ∀i∈N3. x(i) ∈ W3(i)
        using FinProd˙def func1˙1˙L1 by auto
     { fix i assume i∈N1
       with ‹W3:N3→T› have W3(i) ⊆ W1(i) using ZF˙fun˙from˙tot˙val by auto
       with III ‹i∈N1› have x(i) ∈ W1(i) by auto
     } with ‹W1∈N1→T› ‹x:I→⋃T› ‹U = FinProd(𝒳,W1)›
```

```
      have x ∈ U using func1˙1˙L1 FinProd˙def by auto
      moreover
      { fix i assume i∈N2
        with ‹W3:N3→T› have W3(i) ⊆ W2(i) using ZF˙fun˙from˙tot˙val by auto
        with III ‹i∈N2› have x(i) ∈ W2(i) by auto
      } with ‹W2∈N2→T› ‹x:I→⋃T› ‹V = FinProd(𝒳,W2)› have x∈V
          using func1˙1˙L1 FinProd˙def by auto
      ultimately have x ∈ U∩V by simp
    } thus FinProd(𝒳,W3) ⊆ U∩V by auto
  qed
    ultimately show thesis by simp
qed
```

In the next theorem we show the collection of sets defined above as ProductTopBase($\mathcal{X}$,T) satisfies the base condition. This is a condition, defined in Topology˙ZF˙1 that allows to claim that this collection is a base for some topology.

```
theorem prod˙top˙base˙is˙base: assumes T –is a topology″
  shows ProductTopBase(I,T) –satisfies the base condition″
  using assms prod˙top˙base˙inter inter˙closed˙base by simp
```

The (sequence) product topology is indeed a topology on the space of sequences. In the proof we are using the fact that $(\emptyset \to X) = \{\emptyset\}$.

```
theorem seq˙prod˙top˙is˙top:  assumes T –is a topology″
  shows
  SeqProductTopology(I,T) –is a topology″ and
  ProductTopBase(I,T) –is a base for″ SeqProductTopology(I,T) and
  ⋃SeqProductTopology(I,T) = (I→⋃T)
proof -
  from assms show SeqProductTopology(I,T) –is a topology″ and
    I: ProductTopBase(I,T) –is a base for″ SeqProductTopology(I,T)
      using prod˙top˙base˙is˙base SeqProductTopology˙def Top˙1˙2˙T1
        by auto
  from I have ⋃SeqProductTopology(I,T) = ⋃ProductTopBase(I,T)
    using Top˙1˙2˙L5 by simp
  also have ⋃ProductTopBase(I,T) = (I→⋃T)
  proof
   show ⋃ProductTopBase(I,T) ⊆ (I→⋃T) using ProductTopBase˙def FinProd˙def
      by auto
    have 0 ∈ FinPow(I) using empty˙in˙finpow by simp
   hence –FinProd(I→⋃T,W). W∈0→T″ ⊆ (⋃N∈FinPow(I).–FinProd(I→⋃T,W).
W∈N→T″)
      by blast
    then show (I→⋃T) ⊆ ⋃ProductTopBase(I,T) using ProductTopBase˙def Fin-
Prod˙def
      by auto
  qed
  finally show ⋃SeqProductTopology(I,T) = (I→⋃T) by simp
qed
```

54.2 Finite product of topologies

As a special case of the space of functions $I \to X$ we can consider space of lists of elements of X, i.e. space $n \to X$, where n is a natural number (recall that in ZF set theory $n = \{0, 1, ..., n-1\}$). Such spaces model finite cartesian products X^n but are easier to deal with in formalized way (than the said products). This section discusses natural topology defined on $n \to X$ where X is a topological space.

When the index set is finite, the definition of ProductTopBase(I,T) can be simplifed.

```
lemma fin˙prod˙def˙nat: assumes A1: n∈nat and A2: T –is a topology″
  shows ProductTopBase(n,T) = –FinProd(n→⋃T,W). W∈n→T″
proof
  from A1 have n ∈ FinPow(n) using nat˙finpow˙nat fin˙finpow˙self by auto
  then show –FinProd(n→⋃T,W). W∈n→T″ ⊆ ProductTopBase(n,T) using Pro-
ductTopBase˙def
    by auto
  { fix B assume B ∈ ProductTopBase(n,T)
      then obtain N W where  N ∈ FinPow(n) and W ∈ N→T and B = Fin-
Prod(n→⋃T,W)
      using ProductTopBase˙def by auto
    let W_n = –⟨i,if i∈N then W(i) else ⋃T⟩. i∈n″
    from A2 ‹N ∈ FinPow(n)› ‹W∈N→T› have ∀i∈n. (if i∈N then W(i) else ⋃T)
∈ T
      using apply˙funtype FinPow˙def IsATopology˙def by auto
    then have W_n:n→T by (rule ZF˙fun˙from˙total)
    moreover have B = FinProd(n→⋃T,W_n)
    proof
      { fix x assume x∈B
       with ‹B = FinProd(n→⋃T,W)› have x ∈ n→⋃T using FinProd˙def by simp
        moreover have ∀i∈domain(W_n). x(i) ∈ W_n(i)
        proof
          fix i assume i ∈ domain(W_n)
          with ‹W_n:n→T› have i∈n using func1˙1˙L1 by simp
          with ‹x:n→⋃T› have x(i) ∈ ⋃T using apply˙funtype by blast
          with ‹x∈B› ‹B = FinProd(n→⋃T,W)› ‹W ∈ N→T› ‹W_n:n→T› ‹i∈n›
          show x(i) ∈ W_n(i) using func1˙1˙L1 FinProd˙def ZF˙fun˙from˙tot˙val
            by simp
        qed
        ultimately have x ∈ FinProd(n→⋃T,W_n) using FinProd˙def by simp
      } thus B ⊆ FinProd(n→⋃T,W_n) by auto
      next
      { fix x assume x ∈ FinProd(n→⋃T,W_n)
        then have x ∈ n→⋃T and ∀i∈domain(W_n). x(i) ∈ W_n(i)
          using  FinProd˙def by auto
        with ‹W_n:n→T› and ‹N ∈ FinPow(n)› have ∀i∈N. x(i) ∈ W_n(i)
          using func1˙1˙L1 FinPow˙def by auto
        moreover from ‹W_n:n→T› and ‹N ∈ FinPow(n)›
```

```
      have ∀i∈N. W_n(i) = W(i)
        using ZF˙fun˙from˙tot˙val FinPow˙def by auto
      ultimately have ∀i∈N. x(i) ∈ W(i) by simp
      with ‹W ∈ N→T› ‹x ∈ n→⋃T› ‹B = FinProd(n→⋃T,W)› have x∈B
        using func1˙1˙L1 FinProd˙def by simp
    } thus FinProd(n→⋃T,W_n) ⊆ B by auto
  qed
    ultimately have B ∈ –FinProd(n→⋃T,W). W∈n→T″ by auto
  } thus ProductTopBase(n,T) ⊆ –FinProd(n→⋃T,W). W∈n→T″ by auto
qed
```

A technical lemma providing a formula for finite product on one topological space.

```
lemma single˙top˙prod: assumes A1: W:1→τ
  shows FinProd(1→⋃τ,W) = – –⟨0,y⟩″. y ∈ W(0)″
proof -
  have 1 = –0″ by auto
  from A1 have domain(W) = –0″ using func1˙1˙L1 by auto
  then have FinProd(1→⋃τ,W) = –x ∈ 1→⋃τ. x(0) ∈ W(0)″
    using FinProd˙def by simp
  also have –x ∈ 1→⋃τ. x(0) ∈ W(0)″ = – –⟨0,y⟩″. y ∈ W(0)″
  proof
    from ‹1 = –0″› show –x ∈ 1→⋃τ. x(0) ∈ W(0)″ ⊆ – –⟨0,y⟩″. y ∈ W(0)″
      using func˙singleton˙pair by auto
    { fix x assume x ∈ – –⟨0,y⟩″. y ∈ W(0)″
      then obtain y where x = –⟨0,y⟩″ and II: y ∈ W(0) by auto
      with A1 have y ∈ ⋃τ using apply˙funtype by auto
      with ‹x = –⟨0,y⟩″› ‹1 = –0″› have x:1→⋃τ using pair˙func˙singleton
        by auto
      with ‹x = –⟨0,y⟩″› II have x ∈ –x ∈ 1→⋃τ. x(0) ∈ W(0)″
        using pair˙val by simp
    } thus – –⟨0,y⟩″. y ∈ W(0)″ ⊆ –x ∈ 1→⋃τ. x(0) ∈ W(0)″ by auto
  qed
  finally show thesis by simp
qed
```

Intuitively, the topological space of singleton lists valued in X is the same as X. However, each element of this space is a list of length one, i.e a set consisting of a pair $\langle 0, x\rangle$ where x is an element of X. The next lemma provides a formula for the product topology in the corner case when we have only one factor and shows that the product topology of one space is essentially the same as the space.

```
lemma singleton˙prod˙top: assumes A1: τ –is a topology″
  shows
    SeqProductTopology(1,τ) = – – –⟨0,y⟩″. y∈U ″. U∈τ″ and
    IsAhomeomorphism(τ,SeqProductTopology(1,τ),–⟨y,–⟨0,y⟩″⟩.y ∈ ⋃τ″)
proof -
  have –0″ = 1 by auto
```

```
  let b = –⟨y,–⟨0,y⟩″⟩.y ∈ ⋃τ″
  have b ∈ bij(⋃τ,1→⋃τ) using list˙singleton˙bij by blast
   with A1 have –b(U). U∈τ″ –is a topology″ and IsAhomeomorphism(τ, –b(U).
U∈τ″,b)
    using bij˙induced˙top by auto
  moreover have ∀U∈τ. b(U) = – –⟨0,y⟩″. y∈U ″
  proof
    fix U assume U∈τ
    from ‹b ∈ bij(⋃τ,1→⋃τ)› have b:⋃τ→(1→⋃τ) using bij˙def inj˙def
      by simp
    { fix y assume y ∈ ⋃τ
      with ‹b:⋃τ→(1→⋃τ)› have b(y) = –⟨0,y⟩″ using ZF˙fun˙from˙tot˙val
        by simp
    } hence ∀y ∈ ⋃τ. b(y) = –⟨0,y⟩″ by auto
    with ‹U∈τ› ‹b:⋃τ→(1→⋃τ)› show b(U) = – –⟨0,y⟩″. y∈U ″
      using func˙imagedef by auto
  qed
  moreover have ProductTopBase(1,τ) = – – –⟨0,y⟩″. y∈U ″. U∈τ″
  proof
    { fix V assume V ∈ ProductTopBase(1,τ)
      with A1 obtain W where W:1→τ and V = FinProd(1→⋃τ,W)
        using fin˙prod˙def˙nat by auto
      then have V ∈ – – –⟨0,y⟩″. y∈U ″. U∈τ″ using apply˙funtype single˙top˙prod
        by auto
    } thus ProductTopBase(1,τ) ⊆ – – –⟨0,y⟩″. y∈U ″. U∈τ″ by auto
  { fix V assume V ∈ – – –⟨0,y⟩″. y∈U ″. U∈τ″
    then obtain U where U∈τ and V = – –⟨0,y⟩″. y∈U ″ by auto
    let W = –⟨0,U⟩″
    from ‹U∈τ› have W:–0″→τ using pair˙func˙singleton by simp
    with ‹–0″ = 1› have W:1→τ and W(0) = U using pair˙val by auto
    with ‹V = – –⟨0,y⟩″. y∈U ″› have V = FinProd(1→⋃τ,W)
      using single˙top˙prod by simp
    with A1 ‹W:1→τ› have V ∈ ProductTopBase(1,τ) using fin˙prod˙def˙nat
      by auto
   } thus – – –⟨0,y⟩″. y∈U ″. U∈τ″ ⊆ ProductTopBase(1,τ) by auto
  qed
  ultimately have I: ProductTopBase(1,τ) –is a topology″ and
    II: IsAhomeomorphism(τ, ProductTopBase(1,τ),b) by auto
  from A1 have ProductTopBase(1,τ) –is a base for″ SeqProductTopology(1,τ)
    using seq˙prod˙top˙is˙top by simp
 with I have ProductTopBase(1,τ) = SeqProductTopology(1,τ) by (rule base˙topology)
 with ‹ProductTopBase(1,τ) = – – –⟨0,y⟩″. y∈U ″. U∈τ″› II show
    SeqProductTopology(1,τ) = – – –⟨0,y⟩″. y∈U ″. U∈τ″ and
    IsAhomeomorphism(τ,SeqProductTopology(1,τ),–⟨y,–⟨0,y⟩″⟩.y ∈ ⋃τ″) by auto
qed
```

A special corner case of finite˙top˙prod˙homeo: a space X is homeomorphic to the space of one element lists of X.

```
theorem singleton˙prod˙top1: assumes A1: τ –is a topology″
```

shows IsAhomeomorphism(SeqProductTopology(1,τ),τ,–⟨x,x(0)⟩. x∈1→⋃τ″)
proof -
have –⟨x,x(0)⟩. x∈1→⋃τ″ = converse(–⟨y,–⟨0,y⟩″⟩.y∈⋃τ″)
using list˙singleton˙bij by blast
with A1 show thesis using singleton˙prod˙top homeo˙inv by simp
qed

A technical lemma describing the carrier of a (cartesian) product topology of the (sequence) product topology of n copies of topology τ and another copy of τ.

lemma finite˙prod˙top: assumes τ –is a topology″ and T = SeqProductTopology(n,τ)
shows (⋃ProductTopology(T,τ)) = (n→⋃τ)×⋃τ
using assms Top˙1˙4˙T1 seq˙prod˙top˙is˙top by simp

If U is a set from the base of X^n and V is open in X, then $U \times V$ is in the base of X^{n+1}. The next lemma is an analogue of this fact for the function space approach.

lemma finite˙prod˙succ˙base: assumes A1: τ –is a topology″ and A2: n ∈ nat and
A3: U ∈ ProductTopBase(n,τ) and A4: V∈τ
shows –x ∈ succ(n)→⋃τ. Init(x) ∈ U ∧ x(n) ∈ V″ ∈ ProductTopBase(succ(n),τ)
proof -
let B = –x ∈ succ(n)→⋃τ. Init(x) ∈ U ∧ x(n) ∈ V″
from A1 A2 have ProductTopBase(n,τ) = –FinProd(n→⋃τ,W). W∈n→τ″
using fin˙prod˙def˙nat by simp
with A3 obtain W_U where W_U:n→τ and U =FinProd(n→⋃τ,W_U) by auto
let W = Append(W_U,V)
from A4 and ‹W_U:n→τ› have W:succ(n)→τ using append˙props by simp
moreover have B = FinProd(succ(n)→⋃τ,W)
proof
{ fix x assume x∈B
with ‹W:succ(n)→τ› have x ∈ succ(n)→⋃τ and domain(W) = succ(n) using func1˙1˙L1
by auto
moreover from A2 A4 ‹x∈B› ‹U =FinProd(n→⋃τ,W_U)› ‹W_U:n→τ› ‹x ∈ succ(n)→⋃τ›
have ∀i∈succ(n). x(i) ∈ W(i) using func1˙1˙L1 FinProd˙def init˙props append˙props
by simp
ultimately have x ∈ FinProd(succ(n)→⋃τ,W) using FinProd˙def by simp
} thus B ⊆ FinProd(succ(n)→⋃τ,W) by auto
next
{ fix x assume x ∈ FinProd(succ(n)→⋃τ,W)
then have x:succ(n)→⋃τ and I: ∀i ∈ domain(W). x(i) ∈ W(i)
using FinProd˙def by auto
moreover have Init(x) ∈ U
proof -
from A2 and ‹x:succ(n)→⋃τ› have Init(x):n→⋃τ using init˙props by simp
moreover have ∀i∈domain(W_U). Init(x)(i) ∈ W_U(i)

```
        proof -
          from A2 ‹x ∈ FinProd(succ(n)→⋃τ,W)› ‹W:succ(n)→τ› have ∀i∈n. x(i)
∈ W(i)
            using FinProd˙def func1˙1˙L1 by simp
          moreover from A2 ‹x: succ(n)→⋃τ› have ∀i∈n. Init(x)(i) = x(i)
            using init˙props by simp
          moreover from A4 and ‹W_U:n→τ› have ∀i∈n. W(i) =  W_U(i)
            using append˙props by simp
          ultimately have ∀i∈n. Init(x)(i) ∈ W_U(i) by simp
          with ‹W_U:n→τ› show thesis using func1˙1˙L1 by simp
        qed
        ultimately have Init(x) ∈ FinProd(n→⋃τ,W_U) using FinProd˙def by simp
        with ‹U =FinProd(n→⋃τ,W_U)› show thesis by simp
      qed
      moreover have x(n) ∈ V
      proof -
        from ‹W:succ(n)→τ› I  have x(n) ∈ W(n) using func1˙1˙L1 by simp
        moreover from A4 ‹W_U:n→τ› have W(n) = V using append˙props by simp
        ultimately show thesis by simp
      qed
      ultimately have x∈B by simp
    } thus FinProd(succ(n)→⋃τ,W) ⊆ B by auto
  qed
  moreover from A1 A2 have
    ProductTopBase(succ(n),τ) = –FinProd(succ(n)→⋃τ,W). W∈succ(n)→τ″
    using fin˙prod˙def˙nat by simp
  ultimately show thesis by auto
qed
```

If U is open in X^n and V is open in X, then $U \times V$ is open in X^{n+1}. The next lemma is an analogue of this fact for the function space approach.

```
lemma finite˙prod˙succ: assumes A1: τ –is a topology″ and A2: n ∈ nat and
  A3: U ∈ SeqProductTopology(n,τ) and A4: V∈τ
 shows –x ∈ succ(n)→⋃τ. Init(x) ∈ U ∧ x(n) ∈ V″ ∈ SeqProductTopology(succ(n),τ)
  proof -
    from A1 have ProductTopBase(n,τ) –is a base for″ SeqProductTopology(n,τ)
and
     I: ProductTopBase(succ(n),τ) –is a base for″ SeqProductTopology(succ(n),τ)
and
     II: SeqProductTopology(succ(n),τ) –is a topology″
       using seq˙prod˙top˙is˙top by auto
   with A3 have ∃ℬ ∈ Pow(ProductTopBase(n,τ)). U = ⋃ℬ using Top˙1˙2˙L1 by
simp
   then obtain ℬ where ℬ ⊆ ProductTopBase(n,τ) and U = ⋃ℬ by auto
   then have
   –x:succ(n)→⋃τ. Init(x) ∈ U ∧ x(n) ∈ V″ = (⋃B∈ℬ.–x:succ(n)→⋃τ. Init(x)
∈ B ∧ x(n) ∈ V″)
     by auto
   moreover from A1 A2 A4 ‹ℬ ⊆ ProductTopBase(n,τ)› have
```

∀B∈$\mathcal{B}$. (–x:succ(n)→⋃τ. Init(x) ∈ B ∧ x(n) ∈ V″ ∈ ProductTopBase(succ(n),τ))
using finite˙prod˙succ˙base by auto
with I II have
(⋃B∈$\mathcal{B}$.–x:succ(n)→⋃τ. Init(x) ∈ B ∧ x(n) ∈ V″) ∈ SeqProductTopology(succ(n),τ)
using base˙sets˙open union˙indexed˙open by auto
ultimately show thesis by simp
qed

In the Topology˙ZF˙2 theory we define product topology of two topological spaces. The next lemma explains in what sense the topology on finite lists of length n of elements of topological space X can be thought as a model of the product topology on the cartesian product of n copies of that space. Namely, we show that the space of lists of length $n+1$ of elements of X is homeomorphic to the product topology (as defined in Topology˙ZF˙2) of two spaces: the space of lists of length n and X. Recall that if $\mathcal{B}$ is a base (i.e. satisfies the base condition), then the collection $\{\bigcup B | B \in Pow(\mathcal{B})\}$ is a topology (generated by $\mathcal{B}$).

theorem finite˙top˙prod˙homeo: assumes A1: τ –is a topology″ and A2: n ∈ nat and

A3: f = –⟨x,⟨Init(x),x(n)⟩⟩. x ∈ succ(n)→⋃τ″ and
A4: T = SeqProductTopology(n,τ) and
A5: S = SeqProductTopology(succ(n),τ)
shows IsAhomeomorphism(S,ProductTopology(T,τ),f)
proof -
let C = ProductCollection(T,τ)
let B = ProductTopBase(succ(n),τ)
from A1 A4 have T –is a topology″ using seq˙prod˙top˙is˙top by simp
with A1 A5 have S –is a topology″ and ProductTopology(T,τ) –is a topology″
using seq˙prod˙top˙is˙top Top˙1˙4˙T1 by auto
moreover
from assms have f ∈ bij(⋃S,⋃ProductTopology(T,τ))
using lists˙cart˙prod seq˙prod˙top˙is˙top Top˙1˙4˙T1 by simp
then have f: ⋃S→⋃ProductTopology(T,τ) using bij˙is˙fun by simp
ultimately have two˙top˙spaces0(S,ProductTopology(T,τ),f) using two˙top˙spaces0˙def by simp
moreover note ‹f ∈ bij(⋃S,⋃ProductTopology(T,τ))›
moreover from A1 A5 have B –is a base for″ S
using seq˙prod˙top˙is˙top by simp
moreover from A1 ‹T –is a topology″› have C –is a base for″ ProductTopology(T,τ)

using Top˙1˙4˙T1 by auto
moreover have ∀W∈C. f-(W) ∈ S
proof
fix W assume W∈C
then obtain U V where U∈T V∈τ and W = U×V using ProductCollection˙def by auto
from A1 A5 ‹f: ⋃S→⋃ProductTopology(T,τ)› have f: (succ(n)→⋃τ)→⋃Pro-

ductTopology(T,τ)
using seq˙prod˙top˙is˙top by simp
with assms ‹W = U×V› ‹U∈T› ‹V∈τ› show f-(W) ∈ S
using ZF˙fun˙from˙tot˙val func1˙1˙L15 finite˙prod˙succ by simp
qed
moreover have ∀V∈B. f(V) ∈ ProductTopology(T,τ)
proof
fix V assume V∈B
with A1 A2 obtain W_V where W_V ∈ succ(n)→τ and V = FinProd(succ(n)→$\bigcup\tau$,W_V)

using fin˙prod˙def˙nat by auto
let U = FinProd(n→$\bigcup\tau$,Init(W_V))
let W = W_V(n)
have U ∈ T
proof -
from A1 A2 ‹W_V ∈ succ(n)→τ› have U ∈ ProductTopBase(n,τ)
using fin˙prod˙def˙nat init˙props by auto
with A1 A4 show thesis using seq˙prod˙top˙is˙top base˙sets˙open by blast
qed
from A1 ‹W_V ∈ succ(n)→τ› ‹T –is a topology″› ‹U ∈ T› have U×W ∈ Product-
Topology(T,τ)
using apply˙funtype prod˙open˙open˙prod by simp
moreover have f(V) = U×W
proof -
from A2 ‹W_V: succ(n)→τ› have Init(W_V): n→τ and III: ∀k∈n. Init(W_V)(k)
= W_V(k)
using init˙props by auto
then have domain(Init(W_V)) = n using func1˙1˙L1 by simp
have f(V) = –⟨Init(x),x(n)⟩. x∈V″
proof -
have f(V) = –f(x). x∈V″
proof -
from A1 A5 have B –is a base for″ S using seq˙prod˙top˙is˙top by simp
with ‹V∈B› have V ⊆ ⋃S using IsAbaseFor˙def by auto
with ‹f: ⋃S→⋃ProductTopology(T,τ)› show thesis using func˙imagedef by
simp
qed
moreover have ∀x∈V. f(x) = ⟨Init(x),x(n)⟩
proof -
from A1 A3 A5 ‹V = FinProd(succ(n)→$\bigcup\tau$,W_V)› have V ⊆ ⋃S and
fdef: f = –⟨x,⟨Init(x),x(n)⟩⟩. x ∈ ⋃S″ using seq˙prod˙top˙is˙top FinProd˙def

by auto
from ‹f: ⋃S→⋃ProductTopology(T,τ)› fdef have ∀x ∈ ⋃S. f(x) =
⟨Init(x),x(n)⟩
by (rule ZF˙fun˙from˙tot˙val0)
with ‹V ⊆ ⋃S› show thesis by auto
qed
ultimately show thesis by simp

```
qed
also have −⟨Init(x),x(n)⟩. x∈V″ = U×W
proof
  { fix y assume y ∈ −⟨Init(x),x(n)⟩. x∈V″
    then obtain x where I: y = ⟨Init(x),x(n)⟩ and x∈V by auto
    with ‹V = FinProd(succ(n)→⋃τ,W_V)› have
      x:succ(n)→⋃τ and II: ∀k∈domain(W_V). x(k) ∈ W_V(k)
      unfolding FinProd˙def by auto
    with A2 ‹W_V: succ(n)→τ› have IV: ∀k∈n. Init(x)(k) = x(k)
      using init˙props by simp
    have Init(x) ∈ U
    proof -
      from A2 ‹x:succ(n)→⋃τ› have Init(x): n→⋃τ using init˙props by simp
      moreover have ∀k∈domain(Init(W_V)). Init(x)(k) ∈ Init(W_V)(k)
      proof -
       from A2 ‹W_V: succ(n)→τ› have Init(W_V): n→τ using init˙props by simp
        then have domain(Init(W_V)) = n using func1˙1˙L1 by simp
        note III IV  ‹domain(Init(W_V)) = n›
        moreover from II ‹W_V ∈ succ(n)→τ› have ∀k∈n. x(k) ∈ W_V(k)
          using func1˙1˙L1 by simp
        ultimately show thesis by simp
      qed
      ultimately show Init(x) ∈ U using FinProd˙def by simp
    qed
   moreover from ‹W_V: succ(n)→τ› II have x(n) ∈ W using func1˙1˙L1 by simp
    ultimately have ⟨Init(x),x(n)⟩ ∈ U×W by simp
    with I have y ∈ U×W by simp
  } thus −⟨Init(x),x(n)⟩. x∈V″ ⊆ U×W by auto
  { fix y assume y ∈ U×W
    then have fst(y) ∈ U and snd(y) ∈ W by auto
    with ‹domain(Init(W_V)) = n› have fst(y): n→⋃τ and
      V: ∀k∈n. fst(y)(k) ∈ Init(W_V)(k)
      using FinProd˙def by auto
    from ‹W_V: succ(n)→τ› have W ∈ τ using apply˙funtype by simp
    with ‹snd(y) ∈ W› have snd(y) ∈ ⋃τ by auto
    let x = Append(fst(y),snd(y))
    have x∈V
    proof -
   from ‹fst(y): n→⋃τ› ‹snd(y) ∈ ⋃τ› have x:succ(n)→⋃τ using append˙props
by simp
      moreover have ∀i∈domain(W_V). x(i) ∈ W_V(i)
      proof -
        from ‹fst(y): n→⋃τ› ‹snd(y) ∈ ⋃τ›
          have ∀k∈n. x(k) = fst(y)(k) and x(n) = snd(y)
          using append˙props by auto
        moreover from III V have ∀k∈n. fst(y)(k) ∈ W_V(k) by simp
        moreover note ‹snd(y) ∈ W›
        ultimately have ∀i∈succ(n). x(i) ∈ W_V(i) by simp
        with ‹W_V ∈ succ(n)→τ› show thesis using func1˙1˙L1 by simp
```

```
          qed
            ultimately have x ∈ FinProd(succ(n)→⋃τ,W_V) using FinProd˙def by
simp
          with ‹V = FinProd(succ(n)→⋃τ,W_V)› show x∈V by simp
        qed
          moreover from A2 ‹y ∈ U×W› ‹fst(y): n→⋃τ› ‹snd(y) ∈ ⋃τ› have y =
⟨Init(x),x(n)⟩
          using init˙append append˙props by auto
        ultimately have y ∈ −⟨Init(x),x(n)⟩. x∈V″ by auto
      } thus U×W ⊆ −⟨Init(x),x(n)⟩. x∈V″ by auto
    qed
    finally show f(V) = U×W by simp
  qed
  ultimately show f(V) ∈ ProductTopology(T,τ) by simp
 qed
 ultimately show thesis using two˙top˙spaces0.bij˙base˙open˙homeo by simp
qed


end
```

55 Topology 4

```
theory Topology˙ZF˙4 imports Topology˙ZF˙1 Order˙ZF func1 NatOrder˙ZF
begin
```

This theory deals with convergence in topological spaces. Contributed by Daniel de la Concepcion.

55.1 Nets

Nets are a generalization of sequences. It is known that sequences do not determine the behavior of the topological spaces that are not first countable; i.e., have a countable neighborhood base for each point. To solve this problem, nets were defined so that the behavior of any topological space can be thought in terms of convergence of nets.

First we need to define what a directed set is:

```
definition
 IsDirectedSet (˙ directs ˙ 90)
 where r directs D ≡ refl(D,r) ∧ trans(r) ∧ (∀x∈D. ∀y∈D. ∃z∈D. ⟨x,z⟩∈r ∧ ⟨y,z⟩∈r)
```

Any linear order is a directed set; in particular $(\mathbb{N}, \leq)$.

```
lemma linorder˙imp˙directed:
 assumes IsLinOrder(X,r)
 shows r directs X
proof-
```

```
  from assms have trans(r) using IsLinOrder˙def by auto
  moreover
  from assms have r:refl(X,r) using IsLinOrder˙def total˙is˙refl by auto
  moreover
  {
    fix x y
    assume R: x∈X y∈X
    with assms have ⟨x,y⟩∈r ∨ ⟨y,x⟩∈r using IsLinOrder˙def IsTotal˙def by auto
    with r have (⟨x,y⟩∈r ∧ ⟨y,y⟩∈r)∨(⟨y,x⟩∈r ∧ ⟨x,x⟩∈r) using R refl˙def by auto
    then have ∃z∈X. ⟨x,z⟩∈r ∧ ⟨y,z⟩∈r using R by auto
  }
  ultimately show thesis using IsDirectedSet˙def function˙def by auto
qed
```

Natural numbers are a directed set.

```
corollary Le˙directs˙nat:
  shows IsLinOrder(nat,Le) Le directs nat
proof -
  show IsLinOrder(nat,Le)  by (rule NatOrder˙ZF˙1˙L2)
  then show Le directs nat using linorder˙imp˙directed by auto
qed
```

We are able to define the concept of net, now that we now what a directed set is.

```
definition
  IsNet (˙ –is a net on˝ ˙ 90)
   where N –is a net on˝ X ≡ fst(N):domain(fst(N))→X ∧ (snd(N) directs domain(fst(N))) ∧ domain(fst(N))≠0
```

Provided a topology and a net directed on its underlying set, we can talk about convergence of the net in the topology.

```
definition (in topology0)
  NetConverges (˙ →N ˙ 90)
  where N –is a net on˝ ⋃T ⟹ N →N x ≡
 (x∈⋃T) ∧ (∀U∈Pow(⋃T). (x∈int(U) ⟶ (∃t∈domain(fst(N)). ∀m∈domain(fst(N)).

    (⟨t,m⟩∈snd(N) ⟶ fst(N)m∈U))))
```

One of the most important directed sets, is the neighborhoods of a point.

```
theorem (in topology0) directedset˙neighborhoods:
  assumes x∈⋃T
  defines Neigh≡–U∈Pow(⋃T). x∈int(U)˝
  defines r≡–⟨U,V⟩∈(Neigh × Neigh). V⊆U˝
  shows r directs Neigh
proof-
  {
    fix U
    assume U ∈ Neigh
```

```
    then have ⟨U,U⟩ ∈ r using r˙def by auto
  }
  then have refl(Neigh,r) using refl˙def by auto
  moreover
  {
    fix U V W
    assume ⟨U,V⟩ ∈ r ⟨V,W⟩ ∈ r
    then have U ∈ Neigh W ∈ Neigh W⊆U using r˙def by auto
    then have ⟨U,W⟩∈r using r˙def by auto
  }
  then have trans(r) using trans˙def by blast
  moreover
  {
    fix A B
    assume p: A∈Neigh B∈Neigh
    have A∩B ∈ Neigh
    proof-
      from p have A∩B ∈ Pow(⋃T) using Neigh˙def by auto
      moreover
      { from p have x∈int(A)x∈int(B) using Neigh˙def by auto
        then have x∈int(A)∩int(B) by auto
        moreover
        { have int(A)∩int(B)⊆A∩B using Top˙2˙L1  by auto
          moreover have int(A)∩int(B)∈T
            using Top˙2˙L2 Top˙2˙L2 topSpaceAssum IsATopology˙def by blast
          ultimately have int(A)∩int(B)⊆int(A∩B)
          using Top˙2˙L5 by auto
        }
        ultimately have x ∈ int(A∩B) by auto
      }
      ultimately show thesis using Neigh˙def by auto
    qed
    moreover from ‹A∩B ∈ Neigh› have ⟨A,A∩B⟩∈r ∧ ⟨B,A∩B⟩∈r
      using r˙def p by auto
    ultimately
    have ∃z∈Neigh. ⟨A,z⟩∈r ∧ ⟨B,z⟩∈r by auto
  }
  ultimately show thesis using IsDirectedSet˙def by auto
qed
```

There can be nets directed by the neighborhoods that converge to the point; if there is a choice function.

```
theorem (in topology0) net˙direct˙neigh˙converg:
  assumes x∈⋃T
  defines Neigh≡–U∈Pow(⋃T). x∈int(U)″
  defines r≡–⟨U,V⟩∈(Neigh × Neigh). V⊆U″
  assumes f:Neigh→⋃T ∀U∈Neigh. f(U) ∈ U
  shows ⟨f,r⟩ →N x
proof -
```

```
  from assms(4) have dom˙def: Neigh = domain(f) using Pi˙def by auto
  moreover
    have ⋃T∈T using topSpaceAssum IsATopology˙def by auto
    then have int(⋃T)=⋃T using Top˙2˙L3 by auto
    with assms(1) have ⋃T∈Neigh using Neigh˙def by auto
    then have ⋃T∈domain(fst(⟨f,r⟩)) using dom˙def by auto
  moreover from assms(4) dom˙def have fst(⟨f,r⟩):domain(fst(⟨f,r⟩))→⋃T
    by auto
  moreover from assms(1,2,3) dom˙def have snd(⟨f,r⟩) directs domain(fst(⟨f,r⟩))
      using directedset˙neighborhoods by simp
  ultimately have Net: ⟨f,r⟩ –is a net on˝ ⋃T unfolding IsNet˙def by auto
  {
    fix U
    assume U ∈ Pow(⋃T) x ∈ int(U)
    then have U ∈ Neigh using Neigh˙def by auto
    then have t: U ∈ domain(f) using dom˙def by auto
    {
      fix W
      assume A: W∈domain(f) ⟨U,W⟩∈r
      then have W∈Neigh using dom˙def by auto
      with assms(5) have fW∈W by auto
      with A(2) r˙def have fW∈U by auto
    }
    then have ∀W∈domain(f). (⟨U,W⟩∈r ⟶ fW∈U) by auto
    with t have ∃V∈domain(f). ∀W∈domain(f). (⟨V,W⟩∈r ⟶ fW∈U) by auto
  }
 then have ∀U∈Pow(⋃T). (x∈int(U) ⟶ (∃V∈domain(f). ∀W∈domain(f). (⟨V,W⟩∈r
⟶ f(W) ∈ U)))
    by auto
  with assms(1) Net show thesis using NetConverges˙def by auto
qed
```

55.2 Filters

Nets are a generalization of sequences that can make us see that not all topological spaces can be described by sequences. Nevertheless, nets are not always the tool used to deal with convergence. The reason is that they make use of directed sets which are completely unrelated with the topology.

The topological tools to deal with convergence are what is called filters.

```
definition
  IsFilter (˙ –is a filter on˝ ˙ 90)
  where 𝔉 –is a filter on˝ X ≡ (0∉𝔉) ∧ (X∈𝔉) ∧ (𝔉⊆Pow(X)) ∧
  (∀A∈𝔉. ∀B∈𝔉. A∩B∈𝔉) ∧ (∀B∈𝔉. ∀C∈Pow(X). B⊆C ⟶ C∈𝔉)
```

Not all the sets of a filter are needed to be consider at all times; as it happens with a topology we can consider bases.

```
definition
```

IsBaseFilter (˙ –is a base filter″ ˙ 90)
where C –is a base filter″ $\mathfrak{F}$ ≡ C⊆$\mathfrak{F}$ ∧ $\mathfrak{F}$=–A∈Pow(⋃$\mathfrak{F}$). (∃D∈C. D⊆A)″

Not every set is a base for a filter, as it happens with topologies, there is a condition to be satisfied.

definition
SatisfiesFilterBase (˙ –satisfies the filter base condition″ 90)
where C –satisfies the filter base condition″ ≡ (∀A∈C. ∀B∈C. ∃D∈C. D⊆A∩B) ∧ C≠0 ∧ 0∉C

Every set of a filter contains a set from the filter's base.

lemma basic˙element˙filter:
assumes A∈$\mathfrak{F}$ and C –is a base filter″ $\mathfrak{F}$
shows ∃D∈C. D⊆A
proof-
from assms(2) have t:$\mathfrak{F}$=–A∈Pow(⋃$\mathfrak{F}$). (∃D∈C. D⊆A)″ using IsBaseFilter˙def by auto
with assms(1) have A∈–A∈Pow(⋃$\mathfrak{F}$). (∃D∈C. D⊆A)″ by auto
then have A∈Pow(⋃$\mathfrak{F}$) ∃D∈C. D⊆A by auto
then show thesis by auto
qed

The following two results state that the filter base condition is necessary and sufficient for the filter generated by a base, to be an actual filter. The third result, rewrites the previous two.

theorem basic˙filter˙1:
assumes C –is a base filter″ $\mathfrak{F}$ and C –satisfies the filter base condition″
shows $\mathfrak{F}$ –is a filter on″ ⋃$\mathfrak{F}$
proof-
{
fix A B
assume AF: A∈$\mathfrak{F}$ and BF: B∈$\mathfrak{F}$
with assms(1) have ∃DA∈C. DA⊆A using basic˙element˙filter by simp
then obtain DA where perA: DA∈C and subA: DA⊆A by auto
from BF assms have ∃DB∈C. DB⊆B using basic˙element˙filter by simp
then obtain DB where perB: DB∈C and subB: DB⊆B by auto
from assms(2) perA perB have ∃D∈C. D⊆DA∩DB
unfolding SatisfiesFilterBase˙def by auto
then obtain D where D∈C D⊆DA∩DB by auto
with subA subB AF BF have A∩B∈–A ∈ Pow(⋃$\mathfrak{F}$) . ∃D∈C. D ⊆ A″ by auto
with assms(1) have A∩B∈$\mathfrak{F}$ unfolding IsBaseFilter˙def by auto
}
moreover
{
fix A B
assume AF: A∈$\mathfrak{F}$ and BS: B∈Pow(⋃$\mathfrak{F}$) and sub: A⊆B
from assms(1) AF have ∃D∈C. D⊆A using basic˙element˙filter by auto
then obtain D where D⊆A D∈C by auto

```
    with sub BS have B∈–A∈Pow(⋃𝔉). ∃D∈C. D⊆A″ by auto
    with assms(1) have B∈𝔉 unfolding IsBaseFilter˙def by auto
    }
  moreover
  from assms(2) have C≠0 using SatisfiesFilterBase˙def by auto
  then obtain D where D∈C by auto
  with assms(1) have D⊆⋃𝔉 using IsBaseFilter˙def by auto
  with ‹D∈C› have ⋃𝔉∈–A∈Pow(⋃𝔉). ∃D∈C. D⊆A″ by auto
  with assms(1) have ⋃𝔉∈𝔉 unfolding IsBaseFilter˙def by auto
  moreover
  {
    assume 0∈𝔉
    with assms(1) have ∃D∈C. D⊆0 using basic˙element˙filter by simp
    then obtain D where D∈CD⊆0 by auto
    then have D∈C D=0 by auto
    with assms(2) have False using SatisfiesFilterBase˙def by auto
  }
  then have 0∉𝔉 by auto
  ultimately show thesis using IsFilter˙def by auto
qed
```

A base filter satisfies the filter base condition.

```
theorem basic˙filter˙2:
  assumes C –is a base filter″ 𝔉 and 𝔉 –is a filter on″ ⋃𝔉
  shows C –satisfies the filter base condition″
proof-
  {
    fix A B
    assume AF: A∈C and BF: B∈C
    then have A∈𝔉 and B∈𝔉 using assms(1) IsBaseFilter˙def by auto
    then have A∩B∈𝔉 using assms(2) IsFilter˙def by auto
    then have ∃D∈C. D⊆A∩B using assms(1) basic˙element˙filter by blast
  }
  then have ∀A∈C. ∀B∈C. ∃D∈C. D⊆A∩B by auto
  moreover
  {
    assume 0∈C
    then have 0∈𝔉 using assms(1) IsBaseFilter˙def by auto
    then have False using assms(2) IsFilter˙def by auto
  }
  then have 0∉C by auto
  moreover
  {
    assume C=0
    then have 𝔉=0 using assms(1) IsBaseFilter˙def by auto
    then have False using assms(2) IsFilter˙def by auto
  }
  then have C≠0 by auto
  ultimately show thesis using SatisfiesFilterBase˙def by auto
```

qed

A base filter for a collection satisfies the filter base condition iff that collection is in fact a filter.

theorem basic˙filter:
assumes C –is a base filter″ $\mathfrak{F}$
shows (C –satisfies the filter base condition″) $\longleftrightarrow$ ($\mathfrak{F}$ –is a filter on″ $\bigcup\mathfrak{F}$)
using assms basic˙filter˙1 basic˙filter˙2 by auto

A base for a filter determines a filter up to the underlying set.

theorem base˙unique˙filter:
assumes C –is a base filter″ $\mathfrak{F}$1and C –is a base filter″ $\mathfrak{F}$2
shows $\mathfrak{F}1=\mathfrak{F}2 \longleftrightarrow \bigcup\mathfrak{F}1=\bigcup\mathfrak{F}2$
using assms unfolding IsBaseFilter˙def by auto

Suppose that we take any nonempty collection C of subsets of some set X. Then this collection is a base filter for the collection of all supersets (in X) of sets from C.

theorem base˙unique˙filter˙set1:
assumes C ⊆ Pow(X) and C≠0
shows C –is a base filter″ –A∈Pow(X). ∃D∈C. D⊆A″ and ⋃–A∈Pow(X). ∃D∈C. D⊆A″=X
proof-
from assms(1) have C⊆–A∈Pow(X). ∃D∈C. D⊆A″ by auto
moreover
from assms(2) obtain D where D∈C by auto
then have D⊆X using assms(1) by auto
with ‹D∈C› have X∈–A∈Pow(X). ∃D∈C. D⊆A″ by auto
then show ⋃–A∈Pow(X). ∃D∈C. D⊆A″=X by auto
ultimately
show C –is a base filter″ –A∈Pow(X). ∃D∈C. D⊆A″ using IsBaseFilter˙def by auto
qed

A collection C that satisfies the filter base condition is a base filter for some other collection $\mathfrak{F}$ iff $\mathfrak{F}$ is the collection of supersets of C.

theorem base˙unique˙filter˙set2:
assumes C⊆Pow(X) and C –satisfies the filter base condition″
shows ((C –is a base filter″ $\mathfrak{F}$) ∧ $\bigcup\mathfrak{F}$=X) $\longleftrightarrow$ $\mathfrak{F}$=–A∈Pow(X). ∃D∈C. D⊆A″
using assms IsBaseFilter˙def SatisfiesFilterBase˙def base˙unique˙filter˙set1
by auto

A simple corollary from the previous lemma.

corollary base˙unique˙filter˙set3:
assumes C⊆Pow(X) and C –satisfies the filter base condition″
shows C –is a base filter″ –A∈Pow(X). ∃D∈C. D⊆A″ and ⋃–A∈Pow(X). ∃D∈C. D⊆A″ = X
proof -

```
  let 𝔉 = –A∈Pow(X). ∃D∈C. D⊆A″
  from assms have (C –is a base filter″ 𝔉) ∧ ⋃𝔉=X
    using base˙unique˙filter˙set2 by simp
  thus C –is a base filter″ 𝔉 and ⋃𝔉 = X
    by auto
qed
```

The convergence for filters is much easier concept to write. Given a topology and a filter on the same underlying set, we can define convergence as containing all the neighborhoods of the point.

```
definition (in topology0)
  FilterConverges (˙ →F ˙ 50) where
  𝔉–is a filter on″⋃T  ⟹ 𝔉→Fx ≡
  x∈⋃T ∧ (–U∈Pow(⋃T). x∈int(U)″ ⊆ 𝔉)
```

The neighborhoods of a point form a filter that converges to that point.

```
lemma (in topology0) neigh˙filter:
  assumes x∈⋃T
  defines Neigh≡–U∈Pow(⋃T). x∈int(U)″
  shows Neigh –is a filter on″⋃T and Neigh →F x
proof-
  {
    fix A B
    assume p:A∈Neigh B∈Neigh
    have A∩B∈Neigh
    proof-
      from p have A∩B∈Pow(⋃T) using Neigh˙def by auto
      moreover
      {from p have x∈int(A) x∈int(B) using Neigh˙def by auto
      then have x∈int(A)∩int(B) by auto
      moreover
      { have int(A)∩int(B)⊆A∩B using Top˙2˙L1 by auto
        moreover have int(A)∩int(B)∈T
          using Top˙2˙L2 topSpaceAssum IsATopology˙def by blast
        ultimately have int(A)∩int(B)⊆int(A∩B) using Top˙2˙L5 by auto}
        ultimately have x∈int(A∩B) by auto
      }
      ultimately show thesis using Neigh˙def by auto
    qed
    }
  moreover
  {
    fix A B
    assume A: A∈Neigh and B: B∈Pow(⋃T) and sub: A⊆B
    from sub have int(A)∈T int(A)⊆B using Top˙2˙L2 Top˙2˙L1
      by auto
    then have int(A)⊆int(B) using Top˙2˙L5  by auto
    with A have x∈int(B) using Neigh˙def by auto
    with B have B∈Neigh using Neigh˙def by auto
```

```
    }
  moreover
  {
    assume 0∈Neigh
    then have x∈Interior(0,T) using Neigh˙def by auto
    then have x∈0 using Top˙2˙L1 by auto
    then have False by auto
    }
  then have 0∉Neigh by auto
  moreover
  have ⋃T∈T using topSpaceAssum IsATopology˙def by auto
  then have Interior(⋃T,T)=⋃T using Top˙2˙L3 by auto
  with assms(1) have ab: ⋃T∈Neigh unfolding Neigh˙def by auto
  moreover have Neigh⊆Pow(⋃T) using Neigh˙def by auto
  ultimately show Neigh –is a filter on″ ⋃T using IsFilter˙def
    by auto
  moreover from ab have ⋃Neigh=⋃T unfolding Neigh˙def by auto
   ultimately show Neigh →F x using FilterConverges˙def assms(1) Neigh˙def by
auto
qed
```

Note that with the net we built in a previous result, it wasn't clear that we could construct an actual net that converged to the given point without the axiom of choice. With filters, there is no problem.

Another positive point of filters is due to the existence of filter basis. If we have a basis for a filter, then the filter converges to a point iff every neighborhood of that point contains a basic filter element.

```
theorem (in topology0) convergence˙filter˙base1:
  assumes 𝔉 –is a filter on″ ⋃T and C –is a base filter″ 𝔉 and 𝔉 →F x
  shows ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U) and x∈⋃T
proof -
  { fix U
    assume U⊆(⋃T) and x∈int(U)
    with assms(1,3) have U∈𝔉 using FilterConverges˙def by auto
    with assms(2) have ∃D∈C. D⊆U using basic˙element˙filter by blast
  } thus ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U) by auto
  from assms(1,3) show x∈⋃T using  FilterConverges˙def by auto
qed
```

A sufficient condition for a filter to converge to a point.

```
theorem (in topology0) convergence˙filter˙base2:
  assumes 𝔉 –is a filter on″ ⋃T and C –is a base filter″ 𝔉
    and ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U) and x∈⋃T
  shows 𝔉 →F x
proof-
  {
    fix U
    assume AS: U∈Pow(⋃T) x∈int(U)
```

then obtain D where pD:D∈C and s:D⊆U using assms(3) by blast
with assms(2) AS have D∈$\mathfrak{F}$ and D⊆U and U∈Pow(⋃T)
using IsBaseFilter˙def by auto
with assms(1) have U∈$\mathfrak{F}$ using IsFilter˙def by auto
}
with assms(1,4) show thesis using FilterConverges˙def by auto
qed

A necessary and sufficient condition for a filter to converge to a point.

theorem (in topology0) convergence˙filter˙base˙eq:
assumes $\mathfrak{F}$ –is a filter on″ ⋃T and C –is a base filter″ $\mathfrak{F}$
shows ($\mathfrak{F}$ $\rightarrow_F$ x) ⟷ ((∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U)) ∧ x∈⋃T)
proof
assume $\mathfrak{F}$ $\rightarrow_F$ x
with assms show ((∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U)) ∧ x∈⋃T)
using convergence˙filter˙base1 by simp
next
assume (∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U)) ∧ x∈⋃T
with assms show $\mathfrak{F}$ $\rightarrow_F$ x using convergence˙filter˙base2
by auto
qed

55.3 Relation between nets and filters

In this section we show that filters do not generalize nets, but still nets and filter are in w way equivalent as far as convergence is considered.

Let's build now a net from a filter, such that both converge to the same points.

definition
NetOfFilter (Net(˙) 40) where
$\mathfrak{F}$ –is a filter on″ ⋃$\mathfrak{F}$ ⟹ Net($\mathfrak{F}$) ≡
⟨–⟨A,fst(A)⟩. A∈–⟨x,F⟩∈(⋃$\mathfrak{F}$)×$\mathfrak{F}$. x∈F″″,–⟨A,B⟩∈–⟨x,F⟩∈(⋃$\mathfrak{F}$)×$\mathfrak{F}$. x∈F″×–⟨x,F⟩∈(⋃$\mathfrak{F}$)×$\mathfrak{F}$. x∈F″. snd(B)⊆snd(A)″⟩

Net of a filter is indeed a net.

theorem net˙of˙filter˙is˙net:
assumes $\mathfrak{F}$ –is a filter on″ X
shows (Net($\mathfrak{F}$)) –is a net on″ X
proof-
from assms have X∈$\mathfrak{F}$ $\mathfrak{F}$⊆Pow(X) using IsFilter˙def by auto
then have uu:⋃$\mathfrak{F}$=X by blast
let f = –⟨A,fst(A)⟩. A∈–⟨x,F⟩∈(⋃$\mathfrak{F}$)×$\mathfrak{F}$. x∈F″″
let r = –⟨A,B⟩∈–⟨x,F⟩∈(⋃$\mathfrak{F}$)×$\mathfrak{F}$. x∈F″×–⟨x,F⟩∈(⋃$\mathfrak{F}$)×$\mathfrak{F}$. x∈F″. snd(B)⊆snd(A)″
have function(f) using function˙def by auto
moreover have relation(f) using relation˙def by auto
ultimately have f:domain(f)→range(f) using function˙imp˙Pi
by auto

```
have dom:domain(f)=−⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F″ by auto
have range(f)⊆⋃𝔉 by auto
with ‹f:domain(f)→range(f)› have f:domain(f)→⋃𝔉 using fun˙weaken˙type by auto
moreover
{
  {
    fix t
    assume pp:t∈domain(f)
    then have snd(t)⊆snd(t) by auto
    with dom pp have ⟨t,t⟩∈r by auto
  }
  then have refl(domain(f),r) using refl˙def by auto
  moreover
  {
    fix t1 t2 t3
    assume ⟨t1,t2⟩∈r ⟨t2,t3⟩∈r
    then have snd(t3)⊆snd(t1) t1∈domain(f) t3∈domain(f) using dom by auto
    then have ⟨t1,t3⟩∈r by auto
  }
  then have trans(r) using trans˙def by auto
  moreover
  {
    fix x y
    assume as:x∈domain(f)y∈domain(f)
    then have snd(x)∈𝔉 snd(y)∈𝔉 by auto
    then have p:snd(x)∩snd(y)∈𝔉 using assms IsFilter˙def by auto
    {
      assume snd(x)∩snd(y)=0
      with p have 0∈𝔉 by auto
      then have False using assms IsFilter˙def by auto
    }
    then have snd(x)∩snd(y)≠0 by auto
    then obtain xy where xy∈snd(x)∩snd(y) by auto
    then have xy∈snd(x)∩snd(y) ⟨xy,snd(x)∩snd(y)⟩∈(⋃𝔉)×𝔉 using p by auto
    then have ⟨xy,snd(x)∩snd(y)⟩∈−⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F″ by auto
    with dom have d:⟨xy,snd(x)∩snd(y)⟩∈domain(f) by auto
    with as have ⟨x,⟨xy,snd(x)∩snd(y)⟩⟩∈r ∧ ⟨y,⟨xy,snd(x)∩snd(y)⟩⟩∈r by auto
    with d have ∃z∈domain(f). ⟨x,z⟩∈r ∧ ⟨y,z⟩∈r  by blast
  }
  then have ∀x∈domain(f). ∀y∈domain(f). ∃z∈domain(f). ⟨x,z⟩∈r ∧ ⟨y,z⟩∈r by
blast
  ultimately have r directs domain(f) using IsDirectedSet˙def by blast
}
moreover
{
  have p:X∈𝔉 and 0∉𝔉 using assms IsFilter˙def by auto
  then have X≠0 by auto
  then obtain q where q∈X by auto
  with p dom have ⟨q,X⟩∈domain(f) by auto
```

```
    then have domain(f)≠0 by blast
  }
  ultimately have ⟨f,r⟩ –is a net on″⋃𝔉 using IsNet˙def by auto
  then show (Net(𝔉)) –is a net on″ X using NetOfFilter˙def assms uu by auto
qed
```

If a filter converges to some point then its net converges to the same point.

```
theorem (in topology0) filter˙conver˙net˙of˙filter˙conver:
  assumes 𝔉 –is a filter on″ ⋃T and 𝔉 →F x
  shows (Net(𝔉)) →N x
proof-
  from assms have ⋃T∈𝔉 𝔉⊆Pow(⋃T) using IsFilter˙def by auto
  then have uu: ⋃𝔉=⋃T by blast
  from assms(1) have func: fst(Net(𝔉))=–⟨A,fst(A)⟩. A∈–⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F″″
    and dir: snd(Net(𝔉))=–⟨A,B⟩∈–⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F″×–⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F″.
snd(B)⊆snd(A)″
    using NetOfFilter˙def uu by auto
  then have dom˙def: domain(fst(Net(𝔉)))=–⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F″ by auto
  from func have fun: fst(Net(𝔉)): –⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F″ → (⋃𝔉)
    using ZF˙fun˙from˙total by simp
  from assms(1) have NN: (Net(𝔉)) –is a net on″⋃T using net˙of˙filter˙is˙net
    by auto
  moreover from assms have x∈⋃T using FilterConverges˙def
    by auto
  moreover
  {
    fix U
    assume AS: U∈Pow(⋃T) x∈int(U)
    with assms have U∈𝔉 x∈U using Top˙2˙L1 FilterConverges˙def by auto
    then have pp: ⟨x,U⟩∈domain(fst(Net(𝔉))) using dom˙def by auto
    {
      fix m
      assume ASS: m∈domain(fst(Net(𝔉))) ⟨⟨x,U⟩,m⟩∈snd(Net(𝔉))
      from ASS(1) fun func have fst(Net(𝔉))(m) = fst(m)
        using func1˙1˙L1 ZF˙fun˙from˙tot˙val by simp
      with dir ASS have fst(Net(𝔉))(m) ∈ U using dom˙def by auto
    }
   then have ∀m∈domain(fst(Net(𝔉))). (⟨⟨x,U⟩,m⟩∈snd(Net(𝔉)) ⟶ fst(Net(𝔉))m∈U)
by auto
   with pp have ∃t∈domain(fst(Net(𝔉))). ∀m∈domain(fst(Net(𝔉))). (⟨t,m⟩∈snd(Net(𝔉))
⟶ fst(Net(𝔉))m∈U)
      by auto
  }
  then have ∀U∈Pow(⋃T).
    (x∈int(U) ⟶ (∃t∈domain(fst(Net(𝔉))). ∀m∈domain(fst(Net(𝔉))). (⟨t,m⟩∈snd(Net(𝔉))
⟶ fst(Net(𝔉))m∈U)))
      by auto
  ultimately show thesis using NetConverges˙def by auto
qed
```

If a net converges to a point, then a filter also converges to a point.

```
theorem (in topology0) net˙of˙filter˙conver˙filter˙conver:
  assumes 𝔉 –is a filter on″⋃T and (Net(𝔉)) →N x
  shows 𝔉 →F x
proof-
  from assms have ⋃T∈𝔉 𝔉⊆Pow(⋃T) using IsFilter˙def by auto
  then have uu: ⋃𝔉=⋃T by blast
  have x∈⋃T using assms NetConverges˙def net˙of˙filter˙is˙net by auto
  moreover
  {
    fix U
    assume U∈Pow(⋃T) x∈int(U)
    then obtain t where t: t∈domain(fst(Net(𝔉))) and
      reg: ∀m∈domain(fst(Net(𝔉))). ⟨t,m⟩∈snd(Net(𝔉)) ⟶ fst(Net(𝔉))m∈U
        using assms net˙of˙filter˙is˙net NetConverges˙def by blast
    with assms(1) uu obtain t1 t2 where t˙def: t=⟨t1,t2⟩ and t1∈t2 and tFF: t2∈𝔉
      using NetOfFilter˙def by auto
    {
      fix s
      assume s∈t2
      then have ⟨s,t2⟩∈–⟨q1,q2⟩∈⋃𝔉×𝔉. q1∈q2″ using tFF by auto
      moreover
       from assms(1) uu have domain(fst(Net(𝔉)))=–⟨q1,q2⟩∈⋃𝔉×𝔉. q1∈q2″ using
NetOfFilter˙def
        by auto
      ultimately
      have tt: ⟨s,t2⟩∈domain(fst(Net(𝔉))) by auto
      moreover
       from assms(1) uu t t˙def tt have ⟨⟨t1,t2⟩,⟨s,t2⟩⟩∈snd(Net(𝔉)) using NetOfFil-
ter˙def
        by auto
      ultimately
      have fst(Net(𝔉))⟨s,t2⟩∈U using reg t˙def by auto
      moreover
     from assms(1) uu have function(fst(Net(𝔉))) using NetOfFilter˙def function˙def
        by auto
      moreover
      from tt assms(1) uu have ⟨⟨s,t2⟩,s⟩∈fst(Net(𝔉)) using NetOfFilter˙def by auto
      ultimately
      have s∈U using NetOfFilter˙def function˙apply˙equality by auto
    }
    then have t2⊆U by auto
    with tFF assms(1) ‹U∈Pow(⋃T)› have U∈𝔉 using IsFilter˙def by auto
  }
  then have –U∈Pow(⋃T). x∈int(U)″ ⊆ 𝔉 by auto
  ultimately
  show thesis using FilterConverges˙def assms(1) by auto
qed
```

A filter converges to a point if and only if its net converges to the point.

```
theorem (in topology0) filter˙conver˙iff˙net˙of˙filter˙conver:
  assumes 𝔉 –is a filter on″⋃T
  shows (𝔉 →F x) ⟷ ((Net(𝔉)) →N x)
  using filter˙conver˙net˙of˙filter˙conver net˙of˙filter˙conver˙filter˙conver assms
    by auto
```

The previous result states that, when considering convergence, the filters do not generalize nets. When considering a filter, there is always a net that converges to the same points of the original filter.

Now we see that with nets, results come naturally applying the axiom of choice; but with filters, the results come, may be less natural, but with no choice. The reason is that Net(𝔉) is a net that doesn't come into our attention as a first choice; maybe because we restrict ourselves to the anti-symmetry property of orders without realizing that a directed set is not an order.

The following results will state that filters are not just a subclass of nets, but that nets and filters are equivalent on convergence: for every filter there is a net converging to the same points, and also, for every net there is a filter converging to the same points.

```
definition
  FilterOfNet (Filter (˙ .. ˙) 40) where
  (N –is a net on″ X) ⟹ Filter N..X ≡ –A∈Pow(X). ∃D∈—fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)).
s∈snd(N) ∧ fst(s)=t0″″. t0∈domain(fst(N))″. D⊆A″
```

Filter of a net is indeed a filter

```
theorem filter˙of˙net˙is˙filter:
  assumes N –is a net on″ X
  shows (Filter N..X) –is a filter on″ X and
    —fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=t0″″.
t0∈domain(fst(N))″ –is a base filter″ (Filter N..X)
proof -
  let C = —fst(N)(snd(s)). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=t0″″.
t0∈domain(fst(N))″
  have C⊆Pow(X)
  proof -
    {
      fix t
      assume t∈C
      then obtain t1 where t1∈domain(fst(N)) and
         t˙Def: t=–fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧
fst(s)=t1″″
        by auto
      {
        fix x
        assume x∈t
       with t˙Def obtain ss where ss∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=t1″ and
```

```
        x˙def: x = fst(N)(snd(ss)) by blast
      then have snd(ss) ∈ domain(fst(N)) by auto
      from assms have fst(N):domain(fst(N))→X unfolding IsNet˙def by simp
        with ‹snd(ss) ∈ domain(fst(N))› have x∈X using apply˙funtype x˙def
      by auto
    }
    hence t⊆X by auto
  }
  thus thesis by blast
qed
have sat: C –satisfies the filter base condition″
proof -
  from assms obtain t1 where t1∈domain(fst(N)) using IsNet˙def by blast
 hence –fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=t1″″∈C

    by auto
  hence C≠0 by auto
  moreover
  {
    fix U
    assume U∈C
    then obtain q where q˙dom: q∈domain(fst(N)) and
      U˙def: U=–fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧
fst(s)=q″″
      by blast
     with assms have ⟨q,q⟩∈snd(N) ∧ fst(⟨q,q⟩)=q unfolding IsNet˙def IsDirected-
Set˙def refl˙def
      by auto
  with q˙dom have ⟨q,q⟩∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=q″

      by auto
    with U˙def have fst(N)(snd(⟨q,q⟩)) ∈ U by blast
    hence U≠0 by auto
  }
  then have 0∉C by auto
  moreover
  have ∀A∈C. ∀B∈C. (∃D∈C. D⊆A∩B)
  proof
    fix A
    assume pA: A∈C
    show ∀B∈C. ∃D∈C. D⊆A∩B
    proof
      {
        fix B
        assume B∈C
        with pA obtain qA qB where per: qA∈domain(fst(N)) qB∈domain(fst(N))
and
         A˙def: A=–fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=qA″″ and
```

B˙def: B=–fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=qB″″
by blast
have dir: snd(N) directs domain(fst(N)) using assms IsNet˙def by auto
with per obtain qD where ine: ⟨qA,qD⟩∈snd(N) ⟨qB,qD⟩∈snd(N) and perD: qD∈domain(fst(N)) unfolding IsDirectedSet˙def
by blast
let D = –fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=qD″″
from perD have D∈C by auto
moreover
{
fix d
assume d∈D
then obtain sd where sd∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=qD″ and
d˙def: d=fst(N)snd(sd) by blast
then have sdN: sd∈snd(N) and qdd: fst(sd)=qD and sd∈domain(fst(N))×domain(fst(N))

by auto
then obtain qI aa where sd = ⟨aa,qI⟩ qI ∈ domain(fst(N)) aa ∈ domain(fst(N))

by auto
with qdd have sd˙def: sd=⟨qD,qI⟩ and qIdom: qI∈domain(fst(N)) by auto
with sdN have ⟨qD,qI⟩∈snd(N) by auto
from dir have trans(snd(N)) unfolding IsDirectedSet˙def by auto
then have ⟨qA,qD⟩∈snd(N) ∧ ⟨qD,qI⟩∈snd(N) ⟶ ⟨qA,qI⟩∈snd(N) and
⟨qB,qD⟩∈snd(N) ∧ ⟨qD,qI⟩∈snd(N)⟶⟨qB,qI⟩∈snd(N)
using trans˙def by auto
with ine ‹⟨qD,qI⟩∈snd(N)› have ⟨qA,qI⟩∈snd(N) ⟨qB,qI⟩∈snd(N) by auto
with qIdom per have ⟨qA,qI⟩∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=qA″
⟨qB,qI⟩∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=qB″
by auto
then have fst(N)(qI) ∈ A∩B using A˙def B˙def by auto
then have fst(N)(snd(sd)) ∈ A∩B using sd˙def by auto
then have d ∈ A∩B using d˙def by auto
}
then have D ⊆ A∩B by blast
ultimately show ∃D∈C. D⊆A∩B by blast
}
qed
qed
ultimately
show thesis unfolding SatisfiesFilterBase˙def by blast
qed
have
Base: C –is a base filter″ –A∈Pow(X). ∃D∈C. D⊆A″ ⋃–A∈Pow(X). ∃D∈C. D⊆A″=X

```
  proof -
    from ‹C⊆Pow(X)› sat show C –is a base filter″ –A∈Pow(X). ∃D∈C. D⊆A″
      by (rule base˙unique˙filter˙set3)
    from ‹C⊆Pow(X)› sat show ⋃–A∈Pow(X). ∃D∈C. D⊆A″=X
      by (rule base˙unique˙filter˙set3)
  qed
  with sat show (Filter N..X) –is a filter on″ X
    using sat basic˙filter FilterOfNet˙def assms by auto
  from Base(1) show C –is a base filter″ (Filter N..X)
    using FilterOfNet˙def assms by auto
qed
```

Convergence of a net implies the convergence of the corresponding filter.

```
theorem (in topology0) net˙conver˙filter˙of˙net˙conver:
  assumes N –is a net on″ ⋃T and N →_N x
  shows (Filter N..(⋃T)) →_F x
proof -
 let C = —fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=t″″.

      t∈domain(fst(N))″
  from assms(1) have
    (Filter N..(⋃T)) –is a filter on″ (⋃T) and C –is a base filter″ Filter N..(⋃T)
    using filter˙of˙net˙is˙filter by auto
  moreover have ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U)
  proof -
    {
      fix U
      assume U∈Pow(⋃T) x∈int(U)
     with assms have ∃t∈domain(fst(N)). (∀m∈domain(fst(N)). (⟨t,m⟩∈snd(N) ⟶
fst(N)m∈U))
        using NetConverges˙def by auto
        then obtain t where t∈domain(fst(N)) and
          reg: ∀m∈domain(fst(N)). (⟨t,m⟩∈snd(N) ⟶ fst(N)m∈U) by auto
      {
        fix f
        assume f∈–fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧
fst(s)=t″″
          then obtain s where s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧
fst(s)=t″ and
          f˙def: f=fst(N)snd(s) by blast
        hence s∈domain(fst(N))×domain(fst(N)) and s∈snd(N) and fst(s)=t
          by auto
        hence s=⟨t,snd(s)⟩ and snd(s)∈domain(fst(N)) by auto
        with ‹s∈snd(N)› reg have fst(N)snd(s)∈U by auto
        with f˙def have f∈U by auto
      }
    hence –fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=t″″
⊆ U
        by blast
```

```
    with ‹t∈domain(fst(N))› have ∃D∈C. D⊆U
      by auto
  } thus ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U)  by auto
 qed
 moreover from assms have x∈⋃T using NetConverges˙def by auto
 ultimately show (Filter N..(⋃T)) →F x by (rule convergence˙filter˙base2)
qed
```

Convergence of a filter corresponding to a net implies convergence of the net.

```
theorem (in topology0) filter˙of˙net˙conver˙net˙conver:
 assumes N –is a net on″ ⋃T and (Filter N..(⋃T)) →F x
 shows N →N x
proof -
 let C = —fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=t″″.

    t∈domain(fst(N))″
 from assms have I: (Filter N..(⋃T)) –is a filter on″ (⋃T)
   C –is a base filter″ (Filter N..(⋃T)) (Filter N..(⋃T)) →F x
   using filter˙of˙net˙is˙filter by auto
 then have reg: ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U)
   by (rule convergence˙filter˙base1)
 from I have x∈⋃T by (rule convergence˙filter˙base1)
 moreover
 {
   fix U
   assume U∈Pow(⋃T) x∈int(U)
   with reg have ∃D∈C. D⊆U by auto
   then obtain D where D∈C D⊆U
     by auto
   then obtain td where td∈domain(fst(N)) and
      D˙def: D=–fst(N)snd(s). s∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧
fst(s)=td″″
     by auto
   {
     fix m
     assume m∈domain(fst(N)) ⟨td,m⟩∈snd(N)
     with ‹td∈domain(fst(N))› have
       ⟨td,m⟩∈–s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=td″
       by auto
     with D˙def have fst(N)m∈D by auto
     with ‹D⊆U› have fst(N)m∈U by auto
   }
   then have ∀m∈domain(fst(N)). ⟨td,m⟩∈snd(N) ⟶ fst(N)m∈U by auto
   with ‹td∈domain(fst(N))› have
     ∃t∈domain(fst(N)). ∀m∈domain(fst(N)). ⟨t,m⟩∈snd(N) ⟶ fst(N)m∈U
     by auto
 }
 then have
```

```
    ∀U∈Pow(⋃T). x∈int(U) ⟶
      (∃t∈domain(fst(N)). ∀m∈domain(fst(N)). ⟨t,m⟩∈snd(N) ⟶ fst(N)m∈U)
      by auto
  ultimately show thesis using NetConverges˙def assms(1) by auto
qed
```

Filter of net converges to a point x if and only the net converges to x.

```
theorem (in topology0) filter˙of˙net˙conv˙iff˙net˙conv:
  assumes N –is a net on″ ⋃T
  shows ((Filter N..(⋃T)) →F x) ⟷ (N →N x)
  using assms filter˙of˙net˙conver˙net˙conver net˙conver˙filter˙of˙net˙conver
    by auto
```

We know now that filters and nets are the same thing, when working convergence of topological spaces. Sometimes, the nature of filters makes it easier to generalized them as follows.

Instead of considering all subsets of some set X, we can consider only open sets (we get an open filter) or closed sets (we get a closed filter). There are many more useful examples that characterize topological properties.

This type of generalization cannot be done with nets.

Also a filter can give us a topology in the following way:

```
theorem top˙of˙filter:
  assumes 𝔉 –is a filter on″ ⋃𝔉
  shows (𝔉 ∪ –0″) –is a topology″
proof -
  {
    fix A B
    assume A∈(𝔉 ∪ –0″)B∈(𝔉 ∪ –0″)
    then have (A∈𝔉 ∧ B∈𝔉) ∨ (A∩B=0) by auto
    with assms have A∩B∈(𝔉 ∪ –0″) unfolding IsFilter˙def
      by blast
  }
  then have ∀A∈(𝔉 ∪ –0″). ∀B∈(𝔉 ∪ –0″). A∩B∈(𝔉 ∪ –0″) by auto
  moreover
  {
    fix M
    assume A:M∈Pow(𝔉 ∪ –0″)
    then have M=0∨M=–0″∨(∃T∈M. T∈𝔉) by blast
    then have ⋃M=0∨(∃T∈M. T∈𝔉) by auto
    then obtain T where ⋃M=0∨(T∈𝔉 ∧ T∈M) by auto
    then have ⋃M=0∨(T∈𝔉 ∧ T⊆⋃M) by auto
    moreover from this A have ⋃M⊆⋃𝔉 by auto
    ultimately have ⋃M∈(𝔉∪–0″) using IsFilter˙def assms by auto
    }
  then have ∀M∈Pow(𝔉∪–0″). ⋃M∈(𝔉∪–0″) by auto
  ultimately show thesis using IsATopology˙def by auto
qed
```

We can use topology0 locale with filters.

```
lemma topology0˙filter:
  assumes 𝔉 –is a filter on″ ⋃𝔉
  shows topology0(𝔉 ∪ –0″)
  using top˙of˙filter topology0˙def assms by auto
```

The next abbreviation introduces notation where we want to specify the space where the filter convergence takes place.

```
abbreviation FilConvTop(˙ →F ˙ –in″ ˙)
  where 𝔉 →F x –in″ T ≡ topology0.FilterConverges(T,𝔉,x)
```

The next abbreviation introduces notation where we want to specify the space where the net convergence takes place.

```
abbreviation NetConvTop(˙ →N ˙ –in″ ˙)
  where N →N x –in″ T ≡ topology0.NetConverges(T,N,x)
```

Each point of a the union of a filter is a limit of that filter.

```
lemma lim˙filter˙top˙of˙filter:
  assumes 𝔉 –is a filter on″ ⋃𝔉 and x∈⋃𝔉
  shows 𝔉 →F x –in″ (𝔉∪–0″)
proof-
  have ⋃𝔉=⋃(𝔉∪–0″) by auto
  with assms(1) have assms1: 𝔉 –is a filter on″ ⋃(𝔉∪–0″) by auto
  {
    fix U
    assume U∈Pow(⋃(𝔉∪–0″)) x∈Interior(U,(𝔉∪–0″))
    with assms(1) have Interior(U,(𝔉∪–0″))∈𝔉 using topology0˙def top˙of˙filter
      topology0.Top˙2˙L2 by blast
    moreover
    from assms(1) have Interior(U,(𝔉∪–0″))⊆U using topology0˙def top˙of˙filter
      topology0.Top˙2˙L1 by auto
    moreover
    from ‹U∈Pow(⋃(𝔉∪–0″))› have U∈Pow(⋃𝔉) by auto
    ultimately have U∈𝔉 using assms(1) IsFilter˙def by auto
  }
  with assms assms1 show thesis using topology0.FilterConverges˙def top˙of˙filter
    topology0˙def by auto
qed

end
```

56 Topology and neighborhoods

```
theory Topology˙ZF˙4a imports Topology˙ZF˙4
begin
```

This theory considers the relations between topology and systems of neighborhood filters.

56.1 Neighborhood systems

The standard way of defining a topological space is by specifying a collection of sets that we consider "open" (see the Topology˙ZF theory). An alternative of this approach is to define a collection of neighborhoods for each point of the space.

We define a neighborhood system as a function that takes each point $x \in X$ and assigns it a collection of subsets of X which is called the neighborhoods of x. The neighborhoods of a point x form a filter that satisfies an additional axiom that for every neighborhood N of x we can find another one U such that N is a neighborhood of every point of U.

```
definition
  IsNeighSystem (˙ –is a neighborhood system on″ ˙ 90)
  where 𝓜 –is a neighborhood system on″ X ≡ (𝓜 : X→Pow(Pow(X))) ∧
 (∀x∈X. (𝓜(x) –is a filter on″ X) ∧ (∀N∈𝓜(x). x∈N ∧ (∃U∈𝓜(x).∀y∈U.(N∈𝓜(y))
) ))
```

A neighborhood system on X consists of collections of subsets of X.

```
lemma neighborhood˙subset:
  assumes 𝓜 –is a neighborhood system on″ X and x∈X and N∈𝓜(x)
  shows N⊆X and x∈N
proof -
  from ‹𝓜 –is a neighborhood system on″ X› have 𝓜 : X→Pow(Pow(X))
    unfolding IsNeighSystem˙def by simp
  with ‹x∈X› have 𝓜(x) ∈ Pow(Pow(X)) using apply˙funtype by blast
  with ‹N∈𝓜(x)› show N⊆X by blast
  from assms show x∈N using IsNeighSystem˙def by simp
qed
```

Some sources (like Wikipedia) use a bit different definition of neighborhood systems where the U is required to be contained in N. The next lemma shows that this stronger version can be recovered from our definition.

```
lemma neigh˙def˙stronger:
  assumes 𝓜 –is a neighborhood system on″ X and x∈X and N∈𝓜(x)
  shows ∃U∈𝓜(x).U⊆N ∧ (∀y∈U.(N∈𝓜(y)))
proof -
  from assms obtain W where W∈𝓜(x) and areNeigh:∀y∈W.(N∈𝓜(y))
    using  IsNeighSystem˙def by blast
  let U = N∩W
  from assms ‹W∈𝓜(x)› have U ∈ 𝓜(x)
    unfolding IsNeighSystem˙def IsFilter˙def by blast
  moreover have U⊆N by blast
  moreover from areNeigh have ∀y∈U.(N∈𝓜(y)) by auto
  ultimately show thesis by auto
qed
```

56.2 Topology from neighborhood systems

Given a neighborhood system $\{\mathcal{M}_x\}_{x\in X}$ we can define a topology on X. Namely, we consider a subset of X open if $U \in \mathcal{M}_x$ for every element x of U.

The collection of sets defined as above is indeed a topology.

```
theorem topology˙from˙neighs:
  assumes 𝓜 –is a neighborhood system on″ X
  defines Tdef: T ≡ –U∈Pow(X). ∀x∈U. U ∈ 𝓜(x)″
  shows T –is a topology″ and ⋃T = X
proof -
  { fix 𝔘 assume 𝔘 ∈ Pow(T)
    have ⋃𝔘 ∈ T
    proof -
      from ‹𝔘 ∈ Pow(T)› Tdef have ⋃𝔘 ∈ Pow(X) by blast
      moreover
      { fix x assume x ∈ ⋃𝔘
        then obtain U where U∈𝔘 and x∈U by blast
        with assms ‹𝔘 ∈ Pow(T)›
        have U ∈ 𝓜(x) and U ⊆ ⋃𝔘 and  𝓜(x) –is a filter on″ X
          unfolding IsNeighSystem˙def by auto
        with ‹⋃𝔘 ∈ Pow(X)› have ⋃𝔘 ∈ 𝓜(x) unfolding IsFilter˙def
          by simp
      }
      ultimately show ⋃𝔘 ∈ T using Tdef by blast
    qed
  }
  moreover
  { fix U V assume U∈T and V∈T
    have U∩V ∈ T
    proof -
      from Tdef ‹U∈T› ‹U∈T› have U∩V ∈ Pow(X) by auto
      moreover
      { fix x assume x ∈ U∩V
        with assms ‹U∈T› ‹V∈T› Tdef have U ∈ 𝓜(x) V ∈ 𝓜(x) and  𝓜(x) –is a
filter on″ X
          unfolding IsNeighSystem˙def by auto
        then have U∩V ∈ 𝓜(x) unfolding IsFilter˙def by simp
      }
      ultimately show U∩V ∈T using Tdef by simp
    qed
  }
  ultimately show T –is a topology″ unfolding IsATopology˙def by blast
  from assms show ⋃T = X unfolding IsNeighSystem˙def IsFilter˙def by blast
qed
```

Some sources (like Wikipedia) define the open sets generated by a neighborhood system "as those sets containing a neighborhood of each of their

points". The next lemma shows that this definition is equivalent to the one we are using.

```
lemma topology˙from˙neighs1:
  assumes 𝓜 –is a neighborhood system on″ X
  shows –U∈Pow(X). ∀x∈U. U ∈ 𝓜(x)″ = –U∈Pow(X). ∀x∈U. ∃V ∈ 𝓜(x). V⊆U″
proof
  let T = –U∈Pow(X). ∀x∈U. U ∈ 𝓜(x)″
  let S = –U∈Pow(X). ∀x∈U. ∃V ∈ 𝓜(x). V⊆U″
  show S⊆T
  proof -
    { fix U assume U∈S
      then have U∈Pow(X) by simp
      moreover
      from assms ‹U∈S› ‹U∈Pow(X)› have ∀x∈U. U ∈ 𝓜(x)
        unfolding IsNeighSystem˙def IsFilter˙def by blast
      ultimately have U∈T by auto
    } thus thesis by auto
  qed
  show T⊆S by auto
qed
```

56.3 Neighborhood system from topology

Once we have a topology T we can define a natural neighborhood system on $X = \bigcup T$. In this section we define such neighborhood system and prove its basic properties.

For a topology T we define a neighborhood system of T as a function that takes an $x \in X = \bigcup T$ and assigns it a collection supersets of open sets containing x. We call that the "neighborhood system of T"

```
definition
  NeighSystem (–neighborhood system of″ ˙ 91)
  where –neighborhood system of″ T ≡ – ⟨x,–V∈Pow(⋃T).∃U∈T.(x∈U ∧ U⊆V)″⟩.
x ∈ ⋃T ″
```

The next lemma shows that open sets are members of (what we will prove later to be) the natural neighborhood system on $X = \bigcup T$.

```
lemma open˙are˙neighs:
  assumes U∈T x∈U
  shows x ∈ ⋃T and U ∈ –V∈Pow(⋃T).∃U∈T.(x∈U ∧ U⊆V)″
  using assms by auto
```

Another fact we will need is that for every $x \in X = \bigcup T$ the neighborhoods of x form a filter

```
lemma neighs˙is˙filter:
  assumes T –is a topology″ and x ∈ ⋃T
  defines Mdef: 𝓜 ≡ –neighborhood system of″ T
```

```
  shows 𝓜(x) –is a filter on″ (⋃T)
proof -
  let X = ⋃T
  let 𝔉 = –V∈Pow(X).∃U∈T.(x∈U ∧ U⊆V)″
  have 0∉𝔉 by blast
  moreover have X∈𝔉
  proof -
    from assms ‹x∈X› have X ∈ Pow(X) X∈T and x∈X ∧ X⊆X using carr˙open
      by auto
    hence ∃U∈T.(x∈U ∧ U⊆X) by auto
    thus thesis by auto
  qed
  moreover have ∀A∈𝔉. ∀B∈𝔉. A∩B ∈ 𝔉
  proof -
    { fix A B assume A∈𝔉 B∈𝔉
      then obtain U_A U_B where U_A∈T x∈U_A U_A⊆A U_B∈T x∈U_B U_B⊆B
        by auto
      with ‹T –is a topology″› ‹A∈𝔉› ‹B∈𝔉› have A∩B ∈ Pow(X) and
        U_A∩U_B ∈ T x ∈ U_A∩U_B U_A∩U_B ⊆ A∩B using IsATopology˙def
        by auto
      hence A∩B ∈ 𝔉 by blast
    } thus thesis by blast
  qed
  moreover have ∀B∈𝔉. ∀C∈Pow(X). B⊆C ⟶ C∈𝔉
  proof -
    { fix B C assume B∈𝔉 C ∈ Pow(X) B⊆C
      then obtain U where U∈T and x∈U U⊆B by blast
      with ‹C ∈ Pow(X)› ‹B⊆C› have C∈𝔉 by blast
    } thus thesis by auto
  qed
  ultimately have 𝔉 –is a filter on″ X unfolding IsFilter˙def by blast
  with Mdef ‹x∈X› show 𝓜(x) –is a filter on″ X using ZF˙fun˙from˙tot˙val1 Neigh-
System˙def
    by simp
qed
```

The next theorem states that the the natural neighborhood system on $X = \bigcup T$ indeed is a neighborhood system.

```
theorem neigh˙from˙topology:
  assumes T –is a topology″
  shows (–neighborhood system of″ T) –is a neighborhood system on″ (⋃T)
proof -
  let X = ⋃T
  let 𝓜 = –neighborhood system of″ T
  have 𝓜 : X→Pow(Pow(X))
  proof -
    { fix x assume x∈X
      hence –V∈Pow(⋃T).∃U∈T.(x∈U ∧ U⊆V)″ ∈ Pow(Pow(X)) by auto
    } hence ∀x∈X. –V∈Pow(⋃T).∃U∈T.(x∈U ∧ U⊆V)″ ∈ Pow(Pow(X)) by auto
```

```
    then show thesis using ZF˙fun˙from˙total NeighSystem˙def by simp
  qed
  moreover from assms have ∀x∈X. (𝓜(x) –is a filter on" X)
    using neighs˙is˙filter NeighSystem˙def by auto
  moreover have ∀x∈X. ∀N∈𝓜(x). x∈N ∧ (∃U∈𝓜(x).∀y∈U.(N∈𝓜(y)))
  proof -
    { fix x N assume x∈X N ∈ 𝓜(x)
      let 𝔉 = –V∈Pow(X).∃U∈T.(x∈U ∧ U⊆V)"
      from ‹x∈X› have 𝓜(x) = 𝔉 using ZF˙fun˙from˙tot˙val1 NeighSystem˙def
        by simp
      with ‹N ∈ 𝓜(x)› have N∈𝔉 by simp
      hence x∈N by blast
      from ‹N∈𝔉› obtain U where U∈T x∈U and U⊆N by blast
      with ‹N∈𝔉› ‹𝓜(x) = 𝔉› have U ∈ 𝓜(x) by auto
      moreover from assms ‹U∈T› ‹U⊆N› ‹N∈𝔉› have ∀y∈U.(N ∈ 𝓜(y))
        using ZF˙fun˙from˙tot˙val1 open˙are˙neighs neighs˙is˙filter
              NeighSystem˙def IsFilter˙def by auto
      ultimately have ∃U∈𝓜(x).∀y∈U.(N∈𝓜(y)) by blast
      with ‹x∈N› have x∈N ∧ (∃U∈𝓜(x).∀y∈U.(N∈𝓜(y))) by simp
    } thus thesis by auto
  qed
  ultimately show thesis unfolding IsNeighSystem˙def by blast
qed

end
```

57 Topology - examples

theory Topology˙ZF˙examples imports Topology˙ZF Cardinal˙ZF

begin

This theory deals with some concrete examples of topologies.

57.1 CoCardinal Topology

In this section we define and prove the basic properties of the co-cardinal topology on a set X.

The collection of subsets of a set whose complement is strictly bounded by a cardinal is a topology given some assumptions on the cardinal.

definition
CoCardinal(X,T) ≡ –F∈Pow(X). X-F ≺ T"∪ –0"

For any set and any infinite cardinal we prove that CoCardinal(X,Q) forms a topology. The proof is done with an infinite cardinal, but it is obvious that the set Q can be any set equipollent with an infinite cardinal. It is a topology also if the set where the topology is defined is too small or the

cardinal too large; in this case, as it is later proved the topology is a discrete topology. And the last case corresponds with Q=1 which translates in the indiscrete topology.

```
lemma CoCar˙is˙topology:
  assumes InfCard (Q)
  shows CoCardinal(X,Q) –is a topology″
proof -
  let T = CoCardinal(X,Q)
  {
    fix M
    assume A:M∈Pow(T)
    hence M⊆T by auto
    then have M⊆Pow(X) using CoCardinal˙def by auto
    then have ⋃M∈Pow(X) by auto
    moreover
    {
      assume B:M=0
      then have ⋃M∈T using CoCardinal˙def by auto
    }
    moreover
    {
      assume B:M=–0″
      then have ⋃M∈T using CoCardinal˙def by auto
    }
    moreover
    {
      assume B:M ≠0 M≠–0″
      from B obtain T where C:T∈M and T≠0 by auto
      with A have D:X-T ≺ (Q) using CoCardinal˙def by auto
      from C have X-⋃M⊆X-T by blast
      with D have X-⋃M≺ (Q) using subset˙imp˙lepoll lesspoll˙trans1 by blast
    }
    ultimately have ⋃M∈T using CoCardinal˙def by auto
  }
  moreover
  {
    fix U and V
    assume U∈T and V∈T
    then have A:U=0 ∨ (U∈Pow(X) ∧ X-U≺ (Q)) and
      B:V=0 ∨ (V∈Pow(X) ∧ X-V≺ (Q)) using CoCardinal˙def by auto
    hence D:U∈Pow(X)V∈Pow(X) by auto
    have C:X-(U ∩ V)=(X-U)∪(X-V) by fast
   with A B C have U∩V=0∨(U∩V∈Pow(X) ∧ X-(U ∩ V)≺ (Q)) using less˙less˙imp˙un˙less
assms
      by auto
    then have U∩V∈T using CoCardinal˙def by auto
  }
  ultimately show thesis using IsATopology˙def by auto
qed
```

We can use theorems proven in topology0 context for the co-cardinal topology.

```
theorem topology0˙CoCardinal:
  assumes InfCard(T)
  shows topology0(CoCardinal(X,T))
  using topology0˙def CoCar˙is˙topology assms by auto
```

It can also be proven that if CoCardinal(X,T) is a topology, X≠0, Card(T) and T≠0; then T is an infinite cardinal, X≺T or T=1. It follows from the fact that the union of two closed sets is closed. Choosing the appropriate cardinals, the cofinite and the cocountable topologies are obtained.

The cofinite topology is a very special topology because it is closely related to the separation axiom T_1. It also appears naturally in algebraic geometry.

```
definition
  Cofinite (CoFinite ˙ 90) where
  CoFinite X ≡ CoCardinal(X,nat)
```

Cocountable topology in fact consists of the empty set and all cocountable subsets of X.

```
definition
  Cocountable (CoCountable ˙ 90) where
  CoCountable X ≡ CoCardinal(X,csucc(nat))
```

57.2 Total set, Closed sets, Interior, Closure and Boundary

There are several assertions that can be done to the CoCardinal(X,T) topology. In each case, we will not assume sufficient conditions for CoCardinal(X,T) to be a topology, but they will be enough to do the calculations in every posible case.

The topology is defined in the set X

```
lemma union˙cocardinal:
  assumes T≠0
  shows ⋃ CoCardinal(X,T) = X
proof-
  have X:X-X=0 by auto
  have 0 ≲ 0 by auto
  with assms have 0≺11 ≲T using not˙0˙is˙lepoll˙1 lepoll˙imp˙lesspoll˙succ by auto
  then have 0≺T using lesspoll˙trans2  by auto
  with X have (X-X)≺T by auto
  then have X∈CoCardinal(X,T) using CoCardinal˙def by auto
  hence X⊆⋃ CoCardinal(X,T) by blast
  then show ⋃ CoCardinal(X,T)=X using CoCardinal˙def by auto
qed
```

The closed sets are the small subsets of X and X itself.

```
lemma closed˙sets˙cocardinal:
  assumes T≠0
  shows D –is closed in″ CoCardinal(X,T) ⟷ (D∈Pow(X) ∧ D≺T) ∨ D=X
proof-
  {
    assume A:D ⊆ X X - D ∈ CoCardinal(X,T)  D ≠ X
    from A(1,3) have X-(X-D)=D X-D≠0 by auto
    with A(2) have D≺T using CoCardinal˙def by simp
  }
  with assms have D –is closed in″ CoCardinal(X,T) ⟶ (D∈Pow(X) ∧  D≺T)∨
D=X using IsClosed˙def
    union˙cocardinal by auto
  moreover
  {
    assume A:D ≺ TD ⊆ X
    from A(2) have X-(X-D)=D by blast
    with A(1) have X-(X-D)≺ T by auto
    then have X-D∈ CoCardinal(X,T) using CoCardinal˙def by auto
  }
  with assms have (D∈Pow(X) ∧  D≺T)⟶ D –is closed in″ CoCardinal(X,T) using
union˙cocardinal
    IsClosed˙def by auto
  moreover
  have X-X=0 by auto
  then have X-X∈ CoCardinal(X,T)using CoCardinal˙def by auto
  with assms have X–is closed in″ CoCardinal(X,T) using union˙cocardinal
    IsClosed˙def by auto
  ultimately show thesis by auto
qed
```

The interior of a set is itself if it is open or 0 if it isn't open.

```
lemma interior˙set˙cocardinal:
  assumes noC: T≠0 and A⊆X
  shows Interior(A,CoCardinal(X,T))= (if ((X-A) ≺ T) then A else 0)
proof-
  from assms(2) have dif˙dif:X-(X-A)=A by blast
  {
    assume (X-A) ≺ T
    then have (X-A)∈Pow(X) ∧ (X-A) ≺ T by auto
   with noC have (X-A) –is closed in″ CoCardinal(X,T) using closed˙sets˙cocardinal
      by auto
    with noC have X-(X-A)∈CoCardinal(X,T) using IsClosed˙def union˙cocardinal
      by auto
    with dif˙dif have A∈CoCardinal(X,T) by auto
    hence A∈–U∈CoCardinal(X,T). U ⊆ A″ by auto
    hence a1:A⊆⋃–U∈CoCardinal(X,T). U ⊆ A″ by auto
    have a2:⋃–U∈CoCardinal(X,T). U ⊆ A″⊆A by blast
    from a1 a2 have Interior(A,CoCardinal(X,T))=A using Interior˙def by auto}
  moreover
```

```
  {
    assume as:˜((X-A) ≺ T)
    {
      fix U
      assume U ⊆A
      hence X-A ⊆ X-U by blast
      then have Q:X-A ≲ X-U using subset˙imp˙lepoll by auto
      {
        assume X-U≺ T
        with Q have X-A≺ T using lesspoll˙trans1 by auto
        with as have False  by auto
      }
      hence ˜((X-U) ≺ T) by auto
      then have U∉CoCardinal(X,T)∨U=0 using CoCardinal˙def by auto
    }
    hence –U∈CoCardinal(X,T). U ⊆ A″⊆–0″ by blast
    then have Interior(A,CoCardinal(X,T))=0 using Interior˙def by auto
  }
  ultimately show thesis by auto
qed
```

X is a closed set that contains A. This lemma is necessary because we cannot use the lemmas proven in the topology0 context since T≠0″ is too weak for CoCardinal(X,T) to be a topology.

```
lemma X˙closedcov˙cocardinal:
  assumes T≠0 A⊆X
  shows X∈ClosedCovers(A,CoCardinal(X,T)) using ClosedCovers˙def
  using union˙cocardinal closed˙sets˙cocardinal assms by auto
```

The closure of a set is itself if it is closed or X if it isn't closed.

```
lemma closure˙set˙cocardinal:
  assumes T≠0A⊆X
  shows Closure(A,CoCardinal(X,T))=(if (A ≺ T) then A else X)
proof-
  {
    assume A ≺ T
    with assms have A –is closed in″ CoCardinal(X,T) using closed˙sets˙cocardinal
by auto
   with assms(2) have A∈ –D ∈ Pow(X). D –is closed in″ CoCardinal(X,T) ∧ A⊆D″
by auto
    with assms(1) have S:A∈ClosedCovers(A,CoCardinal(X,T)) using ClosedCov-
ers˙def
      using union˙cocardinal by auto
    hence l1:⋂ClosedCovers(A,CoCardinal(X,T))⊆A by blast
    from S have l2:A ⊆ ⋂ClosedCovers(A,CoCardinal(X,T))
      unfolding  ClosedCovers˙def by auto
    from l1 l2 have Closure(A,CoCardinal(X,T))=A using Closure˙def
      by auto
  }
```

```
  moreover
  {
    assume as:¬ A ≺ T
    {
      fix U
      assume A⊆U
      then have Q:A ≲ U using subset˙imp˙lepoll by auto
      {
        assume U≺ T
        with Q have A≺ T using lesspoll˙trans1 by auto
        with as have False by auto
      }
      hence ¬ U ≺ T by auto
        with assms(1) have ¬(U –is closed in″ CoCardinal(X,T)) ∨ U=X using
closed˙sets˙cocardinal
      by auto
    }
   with assms(1) have ∀ U∈Pow(X). U–is closed in″CoCardinal(X,T) ∧ A⊆U⟶U=X
      by auto
    with assms(1) have ClosedCovers(A,CoCardinal(X,T))⊆–X″
      using union˙cocardinal using ClosedCovers˙def by auto
   with assms have ClosedCovers(A,CoCardinal(X,T))=–X″ using X˙closedcov˙cocardinal
      by auto
    then have  Closure(A,CoCardinal(X,T)) = X  using Closure˙def by auto
  }
  ultimately show thesis by auto
qed
```

The boundary of a set is empty if A and $X - A$ are closed, X if not A neither $X - A$ are closed and; if only one is closed, then the closed one is its boundary.

```
lemma boundary˙cocardinal:
  assumes T≠0A⊆X
  shows Boundary(A,CoCardinal(X,T)) = (if A≺ T then (if  (X-A)≺ T then 0 else
A) else (if  (X-A)≺ T then X-A else X))
proof-
  from assms(2) have X-A ⊆ X by auto
  {
    assume AS: A≺T X-A ≺ T
    with assms ‹X-A ⊆ X› have
      Closure(X-A,CoCardinal(X,T)) = X-A and Closure(A,CoCardinal(X,T)) = A
      using closure˙set˙cocardinal by auto
    with assms(1) have Boundary(A,CoCardinal(X,T)) = 0
      using Boundary˙def  union˙cocardinal by auto
  }
  moreover
  {
    assume AS: ˜(A≺T) X-A ≺ T
    with assms ‹X-A ⊆ X› have
```

```
    Closure(X-A,CoCardinal(X,T)) = X-A and Closure(A,CoCardinal(X,T)) = X
    using closure˙set˙cocardinal by auto
  with assms(1) have Boundary(A,CoCardinal(X,T))=X-A using Boundary˙def
    union˙cocardinal by auto
 }
 moreover
 {
  assume AS:˜(A≺T) ˜(X-A ≺ T)
  with assms ‹X-A ⊆ X› have
    Closure(X-A,CoCardinal(X,T))=X and Closure(A,CoCardinal(X,T))=X
    using closure˙set˙cocardinal by auto
 with assms(1) have Boundary(A,CoCardinal(X,T))=X using Boundary˙def union˙cocardinal
    by auto
 }
 moreover
 {
  assume AS:A≺ T ˜(X-A≺T)
  with assms ‹X-A ⊆ X› have
    Closure(X-A,CoCardinal(X,T))=X and Closure(A,CoCardinal(X,T)) = A
    using closure˙set˙cocardinal by auto
 with assms have Boundary(A,CoCardinal(X,T))=A using Boundary˙def union˙cocardinal
    by auto
 }
 ultimately show thesis by auto
qed
```

If the set is too small or the cardinal too large, then the topology is just the discrete topology.

```
lemma discrete˙cocardinal:
 assumes X≺T
 shows CoCardinal(X,T) = Pow(X)
proof
 {
  fix U
  assume U∈CoCardinal(X,T)
  then have U ∈ Pow(X) using CoCardinal˙def by auto
 }
 then show CoCardinal(X,T) ⊆ Pow(X) by auto
 {
  fix U
  assume A:U ∈ Pow(X)
  then have X-U ⊆ X by auto
  then have X-U ≲X using subset˙imp˙lepoll by auto
  then have X-U≺ T using lesspoll˙trans1 assms by auto
  with A have U∈CoCardinal(X,T) using CoCardinal˙def
    by auto
 }
 then show Pow(X) ⊆ CoCardinal(X,T) by auto
qed
```

If the cardinal is taken as T=1 then the topology is indiscrete.

```
lemma indiscrete˙cocardinal:
  shows CoCardinal(X,1) = –0,X″
proof
  {
    fix Q
    assume Q ∈ CoCardinal(X,1)
    then have Q ∈ Pow(X) and Q=0 ∨ X-Q≺1 using CoCardinal˙def by auto
    then have Q ∈ Pow(X) and Q=0 ∨ X-Q=0 using lesspoll˙succ˙iff lepoll˙0˙iff by
auto
    then have Q=0 ∨ Q=X by blast
  }
  then show CoCardinal(X,1) ⊆ –0, X″ by auto
  have 0 ∈ CoCardinal(X,1) using CoCardinal˙def by auto
  moreover
  have 0≺1 and X-X=0 using lesspoll˙succ˙iff by auto
  then have X∈CoCardinal(X,1) using CoCardinal˙def by auto
  ultimately show –0, X″ ⊆ CoCardinal(X,1)  by auto
qed
```

The topological subspaces of the CoCardinal(X,T) topology are also CoCardinal topologies.

```
lemma subspace˙cocardinal:
  shows CoCardinal(X,T) –restricted to″ Y = CoCardinal(Y∩X,T)
proof
  {
    fix M
    assume M ∈ (CoCardinal(X,T) –restricted to″ Y)
   then obtain A where A1:A ∈ CoCardinal(X,T) M=Y ∩ A using RestrictedTo˙def
by auto
    then have M ∈ Pow(X ∩ Y) using CoCardinal˙def by auto
    moreover
    from A1 have (Y ∩ X)-M = (Y ∩ X)-A using CoCardinal˙def by auto
    with ‹(Y ∩ X)-M = (Y ∩ X)-A› have (Y ∩ X)-M⊆ X-A by auto
    then have (Y ∩ X)-M ≲ X-A using subset˙imp˙lepoll by auto
    with A1 have (Y ∩ X)-M ≺ T ∨ M=0 using lesspoll˙trans1 CoCardinal˙def
      by auto
    ultimately have M ∈ CoCardinal(Y∩X, T) using CoCardinal˙def
      by auto
  }
  then show CoCardinal(X,T) –restricted to″ Y ⊆ CoCardinal(Y∩X,T) by auto
  {
    fix M
    let A = M ∪ (X-Y)
    assume A:M ∈ CoCardinal(Y ∩ X,T)
    {
      assume M=0
      hence M=0 ∩ Y by auto
      then have M∈CoCardinal(X,T) –restricted to″ Y using RestrictedTo˙def
```

```
      CoCardinal˙def by auto
    }
    moreover
    {
      assume AS:M≠0
      from A AS have A1:(M∈Pow(Y ∩ X) ∧ (Y ∩ X)-M≺T) using CoCardinal˙def
by auto
      hence A∈Pow(X) by blast
      moreover
      have X-A=(Y ∩ X)-M by blast
      with A1 have X-A≺ T by auto
      ultimately have A∈CoCardinal(X,T) using CoCardinal˙def by auto
    then have AT:Y ∩ A∈CoCardinal(X,T) –restricted to″ Y using RestrictedTo˙def
        by auto
      have Y ∩ A=Y ∩ M by blast
      also from A1 have ...=M by auto
      finally have Y ∩ A=M by simp
      with AT have M∈CoCardinal(X,T) –restricted to″ Y
        by auto
    }
    ultimately have M∈CoCardinal(X,T) –restricted to″ Y by auto
  }
  then show CoCardinal(Y ∩ X, T) ⊆ CoCardinal(X,T) –restricted to″ Y by auto
qed
```

57.3 Excluded Set Topology

In this section, we consider all the subsets of a set which have empty intersection with a fixed set.

The excluded set topology consists of subsets of X that are disjoint with a fixed set U.

```
definition ExcludedSet(X,U) ≡ –F∈Pow(X). U ∩ F=0″∪ –X″
```

For any set; we prove that ExcludedSet(X,Q) forms a topology.

```
theorem excludedset˙is˙topology:
  shows ExcludedSet(X,Q) –is a topology″
proof-
  {
    fix M
    assume M ∈ Pow(ExcludedSet(X,Q))
    then have A:M⊆–F∈Pow(X). Q ∩ F=0″∪ –X″ using ExcludedSet˙def by auto
    hence ⋃M∈Pow(X) by auto
    moreover
    {
      have B:Q ∩⋃M=⋃–Q ∩T. T∈M″ by auto
      {
        assume X∉M
        with A have M⊆–F∈Pow(X). Q ∩ F=0″ by auto
```

```
      with B have Q ∩ ⋃M=0 by auto
    }
    moreover
    {
      assume X∈M
      with A have ⋃M=X by auto
    }
    ultimately have  Q ∩ ⋃M=0 ∨ ⋃M=X by auto
  }
  ultimately have ⋃M∈ExcludedSet(X,Q) using ExcludedSet˙def by auto
}
moreover
{
  fix U V
  assume U∈ExcludedSet(X,Q) V∈ExcludedSet(X,Q)
  then have U∈Pow(X)V∈Pow(X)U=X∨ U ∩ Q=0V=X∨ V ∩ Q=0 using Exclud-
edSet˙def by auto
  hence U∈Pow(X)V∈Pow(X)(U ∩ V)=X ∨ Q∩(U ∩ V)=0 by auto
  then have (U ∩ V)∈ExcludedSet(X,Q) using ExcludedSet˙def by auto
}
ultimately show thesis using IsATopology˙def by auto
qed
```

We can use topology0 when discussing excluded set topology.

```
theorem topology0˙excludedset:
  shows topology0(ExcludedSet(X,T))
  using topology0˙def excludedset˙is˙topology by auto
```

Choosing a singleton set, it is considered a point in excluded topology.

```
definition
  ExcludedPoint(X,p) ≡ ExcludedSet(X,–p˝)
```

57.4 Total set, closed sets, interior, closure and boundary

Here we discuss what are closed sets, interior, closure and boundary in excluded set topology.

The topology is defined in the set X

```
lemma union˙excludedset:
  shows ⋃ExcludedSet(X,T) = X
proof-
  have X ∈ExcludedSet(X,T) using ExcludedSet˙def by auto
  then show thesis using ExcludedSet˙def by auto
qed
```

The closed sets are those which contain the set (X ∩ T) and 0.

```
lemma closed˙sets˙excludedset:
  shows D –is closed in˝ExcludedSet(X,T) ⟷ (D∈Pow(X) ∧ (X ∩ T) ⊆ D) ∨ D=0
```

```
proof-
  {
    fix x
    assume A:D ⊆ X X-D ∈ExcludedSet(X,T) D≠0 x∈T x∈X
    from A(1) have B:X-(X-D)=D by auto
    from A(2) have T∩(X-D)=0∨ X-D=X using ExcludedSet˙def by auto
    hence T∩(X-D)=0∨ X-(X-D)=X-X by auto
    with B have T∩(X-D)=0∨ D=X-X by auto
    hence T∩(X-D)=0∨ D=0 by auto
    with A(3) have T∩(X-D)=0 by auto
    with A(4) have x∉X-D by auto
    with A(5) have x∈D by auto
  }
  moreover
  {
    assume A:X∩T⊆D D⊆X
    from A(1) have X-D⊆X-(X∩T) by auto
    also have ... = X-T by auto
    finally have T∩(X-D) = 0 by auto
    moreover
    have X-D ∈ Pow(X) by auto
    ultimately have X-D ∈ExcludedSet(X,T) using ExcludedSet˙def by auto
  }
  ultimately show thesis using IsClosed˙def union˙excludedset ExcludedSet˙def
    by auto
qed
```

The interior of a set is itself if it is X or the difference with the set T

```
lemma interior˙set˙excludedset:
  assumes A⊆X
  shows Interior(A,ExcludedSet(X,T)) = (if A=X then X else A-T)
proof-
  {
    assume A:A≠X
    from assms have A-T ∈ExcludedSet(X,T) using ExcludedSet˙def by auto
    then have A-T⊆Interior(A,ExcludedSet(X,T))
    using Interior˙def by auto
    moreover
    {
      fix U
      assume U ∈ExcludedSet(X,T) U⊆A
      then have T∩U=0 ∨ U=XU⊆A using ExcludedSet˙def by auto
      with A assms have T∩U=0U⊆A by auto
      then have U-T=UU-T⊆A-T by auto
      then have U⊆A-T by auto
    }
    then have Interior(A,ExcludedSet(X,T))⊆A-T using Interior˙def by auto
    ultimately have Interior(A,ExcludedSet(X,T))=A-T by auto
  }
```

```
  moreover
  have X∈ExcludedSet(X,T) using ExcludedSet˙def
  union˙excludedset by auto
  then have Interior(X,ExcludedSet(X,T)) = X using topology0.Top˙2˙L3
  topology0˙excludedset by auto
  ultimately show thesis by auto
qed
```

The closure of a set is itself if it is 0 or the union with T.

```
lemma closure˙set˙excludedset:
  assumes A⊆X
  shows Closure(A,ExcludedSet(X,T))=(if A=0 then 0 else A ∪(X∩ T))
proof-
  have 0∈ClosedCovers(0,ExcludedSet(X,T)) using ClosedCovers˙def
    closed˙sets˙excludedset by auto
  then have Closure(0,ExcludedSet(X,T))⊆0 using Closure˙def by auto
  hence Closure(0,ExcludedSet(X,T))=0 by blast
  moreover
  {
    assume A:A≠0
   with assms have (A∪(X∩T)) –is closed in″ExcludedSet(X,T) using closed˙sets˙excludedset

      by blast
     then have (A ∪(X∩ T))∈ –D ∈ Pow(X). D –is closed in″ExcludedSet(X,T) ∧
A⊆D″
    using assms by auto
    then have (A ∪(X∩ T))∈ClosedCovers(A,ExcludedSet(X,T)) unfolding Closed-
Covers˙def
    using union˙excludedset by auto
    then have l1:⋂ClosedCovers(A,ExcludedSet(X,T)) ⊆ (A ∪(X∩ T)) by blast
    {
      fix U
      assume U∈ClosedCovers(A,ExcludedSet(X,T))
      then have U–is closed in″ExcludedSet(X,T) and A⊆U using ClosedCovers˙def
       union˙excludedset by auto
      then have U=0∨(X∩T)⊆U and A⊆U using closed˙sets˙excludedset
       by auto
      with A have (X∩T)⊆UA⊆U by auto
      hence (X∩T)∪A⊆U by auto
    }
    with assms have (A ∪(X∩ T)) ⊆ ⋂ClosedCovers(A,ExcludedSet(X,T))
      using topology0.Top˙3˙L3 topology0˙excludedset union˙excludedset
      by auto
    with l1 have ⋂ClosedCovers(A,ExcludedSet(X,T)) = (A∪(X∩T)) by auto
    then have Closure(A, ExcludedSet(X,T)) = A∪(X∩T) using Closure˙def
      by auto
  }
  ultimately show thesis by auto
qed
```

The boundary of a set is 0 if A is X or 0, and X∩T in other case.

```
lemma boundary˙excludedset:
  assumes A⊆X
  shows Boundary(A,ExcludedSet(X,T)) = (if A=0∨A=X then 0 else X∩T)
proof-
  {
    have Closure(0,ExcludedSet(X,T))=0Closure(X - 0,ExcludedSet(X,T))=X
    using closure˙set˙excludedset by auto
    then have Boundary(0,ExcludedSet(X,T)) = 0using Boundary˙def using
      union˙excludedset assms by auto
  }
  moreover
  {
    have X-X=0 by blast
   then have Closure(X,ExcludedSet(X,T)) = X and Closure(X-X,ExcludedSet(X,T))
= 0
    using closure˙set˙excludedset by auto
    then have Boundary(X,ExcludedSet(X,T)) = 0unfolding Boundary˙def using
      union˙excludedset by auto
  }
  moreover
  {
    assume A≠0 and A≠X
    then have X-A≠0 using assms by auto
    with assms ‹A≠0› ‹A⊆X› have Closure(A,ExcludedSet(X,T)) = A ∪ (X∩T)
      using closure˙set˙excludedset by simp
    moreover
    from ‹A⊆X› have X-A ⊆ X by blast
    with ‹X-A≠0› have Closure(X-A,ExcludedSet(X,T)) = (X-A) ∪ (X∩T)
      using closure˙set˙excludedset by simp
    ultimately have Boundary(A,ExcludedSet(X,T)) = X∩T
      using Boundary˙def union˙excludedset by auto
  }
  ultimately show thesis by auto
qed
```

57.5 Special cases and subspaces

This section provides some miscellaneous facts about excluded set topologies.

The excluded set topology is equal in the sets T and X∩T.

```
lemma smaller˙excludedset:
  shows ExcludedSet(X,T) = ExcludedSet(X,(X∩T))
proof
 show ExcludedSet(X,T) ⊆ ExcludedSet(X, X∩T) and ExcludedSet(X, X∩T) ⊆Ex-
cludedSet(X,T)
    unfolding ExcludedSet˙def by auto
```

qed

If the set which is excluded is disjoint with X, then the topology is discrete.

```
lemma empty˙excludedset:
  assumes T∩X=0
  shows ExcludedSet(X,T) = Pow(X)
proof
  from assms show ExcludedSet(X,T) ⊆ Pow(X)  using smaller˙excludedset Exclud-
edSet˙def
    by auto
  from assms show Pow(X) ⊆ExcludedSet(X,T) unfolding ExcludedSet˙def by blast
qed
```

The topological subspaces of the ExcludedSet X T topology are also ExcludedSet topologies.

```
lemma subspace˙excludedset:
  shows ExcludedSet(X,T) –restricted to˝ Y = ExcludedSet(Y ∩ X, T)
proof
  {
    fix M
    assume M∈(ExcludedSet(X,T) –restricted to˝ Y)
     then obtain A where A1:A:ExcludedSet(X,T) M=Y ∩ A unfolding Restrict-
edTo˙def by auto
    then have M∈Pow(X ∩ Y) unfolding ExcludedSet˙def by auto
    moreover
    from A1 have T∩M=0∨M=Y∩X unfolding ExcludedSet˙def by blast
    ultimately have M ∈ ExcludedSet(Y ∩ X,T) unfolding ExcludedSet˙def
      by auto
  }
  then show ExcludedSet(X,T) –restricted to˝ Y ⊆ ExcludedSet(Y∩X,T) by auto
  {
    fix M
    let A = M  ∪ ((X∩Y-T)-Y)
    assume A:M ∈ ExcludedSet(Y∩X,T)
    {
      assume M = Y ∩ X
     then have M ∈ ExcludedSet(X,T) –restricted to˝ Y unfolding RestrictedTo˙def
        ExcludedSet˙def by auto
    }
    moreover
    {
      assume AS:M≠Y ∩ X
       from A AS have A1:(M∈Pow(Y ∩ X) ∧ T∩M=0) unfolding ExcludedSet˙def
by auto
      then have A∈Pow(X) by blast
      moreover
      have T∩A=T∩M by blast
      with A1 have T∩A=0 by auto
      ultimately have A ∈ExcludedSet(X,T) unfolding ExcludedSet˙def by auto
```

then have AT:Y ∩ A ∈ExcludedSet(X,T) –restricted to″ Y unfolding RestrictedTo˙def
by auto
have Y ∩ A=Y ∩ M by blast
also have ...=M using A1 by auto
finally have Y∩A = M by simp
with AT have M ∈ExcludedSet(X,T) –restricted to″ Y by auto
}
ultimately have M ∈ExcludedSet(X,T) –restricted to″ Y by auto
}
then show ExcludedSet(Y ∩ X,T) ⊆ ExcludedSet(X,T) –restricted to″ Y by auto
qed

57.6 Included Set Topology

In this section we consider the subsets of a set which contain a fixed set. The family defined in this section and the one in the previous section are dual; meaning that the closed set of one are the open sets of the other.

We define the included set topology as the collection of supersets of some fixed subset of the space X.

definition
IncludedSet(X,U) ≡ –F∈Pow(X). U ⊆ F″ ∪ –0″

In the next theorem we prove that IncludedSet X Q forms a topology.

theorem includedset˙is˙topology:
shows IncludedSet(X,Q) –is a topology″
proof-
{
fix M
assume M ∈ Pow(IncludedSet(X,Q))
then have A:M⊆–F∈Pow(X). Q ⊆ F″∪ –0″ using IncludedSet˙def by auto
then have ⋃M∈Pow(X) by auto
moreover
haveQ ⊆⋃M∨ ⋃M=0 using A by blast
ultimately have ⋃M∈IncludedSet(X,Q) using IncludedSet˙def by auto
}
moreover
{
fix U V
assume U∈IncludedSet(X,Q) V∈IncludedSet(X,Q)
then have U∈Pow(X)V∈Pow(X)U=0∨ Q⊆UV=0∨ Q⊆V using IncludedSet˙def by auto
then have U∈Pow(X)V∈Pow(X)(U ∩ V)=0 ∨ Q⊆(U ∩ V) by auto
then have (U ∩ V)∈IncludedSet(X,Q) using IncludedSet˙def by auto
}
ultimately show thesis using IsATopology˙def by auto
qed

We can reference the theorems proven in the topology0 context when discussing the included set topology.

```
theorem topology0˙includedset:
  shows topology0(IncludedSet(X,T))
  using topology0˙def includedset˙is˙topology by auto
```

Choosing a singleton set, it is considered a point excluded topology. In the following lemmas and theorems, when neccessary it will be considered that T≠0 and T⊆X. These cases will appear in the special cases section.

```
definition
  IncludedPoint (IncludedPoint ˙ ˙ 90) where
  IncludedPoint X p ≡ IncludedSet(X,–p″)
```

57.7 Basic topological notions in included set topology

This section discusses total set, closed sets, interior, closure and boundary for included set topology.

The topology is defined in the set X.

```
lemma union˙includedset:
  assumes T⊆X
  shows ⋃IncludedSet(X,T) = X
proof-
  from assms have X ∈ IncludedSet(X,T) using IncludedSet˙def by auto
  then show ⋃IncludedSet(X,T) = X using IncludedSet˙def by auto
qed
```

The closed sets are those which are disjoint with T and X.

```
lemma closed˙sets˙includedset:
  assumes T⊆X
  shows D –is closed in″ IncludedSet(X,T) ⟷ (D∈Pow(X) ∧  (D ∩ T)=0)∨ D=X
proof-
  have X-X=0 by blast
  then have X-X∈IncludedSet(X,T) using IncludedSet˙def by auto
  moreover
  {
    assume A:D ⊆ X X - D ∈ IncludedSet(X,T)  D ≠ X
    from A(2) have T⊆(X-D)∨ X-D=0 using IncludedSet˙def by auto
    with A(1) have T⊆(X-D)∨ D=X by blast
    with A(3) have T⊆(X-D) by auto
    hence D∩T=0 by blast
  }
  moreover
  {
    assume A:D∩T=0D⊆X
    from A(1) assms have T⊆(X-D) by blast
    then have X-D∈IncludedSet(X,T) using IncludedSet˙def by auto
```

```
  }
  ultimately show thesis using IsClosed˙def union˙includedset assms by auto
qed
```

The interior of a set is itself if it is open or the empty set if it isn't.

```
lemma interior˙set˙includedset:
  assumes A⊆X
  shows Interior(A,IncludedSet(X,T))= (if T⊆A then A else 0)
proof-
  {
    fix x
    assume A:Interior(A,IncludedSet(X,T)) ≠ 0 x∈T
    have Interior(A,IncludedSet(X,T)) ∈ IncludedSet(X,T) using
      topology0.Top˙2˙L2 topology0˙includedset by auto
    with A(1) have T ⊆ Interior(A,IncludedSet(X,T)) using IncludedSet˙def
      by auto
    with A(2) have x ∈ Interior(A,IncludedSet(X,T)) by auto
    then have x∈A using topology0.Top˙2˙L1 topology0˙includedset by auto}
    moreover
  {
    assume T⊆A
    with assms have A∈IncludedSet(X,T) using IncludedSet˙def by auto
    then have Interior(A,IncludedSet(X,T)) = A using topology0.Top˙2˙L3
      topology0˙includedset by auto
  }
  ultimately show thesis by auto
qed
```

The closure of a set is itself if it is closed or the whole space if it is not.

```
lemma closure˙set˙includedset:
  assumes A⊆X T⊆X
  shows Closure(A,IncludedSet(X,T)) = (if T∩A=0 then A else X)
proof-
  {
    assume AS:T∩A=0
    then have A –is closed in″ IncludedSet(X,T) using closed˙sets˙includedset
      assms by auto
    with assms(1) have Closure(A,IncludedSet(X,T))=A using topology0.Top˙3˙L8
      topology0˙includedset union˙includedset assms(2) by auto
  }
  moreover
  {
    assume AS:T∩A ≠ 0
    have X∈ClosedCovers(A,IncludedSet(X,T)) using ClosedCovers˙def
      closed˙sets˙includedset union˙includedset assms by auto
    then have l1:⋂ClosedCovers(A,IncludedSet(X,T))⊆X using Closure˙def
      by auto
    moreover
    {
```

```
    fix U
    assume U∈ClosedCovers(A,IncludedSet(X,T))
    then have U–is closed in"IncludedSet(X,T)A⊆U using ClosedCovers˙def
      by auto
    then have U=X∨(T∩U)=0A⊆U using closed˙sets˙includedset assms(2)
      by auto
    then have U=X∨(T∩A)=0 by auto
    then have U=X using AS by auto
  }
  then have X ⊆ ⋂ClosedCovers(A,IncludedSet(X,T)) using topology0.Top˙3˙L3
    topology0˙includedset union˙includedset assms by auto
  ultimately have ⋂ClosedCovers(A,IncludedSet(X,T))=X by auto
  then have Closure(A,IncludedSet(X,T)) = X
    using Closure˙def by auto
 }
 ultimately show thesis by auto
qed
```

The boundary of a set is X-A if A contains T completely, is A if $X - A$ contains T completely and X if T is divided between the two sets. The case where T=0 is considered as a special case.

```
lemma boundary˙includedset:
 assumes A⊆X T⊆X T≠0
 shows Boundary(A,IncludedSet(X,T))=(if T⊆A then X-A else (if T∩A=0 then A
else X))
proof -
 from ‹A⊆X› have X-A ⊆ X by auto
 {
  assume T⊆A
  with assms(2,3) have T∩A≠0 and T∩(X-A)=0 by auto
  with assms(1,2) ‹X-A ⊆ X› have
  Closure(A,IncludedSet(X,T)) = X and Closure(X-A,IncludedSet(X,T)) = (X-A)
    using closure˙set˙includedset by auto
  with assms(2) have Boundary(A,IncludedSet(X,T)) = X-A
    using Boundary˙def union˙includedset by auto
 }
 moreover
 {
  assume ~(T⊆A) and T∩A=0
  with assms(2) have T∩(X-A)≠0 by auto
  with assms(1,2) ‹T∩A=0› ‹X-A ⊆ X› have
    Closure(A,IncludedSet(X,T)) = A and Closure(X-A,IncludedSet(X,T)) = X
    using  closure˙set˙includedset by auto
   with assms(1,2) have Boundary(A,IncludedSet(X,T))=A using Boundary˙def
union˙includedset
    by auto
 }
 moreover
 {
```

```
    assume ~(T⊆A) and T∩A ≠ 0
    with assms(1,2) have T∩(X-A) ≠ 0 by auto
    with assms(1,2) ‹T∩A≠0› ‹X-A ⊆ X› have
      Closure(A,IncludedSet(X,T)) = X and Closure(X-A,IncludedSet(X,T)) = X
      using closure˙set˙includedset by auto
    with assms(2) have Boundary(A,IncludedSet(X,T)) = X
      using Boundary˙def union˙includedset  by auto
  }
  ultimately show thesis by auto
qed
```

57.8 Special cases and subspaces

In this section we discuss some corner cases when some parameters in our definitions are empty and provide some facts about subspaces in included set topologies.

The topology is discrete if T=0

```
lemma smaller˙includedset:
  shows IncludedSet(X,0) = Pow(X)
proof
  show IncludedSet(X,0) ⊆ Pow(X) and Pow(X) ⊆ IncludedSet(X,0)
    unfolding IncludedSet˙def by auto
qed
```

If the set which is included is not a subset of X, then the topology is trivial.

```
lemma empty˙includedset:
  assumes ~(T⊆X)
  shows IncludedSet(X,T) = –0″
proof
  from assms show IncludedSet(X,T) ⊆ –0″ and –0″ ⊆ IncludedSet(X,T)
    unfolding IncludedSet˙def by auto
qed
```

The topological subspaces of the IncludedSet(X,T) topology are also IncludedSet topologies. The trivial case does not fit the idea in the demonstration because if Y⊆X then IncludedSet(Y∩X, Y∩T) is never trivial. There is no need for a separate proof because the only subspace of the trivial topology is itself.

```
lemma subspace˙includedset:
  assumes T⊆X
  shows IncludedSet(X,T) –restricted to″ Y = IncludedSet(Y∩X,Y∩T)
proof
  {
    fix M
    assume M ∈ (IncludedSet(X,T) –restricted to″ Y)
     then obtain A where A1:A:IncludedSet(X,T) M = Y∩A unfolding Restrict-
edTo˙def
```

```
    by auto
  then have M ∈ Pow(X∩Y) unfolding IncludedSet˙def by auto
  moreover
  from A1 have Y∩T⊆M ∨ M=0 unfolding IncludedSet˙def by blast
  ultimately have M ∈ IncludedSet(Y∩X, Y∩T) unfolding IncludedSet˙def
    by auto
 }
 then show IncludedSet(X,T) –restricted to″ Y ⊆ IncludedSet(Y∩X, Y∩T)
  by auto
 {
  fix M
  let A = M ∪ T
  assume A:M ∈ IncludedSet(Y∩X, Y∩T)
  {
    assume M=0
    then have M∈IncludedSet(X,T) –restricted to″ Y unfolding RestrictedTo˙def
      IncludedSet˙def by auto
  }
  moreover
  {
    assume AS:M≠0
     from A AS have A1:M∈Pow(Y∩X) ∧ Y∩T⊆M unfolding IncludedSet˙def by
auto
    then have A∈Pow(X) using assms by blast
    moreover
    have T⊆A by blast
    ultimately have A ∈ IncludedSet(X,T) unfolding IncludedSet˙def by auto
    then have AT:Y ∩ A ∈ IncludedSet(X,T) –restricted to″ Yunfolding Restrict-
edTo˙def
      by auto
    from A1 have Y ∩ A=Y ∩ M by blast
    also from A1 have ...=M by auto
    finally have Y∩A = M by simp
    with AT have M ∈ IncludedSet(X,T) –restricted to″ Y
      by auto
  }
  ultimately have M ∈ IncludedSet(X,T) –restricted to″ Y by auto
 }
 thus IncludedSet(Y∩X, Y∩T) ⊆ IncludedSet(X,T) –restricted to″ Y by auto
qed

end
```

58 More examples in topology

```
theory Topology˙ZF˙examples˙1
imports Topology˙ZF˙1 Order˙ZF
begin
```

In this theory file we reformulate the concepts related to a topology in relation with a base of the topology and we give examples of topologies defined by bases or subbases.

58.1 New ideas using a base for a topology

58.2 The topology of a base

Given a family of subsets satisfying the base condition, it is possible to construct a topology where that family is a base of. Even more, it is the only topology with such characteristics.

```
definition
  TopologyWithBase (TopologyBase ˙ 50) where
  U –satisfies the base condition″ ⟹ TopologyBase U ≡ THE T. U –is a base for″
T
```

If a collection U of sets satisfies the base condition then the topology constructed from it is indeed a topology and U is a base for this topology.

```
theorem Base˙topology˙is˙a˙topology:
  assumes U –satisfies the base condition″
  shows (TopologyBase U) –is a topology″ and U –is a base for″ (TopologyBase U)
proof-
  from assms obtain T where U –is a base for″ T using
    Top˙1˙2˙T1(2) by blast
  then have ∃!T. U –is a base for″ T using same˙base˙same˙top ex1I[where P=
    λT. U –is a base for″ T] by blast
  with assms show U –is a base for″ (TopologyBase U)  using theI[where P=
    λT. U –is a base for″ T] TopologyWithBase˙def by auto
  with assms  show (TopologyBase U) –is a topology″ using Top˙1˙2˙T1(1)
    IsAbaseFor˙def by auto
qed
```

A base doesn't need the empty set.

```
lemma base˙no˙0:
  shows B–is a base for″T ⟷ (B-–0″)–is a base for″T
proof-
  {
    fix M
    assume M∈–⋃A . A ∈ Pow(B)″
    then obtain Q where M=⋃QQ∈Pow(B) by auto
    hence M=⋃(Q-–0″)Q-–0″∈Pow(B-–0″) by auto
    hence M∈–⋃A . A ∈ Pow(B - –0″)″ by auto
  }
  hence –⋃A . A ∈ Pow(B)″ ⊆ –⋃A . A ∈ Pow(B - –0″)″ by blast
  moreover
  {
    fix M
```

```
    assume M∈–⋃A . A ∈ Pow(B-–0″)″
    then obtain Q where M=⋃QQ∈Pow(B-–0″) by auto
    hence M=⋃(Q)Q∈Pow(B) by auto
    hence M∈–⋃A . A ∈ Pow(B)″ by auto
  }
  hence  –⋃A . A ∈ Pow(B - –0″)″ ⊆ –⋃A . A ∈ Pow(B)″
    by auto
  ultimately have –⋃A . A ∈ Pow(B - –0″)″ = –⋃A . A ∈ Pow(B)″  by auto
  then show B–is a base for″T ⟷ (B-–0″)–is a base for″T using IsAbaseFor˙def
by auto
qed
```

The interior of a set is the union of all the sets of the base which are fully contained by it.

```
lemma interior˙set˙base˙topology:
  assumes U –is a base for″ T T–is a topology″
  shows Interior(A,T) = ⋃–T∈U. T⊆A″
proof
  have –T∈U. T⊆A″⊆U by auto
  with assms(1) have ⋃–T∈U. T⊆A″∈T
    using IsAbaseFor˙def by auto
  moreover have ⋃–T∈U. T⊆A″ ⊆ A by blast
  ultimately show  ⋃–T∈U. T⊆A″ ⊆ Interior(A,T)
    using assms(2) topology0.Top˙2˙L5 topology0˙def by auto
  {
    fix x
    assume x ∈ Interior(A,T)
    with assms obtain V where V∈U V ⊆ Interior(A,T) x∈V
      using point˙open˙base˙neigh topology0.Top˙2˙L2 topology0˙def
      by blast
    with assms have V∈U x∈V V⊆A using topology0.Top˙2˙L1 topology0˙def
      by auto
    hence x ∈ ⋃–T∈U. T⊆A″ by auto
  }
  thus Interior(A, T) ⊆ ⋃–T ∈ U . T ⊆ A″  by auto
qed
```

In the following, we offer another lemma about the closure of a set given a basis for a topology. This lemma is based on cl˙inter˙neigh and inter˙neigh˙cl. It states that it is only necessary to check the sets of the base, not all the open sets.

```
lemma closure˙set˙base˙topology:
  assumes U –is a base for″ Q Q–is a topology″ A⊆⋃Q
  shows Closure(A,Q) = –x∈⋃Q. ∀T∈U. x∈T⟶A∩T≠0″
proof
  {
    fix x
    assume A:x∈Closure(A,Q)
    with assms(2,3) have B:x∈⋃Q using topology0˙def topology0.Top˙3˙L11(1)
```

```
      by blast
    moreover
    {
      fix T
      assume T∈U x∈T
      with assms(1) have T∈Qx∈T using base˙sets˙open by auto
      with assms(2,3) A have A∩T ≠ 0 using topology0˙def topology0.cl˙inter˙neigh
        by auto
    }
    hence ∀T∈U. x∈T⟶A∩T≠0 by auto
    ultimately have x∈–x∈⋃Q. ∀T∈U. x∈T⟶A∩T≠0″ by auto
  }
  thus Closure(A, Q) ⊆–x∈⋃Q. ∀T∈U. x∈T⟶A∩T≠0″
    by auto
  {
    fix x
    assume AS:x∈–x ∈ ⋃Q . ∀T∈U. x ∈ T ⟶ A ∩ T ≠ 0″
    hence x∈⋃Q by blast
    moreover
    {
      fix R
      assume R∈Q
      with assms(1) obtain W where RR:W⊆U R=⋃W using
        IsAbaseFor˙def by auto
      {
        assume x∈R
        with RR(2) obtain WW where TT:WW∈Wx∈WW by auto
        {
          assume R∩A=0
          with RR(2) TT(1) have WW∩A=0  by auto
           with TT(1) RR(1) have WW∈U WW∩A=0 by auto
          with AS have x∈⋃Q-WW by auto
          with TT(2) have False by auto
        }
        hence R∩A≠0 by auto
      }
    }
    hence ∀U∈Q. x∈U ⟶ U∩A≠0 by auto
   ultimately have x∈Closure(A,Q) using assms(2,3) topology0˙def topology0.inter˙neigh˙cl

      by auto
  }
  then show –x ∈ ⋃Q . ∀T∈U. x ∈ T ⟶ A ∩ T ≠ 0″ ⊆ Closure(A,Q)
    by auto
qed
```

The restriction of a base is a base for the restriction.

```
lemma subspace˙base˙topology:
  assumes B –is a base for″ T
```

```
  shows (B –restricted to″ Y) –is a base for″ (T –restricted to″ Y)
proof -
  from assms have (B –restricted to″ Y) ⊆ (T –restricted to″ Y)
    unfolding IsAbaseFor˙def RestrictedTo˙def by auto
  moreover have (T –restricted to″ Y) = –⋃A. A ∈ Pow(B –restricted to″ Y)″
  proof
    { fix U assume U ∈ (T –restricted to″ Y)
     then obtain W where W∈T and U = W∩Y unfolding RestrictedTo˙def by blast
      with assms obtain C where C∈Pow(B) and W=⋃C unfolding IsAbaseFor˙def
        by blast
      let A=–V∩Y. V∈C″
      from ‹C∈Pow(B)› ‹U = W∩Y› ‹W=⋃C› have
        A ∈ Pow(B –restricted to″ Y) and U=(⋃A)
        unfolding RestrictedTo˙def by auto
      hence U ∈ –⋃A. A ∈ Pow(B –restricted to″ Y)″ by blast
    } thus (T –restricted to″ Y) ⊆ –⋃A. A ∈ Pow(B –restricted to″ Y)″
      by auto
    { fix U assume U ∈ –⋃A. A ∈ Pow(B –restricted to″ Y)″
      then obtain A where A: A ⊆ (B –restricted to″ Y) and U = (⋃A) by auto
      let A0=–C∈B. Y∩C∈A″
      from A have A0 ⊆ B and A = A0 –restricted to″ Y unfolding RestrictedTo˙def
        by auto
      with ‹U = (⋃A)› have A0 ⊆ B and U = ⋃(A0 –restricted to″ Y)
        by auto
     with assms have U ∈ (T –restricted to″ Y) unfolding RestrictedTo˙def IsAbase-
For˙def
        by auto
    } thus –⋃A. A ∈ Pow(B –restricted to″ Y)″ ⊆ (T –restricted to″ Y) by blast
  qed
  ultimately show thesis unfolding IsAbaseFor˙def by simp
qed
```

If the base of a topology is contained in the base of another topology, then the topologies maintain the same relation.

```
theorem base˙subset:
  assumes B–is a base for″TB2–is a base for″T2B⊆B2
  shows T⊆T2
proof
  {
    fix x
    assume x∈T
    with assms(1) obtain M where M⊆Bx=⋃M using IsAbaseFor˙def by auto
    with assms(3) have M⊆B2x=⋃M by auto
    with assms(2) show x∈T2 using IsAbaseFor˙def by auto
  }
qed
```

58.3 Dual Base for Closed Sets

A dual base for closed sets is the collection of complements of sets of a base for the topology.

```
definition
  DualBase (DualBase ˙ ˙ 80) where
  B–is a base for″T ⟹ DualBase B T≡–⋃T-U. U∈B″∪–⋃T″


lemma closed˙inter˙dual˙base:
  assumes D–is closed in″TB–is a base for″T
  obtains M where M⊆DualBase B TD=⋂M
proof-
  assume K:⋀M. M ⊆ DualBase B T ⟹ D = ⋂M ⟹ thesis
  {
    assume AS:D≠⋃T
    from assms(1) have D:D∈Pow(⋃T)⋃T-D∈T using IsClosed˙def by auto
    hence A:⋃T-(⋃T-D)=D⋃T-D∈T by auto
    with assms(2) obtain Q where QQ:Q∈Pow(B)⋃T-D=⋃Q using IsAbaseFor˙def
by auto
    {
      assume Q=0
      then have ⋃Q=0 by auto
      with QQ(2) have ⋃T-D=0 by auto
      with D(1) have D=⋃T by auto
      with AS have False by auto
    }
    hence QNN:Q≠0 by auto
    from QQ(2) A(1) have D=⋃T-⋃Q by auto
    with QNN have D=⋂–⋃T-R. R∈Q″ by auto
    moreover
    with assms(2) QQ(1) have –⋃T-R. R∈Q″⊆DualBase B T using DualBase˙def
      by auto
    with calculation K have thesis by auto
  }
  moreover
  {
    assume AS:D=⋃T
    with assms(2) have –⋃T″⊆DualBase B T using DualBase˙def by auto
    moreover
    have ⋃T = ⋂–⋃T″ by auto
    with calculation K AS have thesis by auto
  }
  ultimately show thesis by auto
qed
```

We have already seen for a base that whenever there is a union of open sets, we can consider only basic open sets due to the fact that any open set is a union of basic open sets. What we should expect now is that when there is

an intersection of closed sets, we can consider only dual basic closed sets.

```
lemma closure˙dual˙base:
  assumes U –is a base for″ QQ–is a topology″A⊆⋃Q
  shows Closure(A,Q)=⋂–T∈DualBase U Q. A⊆T″
proof
  from assms(1) have T:⋃Q∈DualBase U Q using DualBase˙def by auto
  moreover
  {
    fix T
    assume A:T∈DualBase U Q A⊆T
    with assms(1) obtain R where (T=⋃Q-R∧R∈U)∨T=⋃Q using DualBase˙def
      by auto
     with A(2) assms(1,2) have  (T–is closed in″Q)A⊆TT∈Pow(⋃Q) using topol-
ogy0.Top˙3˙L1 topology0˙def
      topology0.Top˙3˙L9 base˙sets˙open by auto
  }
 then have SUB:–T∈DualBase U Q. A⊆T″⊆–T∈Pow(⋃Q). (T–is closed in″Q)∧A⊆T″
    by blast
 with calculation assms(3) have ⋂–T∈Pow(⋃Q). (T–is closed in″Q)∧A⊆T″⊆⋂–T∈DualBase
U Q. A⊆T″
    by auto
  then show Closure(A,Q)⊆⋂–T∈DualBase U Q. A⊆T″ using Closure˙def Closed-
Covers˙def
    by auto
  {
    fix x
    assume A:x∈⋂–T∈DualBase U Q. A⊆T″
    {
      fix T
      assume B:x∈TT∈U
      {
        assume C:A∩T=0
        from B(2) assms(1) have ⋃Q-T∈DualBase U Q using DualBase˙def
          by auto
        moreover
        from C assms(3) have A⊆⋃Q-T by auto
        moreover
        from B(1) have x∉⋃Q-T by auto
        ultimately have x∉⋂–T∈DualBase U Q. A⊆T″ by auto
        with A have False by auto
      }
      hence A∩T≠0 by auto
    }
    hence ∀T∈U. x∈T⟶A∩T≠0 by auto
    moreover
    from T A assms(3) have x∈⋃Q by auto
    with calculation assms have x∈Closure(A,Q) using closure˙set˙base˙topology
     by auto
  }
```

```
  thus ⋂–T ∈ DualBase U Q . A ⊆ T″ ⊆ Closure(A, Q) by auto
qed
```

58.4 Partition topology

In the theory file Partitions_ZF.thy; there is a definition to work with partitions. In this setting is much easier to work with a family of subsets.

```
definition
  IsAPartition (˙–is a partition of″˙ 90) where
  (U –is a partition of″ X) ≡ (⋃U=X ∧ (∀A∈U. ∀B∈U. A=B∨ A∩B=0)∧ 0∉U)
```

A subcollection of a partition is a partition of its union.

```
lemma subpartition:
  assumes U –is a partition of″ X V⊆U
  shows V–is a partition of″⋃V
  using assms unfolding IsAPartition˙def by auto
```

A restriction of a partition is a partition. If the empty set appears it has to be removed.

```
lemma restriction˙partition:
  assumes U –is a partition of″X
  shows ((U –restricted to″ Y)-–0″) –is a partition of″ (X∩Y)
  using assms unfolding IsAPartition˙def RestrictedTo˙def
  by fast
```

Given a partition, the complement of a union of a subfamily is a union of a subfamily.

```
lemma diff˙union˙is˙union˙diff:
  assumes R⊆P P –is a partition of″ X
  shows X - ⋃R=⋃(P-R)
proof
  {
    fix x
    assume x∈X - ⋃R
    hence P:x∈Xx∉⋃R by auto
    {
      fix T
      assume T∈R
      with P(2) have x∉T by auto
    }
   with P(1) assms(2) obtain Q where Q∈(P-R)x∈Q using IsAPartition˙def by auto
    hence x∈⋃(P-R) by auto
  }
  thus X - ⋃R⊆⋃(P-R) by auto
  {
    fix x
    assume x∈⋃(P-R)
    then obtain Q where Q∈P-Rx∈Q by auto
```

```
    hence C: Q∈PQ∉Rx∈Q by auto
    then have x∈⋃P by auto
    with assms(2) have x∈X using IsAPartition˙def by auto
    moreover
    {
      assume x∈⋃R
      then obtain t where G:t∈R x∈t by auto
      with C(3) assms(1) have t∩Q≠0t∈P by auto
      with assms(2) C(1,3) have t=Q using IsAPartition˙def
        by blast
      with C(2) G(1) have False by auto
    }
    hence x∉⋃R by auto
    ultimately have x∈X-⋃R by auto
  }
  thus ⋃(P-R)⊆X - ⋃R by auto
qed
```

58.5 Partition topology is a topology.

A partition satisfies the base condition.

```
lemma partition˙base˙condition:
  assumes P –is a partition of″ X
  shows P –satisfies the base condition″
proof-
  {
    fix U V
    assume AS:U∈P∧V∈P
    with assms have A:U=V∨ U∩V=0 using IsAPartition˙def by auto
    {
      fix x
      assume ASS:x∈U∩V
      with A have U=V by auto
      with AS ASS have U∈Px∈U∧ U⊆U∩V by auto
      hence ∃W∈P. x∈W∧ W⊆U∩V by auto
    }
    hence (∀x ∈ U∩V. ∃W∈P. x∈W ∧ W ⊆ U∩V) by auto
  }
  then show thesis using SatisfiesBaseCondition˙def by auto
qed
```

Since a partition is a base of a topology, and this topology is uniquely determined; we can built it. In the definition we have to make sure that we have a partition.

```
definition
  PartitionTopology (PTopology ˙ ˙ 50) where
  (U –is a partition of″ X) ⟹ PTopology X U ≡ TopologyBase U
```

```
theorem Ptopology˙is˙a˙topology:
  assumes U –is a partition of″ X
  shows (PTopology X U) –is a topology″ and U –is a base for″ (PTopology X U)
  using assms Base˙topology˙is˙a˙topology partition˙base˙condition
    PartitionTopology˙def by auto

lemma topology0˙ptopology:
  assumes U –is a partition of″ X
  shows topology0(PTopology X U)
  using Ptopology˙is˙a˙topology topology0˙def assms by auto
```

58.6 Total set, Closed sets, Interior, Closure and Boundary

The topology is defined in the set X

```
lemma union˙ptopology:
  assumes U –is a partition of″ X
  shows ⋃(PTopology X U)=X
  using assms Ptopology˙is˙a˙topology(2) Top˙1˙2˙L5
    IsAPartition˙def by auto
```

The closed sets are the open sets.

```
lemma closed˙sets˙ptopology:
  assumes T –is a partition of″ X
  showsD –is closed in″ (PTopology X T) ⟷ D∈(PTopology X T)
proof
  from assms
  have B:T–is a base for″(PTopology X T) using Ptopology˙is˙a˙topology(2) by auto
  {
    fix D
    assume D –is closed in″ (PTopology X T)
    with assms have A:D∈Pow(X)X-D∈(PTopology X T)
      using IsClosed˙def union˙ptopology by auto
     from A(2) B obtain R where Q:R⊆T X-D=⋃R using Top˙1˙2˙L1[where B=T
and U=X-D]
      by auto
    from A(1) have X-(X-D)=D by blast
    with Q(2) have D=X-⋃R by auto
    with Q(1) assms have D=⋃(T-R) using diff˙union˙is˙union˙diff
      by auto
    with B show D∈(PTopology X T) using IsAbaseFor˙def by auto
  }
  {
    fix D
    assume D∈(PTopology X T)
    with B obtain R where Q:R⊆TD=⋃R using IsAbaseFor˙def by auto
    hence X-D=X-⋃R by auto
    with Q(1) assms have X-D=⋃(T-R) using diff˙union˙is˙union˙diff
      by auto
    with B have X-D∈(PTopology X T) using IsAbaseFor˙def by auto
```

```
    moreover
    from Q have D⊆⋃T by auto
    with assms have D⊆X using IsAPartition˙def by auto
    with calculation assms show D–is closed in″ (PTopology X T)
      using IsClosed˙def union˙ptopology by auto
  }
qed
```

There is a formula for the interior given by an intersection of sets of the dual base. Is the intersection of all the closed sets of the dual basis such that they do not complement A to X. Since the interior of X must be inside X, we have to enter X as one of the sets to be intersected.

```
lemma interior˙set˙ptopology:
  assumes U –is a partition of″ XA⊆X
 shows Interior(A,(PTopology X U))=⋂–T∈DualBase U (PTopology X U). T=X∨T∪A≠X″
proof
  {
    fix x
    assume x∈Interior(A,(PTopology X U))
    with assms obtain R where A:x∈RR∈(PTopology X U)R⊆A
      using topology0.open˙open˙neigh topology0˙ptopology
      topology0.Top˙2˙L2 topology0.Top˙2˙L1
      by auto
    with assms obtain B where B:B⊆UR=⋃B using Ptopology˙is˙a˙topology(2)
      IsAbaseFor˙def by auto
   from A(1,3) assms have XX:x∈XX∈–T∈DualBase U (PTopology X U). T=X∨T∪A≠X″
      using union˙ptopology[of UX] DualBase˙def[ofU] Ptopology˙is˙a˙topology(2)[of
UX] by (safe,blast,auto)
    moreover
    from B(2) A(1) obtain S where C:S∈Bx∈S by auto
    {
      fix T
      assume AS:T∈DualBase U (PTopology X U)T ∪A≠X
      from AS(1) assms obtain w where (T=X-w∧w∈U)∨(T=X)
        using DualBase˙def union˙ptopology Ptopology˙is˙a˙topology(2)
        by auto
      with assms(2) AS(2) have D:T=X-ww∈U by auto
      from D(2) have w⊆⋃U by auto
       with assms(1) have w⊆⋃(PTopology X U) using Ptopology˙is˙a˙topology(2)
Top˙1˙2˙L5[of UPTopology X U]
        by auto
      with assms(1) have w⊆X using union˙ptopology by auto
      with D(1) have X-T=w by auto
      with D(2) have X-T∈U by auto
      {
        assume x∈X-T
        with C B(1) have S∈US∩(X-T)≠0 by auto
        with ‹X-T∈U› assms(1) have X-T=S using IsAPartition˙def by auto
        with ‹X-T=w›‹T=X-w› have X-S=T by auto
```

with AS(2) have X-S∪A≠X by auto
from A(3) B(2) C(1) have S⊆A by auto
hence X-A⊆X-S by auto
with assms(2) have X-S∪A=X by auto
with ‹X-S∪A≠X› have False by auto
}
then have x∈T using XX by auto
}
ultimately have x∈⋂–T∈DualBase U (PTopology X U). T=X∨T∪A≠X″
by auto
}
thus Interior(A,(PTopology X U))⊆⋂–T∈DualBase U (PTopology X U). T=X∨T∪A≠X″
by auto
{
fix x
assume p:x∈⋂–T∈DualBase U (PTopology X U). T=X∨T∪A≠X″
hence noE:⋂–T∈DualBase U (PTopology X U). T=X∨T∪A≠X″≠0 by auto
{
fix T
assume T∈DualBase U (PTopology X U)
with assms(1) obtain w where T=X∨(w∈U∧T=X-w) using DualBase˙def
Ptopology˙is˙a˙topology(2) union˙ptopology by auto
with assms(1) have T=X∨(w∈(PTopology X U)∧T=X-w) using base˙sets˙open
Ptopology˙is˙a˙topology(2) by blast
with assms(1) have T–is closed in″(PTopology X U) using topology0.Top˙3˙L1[where
T=PTopology X U]
topology0˙ptopology topology0.Top˙3˙L9[where T=PTopology X U] union˙ptopology
by auto
}
moreover
from assms(1) p have X∈–T∈DualBase U (PTopology X U). T=X∨T∪A≠X″and
X:x∈X using Ptopology˙is˙a˙topology(2)
DualBase˙def union˙ptopology by auto
with calculation assms(1) have (⋂–T∈DualBase U (PTopology X U). T=X∨T∪A≠X″)
–is closed in″(PTopology X U)
using topology0.Top˙3˙L4[where K=–T∈DualBase U (PTopology X U). T=X∨T∪A≠X″]
topology0˙ptopology[where U=U and X=X]
by auto
with assms(1) have ab:(⋂–T∈DualBase U (PTopology X U). T=X∨T∪A≠X″)∈(PTopology
X U)
using closed˙sets˙ptopology by auto
with assms(1) obtain B where B∈Pow(U)(⋂–T∈DualBase U (PTopology X U).
T=X∨T∪A≠X″)=⋃B
using Ptopology˙is˙a˙topology(2) IsAbaseFor˙def by auto
with p obtain R where x∈RR∈UR⊆(⋂–T∈DualBase U (PTopology X U).
T=X∨T∪A≠X″)
by auto
with assms(1) have R:x∈RR∈(PTopology X U)R⊆(⋂–T∈DualBase U (PTopology
X U). T=X∨T∪A≠X″)X-R∈DualBase U (PTopology X U)

```
    using base˙sets˙open Ptopology˙is˙a˙topology(2) DualBase˙def union˙ptopology
    by (safe,blast,simp,blast)
  {
    assume (X-R) ∪A≠X
     with R(4) have X-R∈–T∈DualBase U (PTopology X U). T=X∨T∪A≠X″ by
auto
    hence ⋂–T∈DualBase U (PTopology X U). T=X∨T∪A≠X″⊆X-R by auto
    with R(3) have R⊆X-R using subset˙trans[where A=R and C=X-R] by auto
    hence R=0 by blast
    with R(1) have False by auto
  }
  hence I:(X-R) ∪A=X by auto
  {
    fix y
    assume ASR:y∈R
    with R(2) have y∈⋃(PTopology X U) by auto
    with assms(1) have XX:y∈X using union˙ptopology by auto
    with I have y∈(X-R) ∪A by auto
    with XX have y∉R∨y∈A by auto
    with ASR have y∈A by auto
  }
  hence R⊆A by auto
  with R(1,2) have ∃R∈(PTopology X U). (x∈R∧R⊆A) by auto
  with assms(1) have x∈Interior(A,(PTopology X U)) using topology0.Top˙2˙L6
    topology0˙ptopology by auto
 }
 thus ⋂–T ∈ DualBase U PTopology X U . T = X ∨ T ∪ A ≠ X″ ⊆ Interior(A,
PTopology X U)
  by auto
qed
```

The closure of a set is the union of all the sets of the partition which intersect with A.

```
lemma closure˙set˙ptopology:
 assumes U –is a partition of″ XA⊆X
 shows Closure(A,(PTopology X U))=⋃–T∈U. T∩A≠0″
proof
 {
  fix x
  assume A:x∈Closure(A,(PTopology X U))
   with assms have x∈⋃(PTopology X U) using topology0.Top˙3˙L11(1)[where
T=PTopology X U
    and A=A] topology0˙ptopology union˙ptopology by auto
   with assms(1) have x∈⋃U using Top˙1˙2˙L5[where B=U and T=PTopology X
U] Ptopology˙is˙a˙topology(2) by auto
  then obtain W where B:x∈WW∈U by auto
  with A have x∈Closure(A,(PTopology X U))∩W by auto
  moreover
   from assms B(2) have W∈(PTopology X U)A⊆X using base˙sets˙open Ptopol-
```

```
ogy˙is˙a˙topology(2)
      by (safe,blast)
    with calculation assms(1) have A∩W≠0 using topology0˙ptopology[where U=U
and X=X]
      topology0.cl˙inter˙neigh union˙ptopology by auto
    with B have x∈⋃–T∈U. T∩A≠0″ by blast
  }
  thus Closure(A, PTopology X U) ⊆ ⋃–T ∈ U . T ∩ A ≠ 0″ by auto
  {
    fix x
    assume x∈⋃–T ∈ U . T ∩ A ≠ 0″
    then obtain T where A:x∈TT∈UT∩A≠0 by auto
    from assms have A⊆⋃(PTopology X U) using union˙ptopology by auto
    moreover
    from A(1,2) assms(1) have x∈⋃(PTopology X U) using Top˙1˙2˙L5[where B=U
and T=PTopology X U]
      Ptopology˙is˙a˙topology(2) by auto
    moreover
    {
      fix Q
      assume B:Q∈(PTopology X U)x∈Q
      with assms(1) obtain M where C:Q=⋃MM⊆U using
        Ptopology˙is˙a˙topology(2)
        IsAbaseFor˙def by auto
      from B(2) C(1) obtain R where D:R∈Mx∈R by auto
      with C(2) A(1,2) have R∩T≠0R∈UT∈U by auto
      with assms(1) have R=T using IsAPartition˙def by auto
      with C(1) D(1) have T⊆Q by auto
      with A(3) have Q∩A≠0 by auto
    }
    then have ∀Q∈(PTopology X U). x∈Q ⟶ Q∩A≠0 by auto
     with calculation assms(1) have x∈Closure(A,(PTopology X U)) using topol-
ogy0.inter˙neigh˙cl
      topology0˙ptopology by auto
  }
  then show ⋃–T ∈ U . T ∩ A ≠ 0″ ⊆ Closure(A, PTopology X U)  by auto
qed
```

The boundary of a set is given by the union of the sets of the partition which have non empty intersection with the set but that are not fully contained in it. Another equivalent statement would be: the union of the sets of the partition which have non empty intersection with the set and its complement.

```
lemma boundary˙set˙ptopology:
  assumes U –is a partition of″ XA⊆X
  shows Boundary(A,(PTopology X U))=⋃–T∈U. T∩A≠0 ∧ ˜(T⊆A)″
proof-
  from assms have Closure(A,(PTopology X U))=⋃–T ∈ U . T ∩ A ≠ 0″ using
    closure˙set˙ptopology by auto
  moreover
```

```
  from assms(1) have Interior(A,(PTopology X U))=⋃–T ∈ U . T ⊆ A″ using
   interior˙set˙base˙topology Ptopology˙is˙a˙topology[where U=U and X=X] by auto
  with calculation assms have A:Boundary(A,(PTopology X U))=⋃–T ∈ U . T ∩
A ≠ 0″ - ⋃–T ∈ U . T ⊆ A″
    using topology0.Top˙3˙L12 topology0˙ptopology union˙ptopology
    by auto
  from assms(1) have (–T ∈ U . T ∩ A ≠ 0″) –is a partition of″ ⋃(–T ∈ U . T ∩
A ≠ 0″)
    using subpartition by blast
  moreover
  {
    fix T
    assume T∈UT⊆A
    with assms(1) have T∩A=TT≠0 using IsAPartition˙def by auto
    with ‹T∈U› have T∩A≠0T∈U by auto
  }
  then have –T ∈ U . T ⊆ A″⊆–T ∈ U . T ∩ A ≠ 0″  by auto
  ultimately have ⋃–T ∈ U . T ∩ A ≠ 0″ - ⋃–T ∈ U . T ⊆ A″=⋃((–T ∈ U . T
∩ A ≠ 0″)-(–T ∈ U . T ⊆ A″))
  using diff˙union˙is˙union˙diff by auto
  also have ...=⋃(–T ∈ U . T ∩ A ≠ 0 ∧ ˜(T⊆A)″) by blast
  with calculation A show thesis by auto
qed
```

58.7 Special cases and subspaces

The discrete and the indiscrete topologies appear as special cases of this partition topologies.

```
lemma discrete˙partition:
  shows —x″.x∈X″ –is a partition of″X
  using IsAPartition˙def by auto

lemma indiscrete˙partition:
  assumes X≠0
  shows –X″ –is a partition of″ X
  using assms IsAPartition˙def by auto

theorem discrete˙ptopology:
  shows (PTopology X —x″.x∈X″)=Pow(X)
proof
  {
    fix t
    assume t∈(PTopology X —x″.x∈X″)
    hence t⊆⋃(PTopology X —x″.x∈X″) by auto
    then have t∈Pow(X) using union˙ptopology
      discrete˙partition by auto
  }
  thus (PTopology X —x″.x∈X″)⊆Pow(X) by auto
  {
```

```
    fix t
    assume A:t∈Pow(X)
    have ⋃(—x″. x∈t″)=t by auto
    moreover
    from A have —x″. x∈t″∈Pow(—x″.x∈X″) by auto
    hence ⋃(—x″. x∈t″)∈–⋃A . A ∈ Pow(—x″ . x ∈ X″)″ by auto
    ultimately
    have t∈(PTopology X —x″ . x ∈ X″) using Ptopology˙is˙a˙topology(2)
      discrete˙partition IsAbaseFor˙def by auto
  }
  thus Pow(X) ⊆ (PTopology X —x″ . x ∈ X″)  by auto
qed

theorem indiscrete˙ptopology:
  assumes X≠0
  shows (PTopology X –X″)=–0,X″
proof
  {
    fix T
    assume T∈(PTopology X –X″)
    with assms obtain M where M⊆–X″⋃M=T using Ptopology˙is˙a˙topology(2)
      indiscrete˙partition IsAbaseFor˙def by auto
    then have T=0∨T=X by auto
  }
  then show (PTopology X –X″)⊆–0,X″ by auto
 from assms have 0∈(PTopology X –X″) using Ptopology˙is˙a˙topology(1) empty˙open
    indiscrete˙partition by auto
  moreover
   from assms have ⋃(PTopology X –X″)∈(PTopology X –X″) using union˙open
Ptopology˙is˙a˙topology(1)
    indiscrete˙partition by auto
  with assms have X∈(PTopology X –X″) using union˙ptopology indiscrete˙partition
    by auto
  ultimately show –0,X″⊆(PTopology X –X″) by auto
qed
```

The topological subspaces of the (PTopology X U) are partition topologies.

```
lemma subspace˙ptopology:
  assumes U–is a partition of″X
  shows (PTopology X U) –restricted to″ Y=(PTopology (X∩Y) ((U –restricted to″
Y)-–0″))
proof-
 from assms have U–is a base for″(PTopology X U) using Ptopology˙is˙a˙topology(2)
    by auto
  then have (U–restricted to″ Y)–is a base for″(PTopology X U)–restricted to″ Y
    using subspace˙base˙topology by auto
   then have ((U–restricted to″ Y)-–0″)–is a base for″(PTopology X U)–restricted
to″ Y using base˙no˙0
    by auto
```

```
  moreover
  from assms have ((U–restricted to″ Y)-–0″) –is a partition of″ (X∩Y)
    using restriction˙partition by auto
 then have ((U–restricted to″ Y)-–0″)–is a base for″(PTopology (X∩Y) ((U –restricted
to″ Y)-–0″))
    using Ptopology˙is˙a˙topology(2) by auto
  ultimately show thesis using same˙base˙same˙top by auto
qed
```

58.8 Order topologies

58.9 Order topology is a topology

Given a totally ordered set, several topologies can be defined using the order relation. First we define an open interval, notice that the set defined as Interval is a closed interval; and open rays.

```
definition
  IntervalX where
  IntervalX(X,r,b,c)≡(Interval(r,b,c)∩X)-–b,c″
definition
  LeftRayX where
  LeftRayX(X,r,b)≡–c∈X. ⟨c,b⟩∈r″-–b″
definition
  RightRayX where
  RightRayX(X,r,b)≡–c∈X. ⟨b,c⟩∈r″-–b″
```

Intersections of intervals and rays.

```
lemma inter˙two˙intervals:
  assumes bu∈Xbv∈Xcu∈Xcv∈XIsLinOrder(X,r)
 shows IntervalX(X,r,bu,cu)∩IntervalX(X,r,bv,cv)=IntervalX(X,r,GreaterOf(r,bu,bv),SmallerOf(r,cu,cv))
proof
  have T:GreaterOf(r,bu,bv)∈XSmallerOf(r,cu,cv)∈X using assms
   GreaterOf˙def SmallerOf˙def by (cases ⟨bu,bv⟩∈r,simp,simp,cases ⟨cu,cv⟩∈r,simp,simp)
  {
    fix x
    assume ASS:x∈IntervalX(X,r,bu,cu)∩IntervalX(X,r,bv,cv)
    then have x∈IntervalX(X,r,bu,cu)x∈IntervalX(X,r,bv,cv) by auto
    then have BB:x∈Xx∈Interval(r,bu,cu)x≠bux≠cux∈Interval(r,bv,cv)x≠bvx≠cv
    using IntervalX˙def assms by auto
    then have x∈X by auto
    moreover
    have x≠GreaterOf(r,bu,bv)x≠SmallerOf(r,cu,cv)
    proof-
    show x≠GreaterOf(r,bu,bv) using GreaterOf˙def BB(6,3) by (cases ⟨bu,bv⟩∈r,simp+)
    show x≠SmallerOf(r,cu,cv) using SmallerOf˙def BB(7,4) by (cases ⟨cu,cv⟩∈r,simp+)
    qed
    moreover
    have ⟨bu,x⟩∈r⟨x,cu⟩∈r⟨bv,x⟩∈r⟨x,cv⟩∈r using BB(2,5) Order˙ZF˙2˙L1A by auto
```

```
  then have ⟨GreaterOf(r,bu,bv),x⟩∈r⟨x,SmallerOf(r,cu,cv)⟩∈r using GreaterOf˙def
SmallerOf˙def
     by (cases ⟨bu,bv⟩∈r,simp,simp,cases ⟨cu,cv⟩∈r,simp,simp)
  then have x∈Interval(r,GreaterOf(r,bu,bv),SmallerOf(r,cu,cv)) using Order˙ZF˙2˙L1
by auto
   ultimately
  have x∈IntervalX(X,r,GreaterOf(r,bu,bv),SmallerOf(r,cu,cv)) using IntervalX˙def
T by auto
 }
  then show IntervalX(X, r, bu, cu) ∩ IntervalX(X, r, bv, cv) ⊆ IntervalX(X, r,
GreaterOf(r, bu, bv), SmallerOf(r, cu, cv))
   by auto
 {
   fix x
   assume x∈IntervalX(X,r,GreaterOf(r,bu,bv),SmallerOf(r,cu,cv))
  then have BB:x∈Xx∈Interval(r,GreaterOf(r,bu,bv),SmallerOf(r,cu,cv))x≠GreaterOf(r,bu,bv)x≠SmallerOf(r,
   using IntervalX˙def T by auto
   then have x∈X by auto
   moreover
    from BB(2) have CC:⟨GreaterOf(r,bu,bv),x⟩∈r⟨x,SmallerOf(r,cu,cv)⟩∈r using
Order˙ZF˙2˙L1A by auto
   {
     {
       assume AS:⟨bu,bv⟩∈r
       then have GreaterOf(r,bu,bv)=bv using GreaterOf˙def by auto
       then have ⟨bv,x⟩∈r using CC(1) by auto
         with AS have ⟨bu,x⟩∈r ⟨bv,x⟩∈r using assms IsLinOrder˙def trans˙def by
(safe, blast)
     }
     moreover
     {
       assume AS:⟨bu,bv⟩∉r
       then have GreaterOf(r,bu,bv)=bu using GreaterOf˙def by auto
       then have ⟨bu,x⟩∈r using CC(1) by auto
      from AS have ⟨bv,bu⟩∈r using assms IsLinOrder˙def IsTotal˙def assms by auto
       with ‹⟨bu,x⟩∈r› have ⟨bu,x⟩∈r ⟨bv,x⟩∈r using assms IsLinOrder˙def trans˙def
by (safe, blast)
     }
     ultimately have R:⟨bu,x⟩∈r ⟨bv,x⟩∈r by auto
     moreover
     {
       assume AS:x=bu
       then have ⟨bv,bu⟩∈r using R(2) by auto
       then have GreaterOf(r,bu,bv)=bu using GreaterOf˙def assms IsLinOrder˙def
       antisym˙def by auto
       then have False using AS BB(3) by auto
     }
     moreover
     {
```

```
      assume AS:x=bv
      then have ⟨bu,bv⟩∈r using R(1) by auto
      then have GreaterOf(r,bu,bv)=bv using GreaterOf˙def by auto
      then have False using AS BB(3) by auto
    }
    ultimately have ⟨bu,x⟩∈r ⟨bv,x⟩∈rx≠bux≠bv by auto
  }
  moreover
  {
    {
      assume AS:⟨cu,cv⟩∈r
      then have SmallerOf(r,cu,cv)=cu using SmallerOf˙def by auto
      then have ⟨x,cu⟩∈r using CC(2) by auto
      with AS have ⟨x,cu⟩∈r ⟨x,cv⟩∈r using assms IsLinOrder˙def trans˙def by(safe
,blast)
    }
    moreover
    {
      assume AS:⟨cu,cv⟩∉r
      then have SmallerOf(r,cu,cv)=cv using SmallerOf˙def by auto
      then have ⟨x,cv⟩∈r using CC(2) by auto
      from AS have ⟨cv,cu⟩∈r using assms IsLinOrder˙def IsTotal˙def by auto
      with ‹⟨x,cv⟩∈r› have ⟨x,cv⟩∈r ⟨x,cu⟩∈r using assms IsLinOrder˙def trans˙def
by(safe ,blast)
    }
    ultimately have R:⟨x,cv⟩∈r ⟨x,cu⟩∈r by auto
    moreover
    {
      assume AS:x=cv
      then have ⟨cv,cu⟩∈r using R(2) by auto
      then have SmallerOf(r,cu,cv)=cv using SmallerOf˙def assms IsLinOrder˙def
      antisym˙def by auto
      then have False using AS BB(4) by auto
    }
    moreover
    {
      assume AS:x=cu
      then have ⟨cu,cv⟩∈r using R(1) by auto
      then have SmallerOf(r,cu,cv)=cu using SmallerOf˙def by auto
      then have False using AS BB(4) by auto
    }
    ultimately have ⟨x,cu⟩∈r ⟨x,cv⟩∈rx≠cux≠cv by auto
  }
  ultimately
  have x∈IntervalX(X,r,bu,cu) x∈IntervalX(X,r,bv,cv) using Order˙ZF˙2˙L1 Inter-
valX˙def
    assms by auto
  then have x∈IntervalX(X, r, bu, cu) ∩ IntervalX(X, r, bv, cv)  by auto
}
```

```
  then show IntervalX(X,r,GreaterOf(r,bu,bv),SmallerOf(r,cu,cv)) ⊆ IntervalX(X,
r, bu, cu) ∩ IntervalX(X, r, bv, cv)
    by auto
qed

lemma inter˙rray˙interval:
  assumes bv∈Xbu∈Xcv∈XIsLinOrder(X,r)
 shows RightRayX(X,r,bu)∩IntervalX(X,r,bv,cv)=IntervalX(X,r,GreaterOf(r,bu,bv),cv)
proof
  {
    fix x
    assume x∈RightRayX(X,r,bu)∩IntervalX(X,r,bv,cv)
    then have x∈RightRayX(X,r,bu)x∈IntervalX(X,r,bv,cv) by auto
   then have BB:x∈Xx≠bux≠bvx≠cv⟨bu,x⟩∈rx∈Interval(r,bv,cv) using RightRayX˙def
IntervalX˙def
      by auto
    then have ⟨bv,x⟩∈r⟨x,cv⟩∈r using Order˙ZF˙2˙L1A by auto
     with ‹⟨bu,x⟩∈r› have ⟨GreaterOf(r,bu,bv),x⟩∈r using GreaterOf˙def by (cases
⟨bu,bv⟩∈r,simp+)
     with ‹⟨x,cv⟩∈r› have x∈Interval(r,GreaterOf(r,bu,bv),cv) using Order˙ZF˙2˙L1
by auto
    then have x∈IntervalX(X,r,GreaterOf(r,bu,bv),cv) using BB(1-4) IntervalX˙def
GreaterOf˙def
      by (simp)
  }
   then show RightRayX(X, r, bu) ∩ IntervalX(X, r, bv, cv) ⊆ IntervalX(X, r,
GreaterOf(r, bu, bv), cv) by auto
  {
    fix x
    assume x∈IntervalX(X, r, GreaterOf(r, bu, bv), cv)
     then have x∈Xx∈Interval(r,GreaterOf(r, bu, bv), cv)x≠cvx≠GreaterOf(r, bu,
bv) using IntervalX˙def by auto
    then have R:⟨GreaterOf(r, bu, bv),x⟩∈r⟨x,cv⟩∈r using Order˙ZF˙2˙L1A by auto
    with ‹x≠cv› have ⟨x,cv⟩∈rx≠cv by auto
    moreover
    {
      {
        assume AS:⟨bu,bv⟩∈r
        then have GreaterOf(r,bu,bv)=bv using GreaterOf˙def by auto
        then have ⟨bv,x⟩∈r using R(1) by auto
      with AS have ⟨bu,x⟩∈r ⟨bv,x⟩∈r using assms unfolding IsLinOrder˙def trans˙def
by (safe,blast)
      }
      moreover
      {
        assume AS:⟨bu,bv⟩∉r
        then have GreaterOf(r,bu,bv)=bu using GreaterOf˙def by auto
        then have ⟨bu,x⟩∈r using R(1) by auto
          from AS have ⟨bv,bu⟩∈r using assms unfolding IsLinOrder˙def IsTotal˙def
```

using assms by auto
with ‹⟨bu,x⟩∈r› have ⟨bu,x⟩∈r ⟨bv,x⟩∈r using assms unfolding IsLinOrder˙def trans˙def by (safe,blast)
}
ultimately have T:⟨bu,x⟩∈r ⟨bv,x⟩∈r by auto
moreover
{
assume AS:x=bu
then have ⟨bv,bu⟩∈r using T(2) by auto
then have GreaterOf(r,bu,bv)=bu unfolding GreaterOf˙def using assms unfolding IsLinOrder˙def
antisym˙def by auto
with ‹x≠GreaterOf(r,bu,bv)› have False using AS by auto
}
moreover
{
assume AS:x=bv
then have ⟨bu,bv⟩∈r using T(1) by auto
then have GreaterOf(r,bu,bv)=bv unfolding GreaterOf˙def by auto
with ‹x≠GreaterOf(r,bu,bv)› have False using AS by auto
}
ultimately have ⟨bu,x⟩∈r ⟨bv,x⟩∈rx≠bux≠bv by auto
}
with calculation ‹x∈X› have x∈RightRayX(X, r, bu)x∈IntervalX(X, r, bv, cv) unfolding RightRayX˙def IntervalX˙def
using Order˙ZF˙2˙L1 by auto
then have x∈RightRayX(X, r, bu) ∩ IntervalX(X, r, bv, cv) by auto
}
then show IntervalX(X, r, GreaterOf(r, bu, bv), cv) ⊆ RightRayX(X, r, bu) ∩ IntervalX(X, r, bv, cv) by auto
qed

lemma inter˙lray˙interval:
assumes bv∈Xcu∈Xcv∈XIsLinOrder(X,r)
shows LeftRayX(X,r,cu)∩IntervalX(X,r,bv,cv)=IntervalX(X,r,bv,SmallerOf(r,cu,cv))
proof
{
fix x assume x∈LeftRayX(X,r,cu)∩IntervalX(X,r,bv,cv)
then have B:x≠cux∈X⟨x,cu⟩∈r⟨bv,x⟩∈r⟨x,cv⟩∈rx≠bvx≠cv unfolding LeftRayX˙def IntervalX˙def Interval˙def
by auto
from ‹⟨x,cu⟩∈r› ‹⟨x,cv⟩∈r› have C:⟨x,SmallerOf(r, cu, cv)⟩∈r using SmallerOf˙def by (cases ⟨cu,cv⟩∈r,simp+)
from B(7,1) have x≠SmallerOf(r,cu,cv) using SmallerOf˙def by (cases ⟨cu,cv⟩∈r,simp+)
then have x∈IntervalX(X,r,bv,SmallerOf(r,cu,cv)) using B C IntervalX˙def Order˙ZF˙2˙L1 by auto
}
then show LeftRayX(X, r, cu) ∩ IntervalX(X, r, bv, cv) ⊆ IntervalX(X, r, bv,

```
SmallerOf(r, cu, cv)) by auto
  {
    fix x assume x∈IntervalX(X,r,bv,SmallerOf(r,cu,cv))
      then have R:x∈X⟨bv,x⟩∈r⟨x,SmallerOf(r,cu,cv)⟩∈rx≠bvx≠SmallerOf(r,cu,cv)
using IntervalX˙def Interval˙def
      by auto
    then have ⟨bv,x⟩∈rx≠bv by auto
    moreover
    {
      {
        assume AS:⟨cu,cv⟩∈r
        then have SmallerOf(r,cu,cv)=cu using SmallerOf˙def by auto
        then have ⟨x,cu⟩∈r using R(3) by auto
      with AS have ⟨x,cu⟩∈r ⟨x,cv⟩∈r using assms unfolding IsLinOrder˙def trans˙def
by (safe, blast)
      }
      moreover
      {
        assume AS:⟨cu,cv⟩∉r
        then have SmallerOf(r,cu,cv)=cv using SmallerOf˙def by auto
        then have ⟨x,cv⟩∈r using R(3) by auto
       from AS have ⟨cv,cu⟩∈r using assms IsLinOrder˙def IsTotal˙def assms by auto
         with ‹⟨x,cv⟩∈r› have ⟨x,cv⟩∈r ⟨x,cu⟩∈r using assms IsLinOrder˙def trans˙def
by (safe, blast)
      }
      ultimately have T:⟨x,cv⟩∈r ⟨x,cu⟩∈r by auto
      moreover
      {
        assume AS:x=cu
        then have ⟨cu,cv⟩∈r using T(1) by auto
        then have SmallerOf(r,cu,cv)=cu using SmallerOf˙def assms IsLinOrder˙def
          antisym˙def by auto
        with ‹x≠SmallerOf(r,cu,cv)› have False using AS by auto
      }
      moreover
      {
        assume AS:x=cv
        then have ⟨cv,cu⟩∈r using T(2) by auto
        then have SmallerOf(r,cu,cv)=cv using SmallerOf˙def assms IsLinOrder˙def
        antisym˙def by auto
        with ‹x≠SmallerOf(r,cu,cv)› have False using AS by auto
      }
      ultimately have ⟨x,cu⟩∈r ⟨x,cv⟩∈rx≠cux≠cv by auto
    }
   with calculation ‹x∈X› have x∈LeftRayX(X,r,cu)x∈IntervalX(X, r, bv, cv) using
LeftRayX˙def IntervalX˙def Interval˙def
      by auto
    then have x∈LeftRayX(X, r, cu) ∩ IntervalX(X, r, bv, cv) by auto
  }
```

```
  then show IntervalX(X, r, bv, SmallerOf(r, cu, cv)) ⊆ LeftRayX(X, r, cu) ∩
IntervalX(X, r, bv, cv)  by auto
qed

lemma inter˙lray˙rray:
  assumes bu∈Xcv∈XIsLinOrder(X,r)
  shows LeftRayX(X,r,bu)∩RightRayX(X,r,cv)=IntervalX(X,r,cv,bu)
  unfolding LeftRayX˙def RightRayX˙def IntervalX˙def Interval˙def by auto

lemma inter˙lray˙lray:
  assumes bu∈Xcv∈XIsLinOrder(X,r)
  shows LeftRayX(X,r,bu)∩LeftRayX(X,r,cv)=LeftRayX(X,r,SmallerOf(r,bu,cv))
proof
  {
    fix x
    assume x∈LeftRayX(X,r,bu)∩LeftRayX(X,r,cv)
    then have B:x∈X⟨x,bu⟩∈r⟨x,cv⟩∈rx≠bux≠cv using LeftRayX˙def by auto
    then have C:⟨x,SmallerOf(r,bu,cv)⟩∈r using SmallerOf˙def by (cases ⟨bu,cv⟩∈r,
auto)
    from B have D:x≠SmallerOf(r,bu,cv) using SmallerOf˙def by (cases ⟨bu,cv⟩∈r,
auto)
    from B C D have x∈LeftRayX(X,r,SmallerOf(r,bu,cv)) using LeftRayX˙def by
auto
  }
 then show LeftRayX(X, r, bu) ∩ LeftRayX(X, r, cv) ⊆ LeftRayX(X, r, SmallerOf(r,
bu, cv)) by auto
  {
    fix x
    assume x∈LeftRayX(X, r, SmallerOf(r, bu, cv))
   then have R:x∈X⟨x,SmallerOf(r,bu,cv)⟩∈rx≠SmallerOf(r,bu,cv) using LeftRayX˙def
by auto
    {
      {
        assume AS:⟨bu,cv⟩∈r
        then have SmallerOf(r,bu,cv)=bu using SmallerOf˙def by auto
        then have ⟨x,bu⟩∈r using R(2) by auto
       with AS have ⟨x,bu⟩∈r ⟨x,cv⟩∈r using assms IsLinOrder˙def trans˙def by(safe,
blast)
      }
      moreover
      {
        assume AS:⟨bu,cv⟩∉r
        then have SmallerOf(r,bu,cv)=cv using SmallerOf˙def by auto
        then have ⟨x,cv⟩∈r using R(2) by auto
       from AS have ⟨cv,bu⟩∈r using assms IsLinOrder˙def IsTotal˙def assms by auto
        with ‹⟨x,cv⟩∈r› have ⟨x,cv⟩∈r ⟨x,bu⟩∈r using assms IsLinOrder˙def trans˙def
by(safe, blast)
      }
      ultimately have T:⟨x,cv⟩∈r ⟨x,bu⟩∈r by auto
```

```
      moreover
      {
        assume AS:x=bu
        then have ⟨bu,cv⟩∈r using T(1) by auto
        then have SmallerOf(r,bu,cv)=bu using SmallerOf˙def assms IsLinOrder˙def
          antisym˙def by auto
        with ‹x≠SmallerOf(r,bu,cv)› have False using AS by auto
      }
      moreover
      {
        assume AS:x=cv
        then have ⟨cv,bu⟩∈r using T(2) by auto
        then have SmallerOf(r,bu,cv)=cv using SmallerOf˙def assms IsLinOrder˙def
          antisym˙def by auto
        with ‹x≠SmallerOf(r,bu,cv)› have False using AS by auto
      }
      ultimately have ⟨x,bu⟩∈r ⟨x,cv⟩∈rx≠bux≠cv by auto
    }
   with ‹x∈X› have x∈ LeftRayX(X, r, bu) ∩ LeftRayX(X, r, cv) using LeftRayX˙def
by auto
  }
  then show LeftRayX(X, r, SmallerOf(r, bu, cv)) ⊆ LeftRayX(X, r, bu) ∩ Left-
RayX(X, r, cv)  by auto
qed

lemma inter˙rray˙rray:
  assumes bu∈Xcv∈XIsLinOrder(X,r)
 shows RightRayX(X,r,bu)∩RightRayX(X,r,cv)=RightRayX(X,r,GreaterOf(r,bu,cv))
proof
  {
    fix x
    assume x∈RightRayX(X,r,bu)∩RightRayX(X,r,cv)
    then have B:x∈X⟨bu,x⟩∈r⟨cv,x⟩∈rx≠bux≠cv using RightRayX˙def by auto
   then have C:⟨GreaterOf(r,bu,cv),x⟩∈r using GreaterOf˙def by (cases ⟨bu,cv⟩∈r,auto)
   from B have D:x≠GreaterOf(r,bu,cv) using GreaterOf˙def by (cases ⟨bu,cv⟩∈r,auto)
    from B C D have x∈RightRayX(X,r,GreaterOf(r,bu,cv)) using RightRayX˙def
by auto
  }
   then show  RightRayX(X, r, bu) ∩ RightRayX(X, r, cv) ⊆ RightRayX(X, r,
GreaterOf(r, bu, cv)) by auto
  {
    fix x
    assume x∈RightRayX(X, r, GreaterOf(r, bu, cv))
      then have R:x∈X⟨GreaterOf(r,bu,cv),x⟩∈rx≠GreaterOf(r,bu,cv) using Righ-
tRayX˙def by auto
    {
      {
        assume AS:⟨bu,cv⟩∈r
        then have GreaterOf(r,bu,cv)=cv using GreaterOf˙def by auto
```

then have ⟨cv,x⟩∈r using R(2) by auto
with AS have ⟨bu,x⟩∈r ⟨cv,x⟩∈r using assms IsLinOrder˙def trans˙def by(safe, blast)
}
moreover
{
assume AS:⟨bu,cv⟩∉r
then have GreaterOf(r,bu,cv)=bu using GreaterOf˙def by auto
then have ⟨bu,x⟩∈r using R(2) by auto
from AS have ⟨cv,bu⟩∈r using assms IsLinOrder˙def IsTotal˙def assms by auto
with ‹⟨bu,x⟩∈r› have ⟨cv,x⟩∈r ⟨bu,x⟩∈r using assms IsLinOrder˙def trans˙def by(safe, blast)
}
ultimately have T:⟨cv,x⟩∈r ⟨bu,x⟩∈r by auto
moreover
{
assume AS:x=bu
then have ⟨cv,bu⟩∈r using T(1) by auto
then have GreaterOf(r,bu,cv)=bu using GreaterOf˙def assms IsLinOrder˙def antisym˙def by auto
with ‹x≠GreaterOf(r,bu,cv)› have False using AS by auto
}
moreover
{
assume AS:x=cv
then have ⟨bu,cv⟩∈r using T(2) by auto
then have GreaterOf(r,bu,cv)=cv using GreaterOf˙def assms IsLinOrder˙def antisym˙def by auto
with ‹x≠GreaterOf(r,bu,cv)› have False using AS by auto
}
ultimately have ⟨bu,x⟩∈r ⟨cv,x⟩∈rx≠bux≠cv by auto
}
with ‹x∈X› have x∈ RightRayX(X, r, bu) ∩ RightRayX(X, r, cv) using RightRayX˙def by auto
}
then show RightRayX(X, r, GreaterOf(r, bu, cv)) ⊆ RightRayX(X, r, bu) ∩ RightRayX(X, r, cv) by auto
qed

The open intervals and rays satisfy the base condition.

lemma intervals˙rays˙base˙condition:
assumes IsLinOrder(X,r)
shows –IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″∪–LeftRayX(X,r,b). b∈X″∪–RightRayX(X,r,b). b∈X″ –satisfies the base condition″
proof-
let I=–IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″
let R=–RightRayX(X,r,b). b∈X″
let L=–LeftRayX(X,r,b). b∈X″
let B=–IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″∪–LeftRayX(X,r,b). b∈X″∪–RightRayX(X,r,b).

```
b∈X″
  {
    fix U V
    assume A:U∈BV∈B
    then have dU:U∈I∨U∈L∨U∈Rand dV:V∈I∨V∈L∨V∈R by auto
    {
      assume S:V∈I
      {
        assume U∈I
     with S obtain bu cu bv cv where A:U=IntervalX(X,r,bu,cu)V=IntervalX(X,r,bv,cv)bu∈Xcu∈Xbv∈Xcv∈X
          by auto
     then have SmallerOf(r,cu,cv)∈XGreaterOf(r,bu,bv)∈X by (cases ⟨cu,cv⟩∈r,simp
add:SmallerOf˙def A,simp add:SmallerOf˙def A,
          cases ⟨bu,bv⟩∈r,simp add:GreaterOf˙def A,simp add:GreaterOf˙def A)
        with A have U∩V∈B using inter˙two˙intervals assms by auto
      }
      moreover
      {
        assume U∈L
     with S obtain bu bv cv where A:U=LeftRayX(X, r,bu)V=IntervalX(X,r,bv,cv)bu∈Xbv∈Xcv∈X
        by auto
      then have SmallerOf(r,bu,cv)∈X using SmallerOf˙def by (cases ⟨bu,cv⟩∈r,auto)
        with A have U∩V∈B using inter˙lray˙interval assms by auto
      }
      moreover
      {
        assume U∈R
     with S obtain cu bv cv where A:U=RightRayX(X,r,cu)V=IntervalX(X,r,bv,cv)cu∈Xbv∈Xcv∈X
        by auto
      then have GreaterOf(r,cu,bv)∈X using GreaterOf˙def by (cases ⟨cu,bv⟩∈r,auto)
        with A have U∩V∈B using inter˙rray˙interval assms by auto
      }
      ultimately have U∩V∈B using dU by auto
    }
    moreover
    {
      assume S:V∈L
      {
        assume U∈I
     with S obtain bu bv cv where A:V=LeftRayX(X, r,bu)U=IntervalX(X,r,bv,cv)bu∈Xbv∈Xcv∈X
          by auto
          then have SmallerOf(r,bu,cv)∈X using SmallerOf˙def by (cases ⟨bu,cv⟩∈r,
auto)
        have U∩V=V∩U by auto
        with A ‹SmallerOf(r,bu,cv)∈X› have U∩V∈B using inter˙lray˙interval assms
by auto
      }
      moreover
      {
```

```
      assume U∈R
    with S obtain bu cv where A:V=LeftRayX(X,r,bu)U=RightRayX(X,r,cv)bu∈Xcv∈X
      by auto
      have U∩V=V∩U by auto
      with A have U∩V∈B using inter˙lray˙rray assms by auto
    }
    moreover
    {
      assume U∈L
    with S obtain bu bv where A:U=LeftRayX(X,r,bu)V=LeftRayX(X,r,bv)bu∈Xbv∈X
      by auto
       then have SmallerOf(r,bu,bv)∈X using SmallerOf˙def by (cases ⟨bu,bv⟩∈r,
auto)
      with A have U∩V∈B using inter˙lray˙lray assms by auto
    }
    ultimately have U∩V∈B using dU by auto
  }
  moreover
  {
    assume S:V∈R
    {
      assume U∈I
    with S obtain cu bv cv where A:V=RightRayX(X,r,cu)U=IntervalX(X,r,bv,cv)cu∈Xbv∈Xcv∈X
      by auto
    then have GreaterOf(r,cu,bv)∈X using GreaterOf˙def by (cases ⟨cu,bv⟩∈r,auto)
      have U∩V=V∩U by auto
      with A ‹GreaterOf(r,cu,bv)∈X› have U∩V∈B using inter˙rray˙interval assms
by auto
    }
    moreover
    {
      assume U∈L
    with S obtain bu cv where A:U=LeftRayX(X,r,bu)V=RightRayX(X,r,cv)bu∈Xcv∈X
      by auto
      then have U∩V∈B using inter˙lray˙rray assms by auto
    }
    moreover
    {
      assume U∈R
    with S obtain cu cv where A:U=RightRayX(X,r,cu)V=RightRayX(X,r,cv)cu∈Xcv∈X
      by auto
    then have GreaterOf(r,cu,cv)∈X using GreaterOf˙def by (cases ⟨cu,cv⟩∈r,auto)
      with A have U∩V∈B using inter˙rray˙rray assms by auto
    }
    ultimately have U∩V∈B using dU by auto
  }
  ultimately have  S:U∩V∈B using dV by auto
  {
    fix x
```

```
    assume x∈U∩V
    then have x∈U∩V∧U∩V⊆U∩V by auto
    then have ∃W. W∈(B)∧ x∈W ∧ W ⊆ U∩V using S by blast
    then have ∃W∈(B). x∈W ∧ W ⊆ U∩V by blast
  }
  hence (∀x ∈ U∩V. ∃W∈(B). x∈W ∧ W ⊆ U∩V) by auto
 }
 then show thesis using SatisfiesBaseCondition˙def by auto
qed
```

Since the intervals and rays form a base of a topology, and this topology is uniquely determined; we can built it. In the definition we have to make sure that we have a totally ordered set.

```
definition
 OrderTopology (OrdTopology ˙ ˙ 50) where
 IsLinOrder(X,r) ⟹ OrdTopology X r ≡ TopologyBase –IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″∪–LeftRayX(X,r,b).
b∈X″∪–RightRayX(X,r,b). b∈X″

theorem Ordtopology˙is˙a˙topology:
 assumes IsLinOrder(X,r)
 shows (OrdTopology X r) –is a topology″ and –IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″∪–LeftRayX(X,r,b).
b∈X″∪–RightRayX(X,r,b). b∈X″ –is a base for″ (OrdTopology X r)
 using assms Base˙topology˙is˙a˙topology intervals˙rays˙base˙condition
  OrderTopology˙def by auto

lemma topology0˙ordtopology:
 assumes IsLinOrder(X,r)
 shows topology0(OrdTopology X r)
 using Ordtopology˙is˙a˙topology topology0˙def assms by auto
```

58.10 Total set

The topology is defined in the set X, when X has more than one point

```
lemma union˙ordtopology:
 assumes IsLinOrder(X,r)∃x y. x≠y ∧ x∈X∧ y∈X
 shows ⋃(OrdTopology X r)=X
proof
 let B=–IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″∪–LeftRayX(X,r,b). b∈X″∪–RightRayX(X,r,b).
b∈X″
 have base:B –is a base for″ (OrdTopology X r) using Ordtopology˙is˙a˙topology(2)
assms(1)
  by auto
 from assms(2) obtain x y where T:x≠y ∧ x∈X∧ y∈X by auto
 then have B:x∈LeftRayX(X,r,y)∨x∈RightRayX(X,r,y) using LeftRayX˙def Righ-
tRayX˙def
  assms(1) IsLinOrder˙def IsTotal˙def by auto
 then have x∈⋃B using T by auto
 then have x:x∈⋃(OrdTopology X r) using Top˙1˙2˙L5 base by auto
```

```
  {
    fix z
    assume z:z∈X
    {
      assume x=z
      then have z∈⋃(OrdTopology X r) using x by auto
    }
    moreover
    {
      assume x≠z
    with z T have z∈LeftRayX(X,r,x)∨z∈RightRayX(X,r,x)x∈X using LeftRayX˙def
RightRayX˙def
        assms(1) IsLinOrder˙def IsTotal˙def by auto
      then have z∈⋃B by auto
      then have z∈⋃(OrdTopology X r) using Top˙1˙2˙L5 base by auto
    }
    ultimately have z∈⋃(OrdTopology X r) by auto
  }
  then show X⊆⋃(OrdTopology X r) by auto
  have ⋃B⊆X using IntervalX˙def LeftRayX˙def RightRayX˙def by auto
  then show ⋃(OrdTopology X r)⊆X using Top˙1˙2˙L5 base by auto
qed
```

The interior, closure and boundary can be calculated using the formulas proved in the section that deals with the base.

The subspace of an order topology doesn't have to be an order topology.

58.11 Right order and Left order topologies.

Notice that the left and right rays are closed under intersection, hence they form a base of a topology. They are called right order topology and left order topology respectively.

If the order in X has a minimal or a maximal element, is necessary to consider X as an element of the base or that limit point wouldn't be in any basic open set.

58.11.1 Right and Left Order topologies are topologies

```
lemma leftrays˙base˙condition:
assumes IsLinOrder(X,r)
shows –LeftRayX(X,r,b). b∈X″∪–X″ –satisfies the base condition″
proof-
  {
    fix U V
    assume U∈–LeftRayX(X,r,b). b∈X″∪–X″V∈–LeftRayX(X,r,b). b∈X″∪–X″
   then obtain b c where A:(b∈X∧U=LeftRayX(X,r,b))∨U=X(c∈X∧V=LeftRayX(X,r,c))∨V=XU⊆XV⊆X
```

```
    unfolding LeftRayX˙def by auto
   then have (U∩V=LeftRayX(X,r,SmallerOf(r,b,c))∧b∈X∧c∈X)∨U∩V=X∨(U∩V=LeftRayX(X,r,c)∧c∈X)∨(U
      using inter˙lray˙lray assms by auto
    moreover
      have b∈X∧c∈X ⟶ SmallerOf(r,b,c)∈X unfolding SmallerOf˙def by (cases
⟨b,c⟩∈r,auto)
    ultimately have U∩V∈–LeftRayX(X,r,b). b∈X″∪–X″ by auto
    hence ∀x∈U∩V. ∃W∈–LeftRayX(X,r,b). b∈X″∪–X″. x∈W∧W⊆U∩V by blast
  }
  moreover
  then show thesis using SatisfiesBaseCondition˙def by auto
qed

lemma rightrays˙base˙condition:
assumes IsLinOrder(X,r)
shows –RightRayX(X,r,b). b∈X″∪–X″ –satisfies the base condition″
proof-
  {
    fix U V
    assume U∈–RightRayX(X,r,b). b∈X″∪–X″V∈–RightRayX(X,r,b). b∈X″∪–X″
   then obtain b c where A:(b∈X∧U=RightRayX(X,r,b))∨U=X(c∈X∧V=RightRayX(X,r,c))∨V=XU⊆XV⊆X
    unfolding RightRayX˙def by auto
   then have (U∩V=RightRayX(X,r,GreaterOf(r,b,c))∧b∈X∧c∈X)∨U∩V=X∨(U∩V=RightRayX(X,r,c)∧c∈X)

      using inter˙rray˙rray assms by auto
    moreover
   have b∈X∧c∈X ⟶ GreaterOf(r,b,c)∈X using GreaterOf˙def by (cases ⟨b,c⟩∈r,auto)
    ultimately have U∩V∈–RightRayX(X,r,b). b∈X″∪–X″ by auto
   hence ∀x∈U∩V. ∃W∈–RightRayX(X,r,b). b∈X″∪–X″. x∈W∧W⊆U∩V by blast
  }
  then show thesis using SatisfiesBaseCondition˙def by auto
qed

definition
  LeftOrderTopology (LOrdTopology ˙ ˙ 50) where
 IsLinOrder(X,r) ⟹ LOrdTopology X r ≡ TopologyBase –LeftRayX(X,r,b). b∈X″∪–X″

definition
  RightOrderTopology (ROrdTopology ˙ ˙ 50) where
   IsLinOrder(X,r) ⟹ ROrdTopology X r ≡ TopologyBase –RightRayX(X,r,b).
b∈X″∪–X″

theorem LOrdtopology˙ROrdtopology˙are˙topologies:
  assumes IsLinOrder(X,r)
  shows (LOrdTopology X r) –is a topology″ and –LeftRayX(X,r,b). b∈X″∪–X″ –is
a base for″ (LOrdTopology X r)
  and (ROrdTopology X r) –is a topology″ and –RightRayX(X,r,b). b∈X″∪–X″ –is
a base for″ (ROrdTopology X r)
 using Base˙topology˙is˙a˙topology leftrays˙base˙condition assms rightrays˙base˙condition
```

LeftOrderTopology˙def RightOrderTopology˙def by auto

lemma topology0˙lordtopology˙rordtopology:
assumes IsLinOrder(X,r)
shows topology0(LOrdTopology X r) and topology0(ROrdTopology X r)
using LOrdtopology˙ROrdtopology˙are˙topologies topology0˙def assms by auto

58.11.2 Total set

The topology is defined on the set X

lemma union˙lordtopology˙rordtopology:
assumes IsLinOrder(X,r)
shows ⋃(LOrdTopology X r)=X and ⋃(ROrdTopology X r)=X
using Top˙1˙2˙L5[OF LOrdtopology˙ROrdtopology˙are˙topologies(2)[OF assms]]
Top˙1˙2˙L5[OF LOrdtopology˙ROrdtopology˙are˙topologies(4)[OF assms]]
unfolding LeftRayX˙def RightRayX˙def by auto

58.12 Union of Topologies

The union of two topologies is not a topology. A way to overcome this fact is to define the following topology:

definition
joinT (joinT ˙ 90) where
(∀T∈M. T–is a topology″ ∧ (∀Q∈M. ⋃Q=⋃T)) ⟹ (joinT M ≡ THE T. (⋃M)–is a subbase for″ T)

First let's proof that given a family of sets, then it is a subbase for a topology.

The first result states that from any family of sets we get a base using finite intersections of them. The second one states that any family of sets is a subbase of some topology.

theorem subset˙as˙subbase:
shows –⋂A. A ∈ FinPow(B)″ –satisfies the base condition″
proof-
{
fix U V
assume A:U∈–⋂A. A ∈ FinPow(B)″ ∧ V∈–⋂A. A ∈ FinPow(B)″
then obtain M R where MR:Finite(M)Finite(R)M⊆BR⊆B
U=⋂MV=⋂R
using FinPow˙def by auto
{
fix x
assume AS:x∈U∩V
then have N:M≠0R≠0 using MR(5,6) by auto
have Finite(M ∪R) using MR(1,2) by auto
moreover
have M ∪ R∈Pow(B) using MR(3,4) by auto
ultimately have M∪R∈FinPow(B) using FinPow˙def by auto

```
      then have ⋂(M∪R)∈–⋂A. A ∈ FinPow(B)″ by auto
      moreover
      from N have ⋂(M∪R)⊆⋂M⋂(M∪R)⊆⋂R by auto
      then have ⋂(M∪R)⊆U∩V using MR(5,6) by auto
      moreover
      {
        fix S
        assume S∈M ∪ R
        then have S∈M∨S∈R by auto
        then have x∈S using AS MR(5,6) by auto
      }
      then have x∈⋂(M ∪ R) using N by auto
      ultimately have ∃W∈–⋂A. A ∈ FinPow(B)″. x∈W∧W⊆U∩V by blast
    }
    then have (∀x ∈ U∩V. ∃W∈–⋂A. A ∈ FinPow(B)″. x∈W ∧ W ⊆ U∩V) by
auto
  }
  then have ∀U V. ((U∈–⋂A. A ∈ FinPow(B)″ ∧ V∈–⋂A. A ∈ FinPow(B)″) ⟶
    (∀x ∈ U∩V. ∃W∈–⋂A. A ∈ FinPow(B)″. x∈W ∧ W ⊆ U∩V)) by auto
  then show –⋂A. A ∈ FinPow(B)″ –satisfies the base condition″
    using SatisfiesBaseCondition˙def by auto
qed

theorem Top˙subbase:
  assumes T = –⋃A. A∈Pow(–⋂A. A ∈ FinPow(B)″)″
  shows T –is a topology″ and B –is a subbase for″ T
proof-
  {
    fix S
    assume S∈B
    then have –S″∈FinPow(B)⋂–S″=S using FinPow˙def by auto
    then have –S″∈Pow(–⋂A. A ∈ FinPow(B)″) by (blast+)
    then have ⋃–S″∈–⋃A. A∈Pow(–⋂A. A ∈ FinPow(B)″)″ by blast
    then have S∈–⋃A. A∈Pow(–⋂A. A ∈ FinPow(B)″)″ by auto
    then have S∈T using assms by auto
  }
  then have B⊆T by auto
  moreover
  have –⋂A. A ∈ FinPow(B)″ –satisfies the base condition″
    using subset˙as˙subbase by auto
  then have T –is a topology″ and –⋂A. A ∈ FinPow(B)″ –is a base for″ T
    using Top˙1˙2˙T1 assms by auto
  ultimately show T –is a topology″ and B–is a subbase for″T
    using IsAsubBaseFor˙def by auto
qed
```

A subbase defines a unique topology.

```
theorem same˙subbase˙same˙top:
  assumes B –is a subbase for″ T and B –is a subbase for″ S
```

shows T = S
using IsAsubBaseFor˙def assms same˙base˙same˙top
by auto

end

59 Properties in Topology

theory Topology˙ZF˙properties imports Topology˙ZF˙examples Topology˙ZF˙examples˙1

begin

This theory deals with topological properties which make use of cardinals.

59.1 Properties of compactness

It is already defined what is a compact topological space, but the is a generalization which may be useful sometimes.

definition
IsCompactOfCard (˙–is compact of cardinal"˙ –in"˙ 90)
where K–is compact of cardinal" Q–in"T ≡ (Card(Q) ∧ K ⊆ ⋃T ∧
(∀ M∈Pow(T). K ⊆ ⋃M ⟶ (∃ N ∈ Pow(M). K ⊆ ⋃N ∧ N≺Q)))

The usual compact property is the one defined over the cardinal of the natural numbers.

lemma Compact˙is˙card˙nat:
shows K–is compact in"T ⟷ (K–is compact of cardinal" nat –in"T)
proof
{
assume K–is compact in"T
then have sub:K ⊆ ⋃T and reg:(∀ M∈Pow(T). K ⊆ ⋃M ⟶ (∃ N ∈ FinPow(M).
K ⊆ ⋃N))
using IsCompact˙def by auto
{
fix M
assume M∈Pow(T)K⊆⋃M
with reg obtain N where N∈FinPow(M) K⊆⋃N by blast
then have Finite(N) using FinPow˙def by auto
then obtain n where A:n∈natN ≈n using Finite˙def by auto
from A(1) have n≺nat using n˙lesspoll˙nat by auto
with A(2) have N≲nat using lesspoll˙def eq˙lepoll˙trans by auto
moreover
{
assume N ≈nat
then have nat ≈ N using eqpoll˙sym by auto
with A(2) have nat ≈n using eqpoll˙trans by blast
then have n ≈nat using eqpoll˙sym by auto

with ‹n≺nat› have False using lesspoll˙def by auto
}
then have ˜(N ≈nat) by auto
with calculation ‹K⊆⋃N›‹N∈FinPow(M)› have N≺natK⊆⋃NN∈Pow(M) using lesspoll˙def
FinPow˙def by auto
hence (∃ N ∈ Pow(M). K ⊆ ⋃N ∧ N≺nat) by auto
}
with sub show K–is compact of cardinal″ nat –in″T using IsCompactOfCard˙def Card˙nat by auto
}
{
assume (K–is compact of cardinal″ nat –in″T)
then have sub:K⊆⋃T and reg:(∀ M∈Pow(T). K ⊆ ⋃M ⟶ (∃ N ∈ Pow(M). K ⊆ ⋃N ∧ N≺nat))
using IsCompactOfCard˙def by auto
{
fix M
assume M∈Pow(T)K⊆⋃M
with reg have (∃ N ∈ Pow(M). K ⊆ ⋃N ∧ N≺nat) by auto
then obtain N where N∈Pow(M)K⊆⋃NN≺nat by blast
then have N∈FinPow(M)K⊆⋃N using lesspoll˙nat˙is˙Finite FinPow˙def by auto
hence ∃N∈FinPow(M). K⊆⋃N by auto
}
with sub show K–is compact in″T using IsCompact˙def by auto
}
qed

Another property of this kind widely used is the Lindeloef property; it is the one on the successor of the natural numbers.

definition
IsLindeloef (˙–is lindeloef in″˙ 90) where
K –is lindeloef in″ T≡K–is compact of cardinal″csucc(nat)–in″T

It would be natural to think that every countable set with any topology is Lindeloef; but this statement is not provable in ZF. The reason is that to build a subcover, most of the time we need to choose sets from an infinite collection which cannot be done in ZF. Additional axioms are needed, but strictly weaker than the axiom of choice.

However, if the topology has not many open sets, then the topological space is indeed compact.

theorem card˙top˙comp:
assumes Card(Q) T≺Q K⊆⋃T
shows (K)–is compact of cardinal″Q–in″T
proof-
{

fix M assume M:M⊆T K⊆⋃M
from M(1) assms(2) have M≺Q using subset˙imp˙lepoll lesspoll˙trans1 by blast
with M(2) have ∃N∈Pow(M). K⊆⋃N ∧ N≺Q by auto
}
with assms(1,3) show thesis unfolding IsCompactOfCard˙def by auto
qed

The union of two compact sets, is compact; of any cardinality.

theorem union˙compact:
assumes K–is compact of cardinal″Q–in″T K1–is compact of cardinal″Q–in″T InfCard(Q)
shows (K ∪ K1)–is compact of cardinal″Q–in″T unfolding IsCompactOfCard˙def
proof(safe)
from assms(1) show Card(Q) unfolding IsCompactOfCard˙def by auto
fix x assume x∈K then show x∈⋃T using assms(1) unfolding IsCompactOf-Card˙def by blast
next
fix x assume x∈K1 then show x∈⋃T using assms(2) unfolding IsCompactOf-Card˙def by blast
next
fix M assume M⊆T K∪K1⊆⋃M
then have K⊆⋃MK1⊆⋃M by auto
with ‹M⊆T› have ∃N∈Pow(M). K ⊆ ⋃N ∧ N ≺ Q∃N∈Pow(M). K1 ⊆ ⋃N ∧ N ≺ Q using assms unfolding IsCompactOfCard˙def
by auto
then obtain NK NK1 where NK∈Pow(M)NK1∈Pow(M)K ⊆ ⋃NKK1 ⊆ ⋃NK1NK ≺ QNK1 ≺ Q by auto
then have NK∪NK1 ≺ QK∪K1⊆⋃(NK∪NK1)NK∪NK1∈Pow(M) using assms(3) less˙less˙imp˙un˙less by auto
then show ∃N∈Pow(M). K∪K1⊆⋃N ∧ N≺Q by auto
qed

If a set is compact of cardinality Q for some topology, it is compact of cardinality Q for every coarser topology.

theorem compact˙coarser:
assumes T1⊆T and ⋃T1=⋃T and (K)–is compact of cardinal″Q–in″T
shows (K)–is compact of cardinal″Q–in″T1
proof-
{
fix M
assume AS:M∈Pow(T1)K⊆⋃M
then have M∈Pow(T)K⊆⋃M using assms(1) by auto
then have ∃N∈Pow(M).K⊆⋃N∧N≺Q using assms(3) unfolding IsCompactOf-Card˙def by auto
}
then show (K)–is compact of cardinal″Q–in″T1 using assms(3,2) unfolding Is-CompactOfCard˙def by auto
qed

If some set is compact for some cardinal, it is compact for any greater cardinal.

```
theorem compact˙greater˙card:
  assumes Q≲Q1 and (K)–is compact of cardinal″Q–in″T and Card(Q1)
  shows (K)–is compact of cardinal″Q1–in″T
proof-
  {
    fix M
    assume AS: M∈Pow(T)K⊆⋃M
    then have ∃N∈Pow(M).K⊆⋃N∧N≺Q using assms(2) unfolding IsCompactOf-
Card˙def by auto
    then have ∃N∈Pow(M).K⊆⋃N∧N≺Q1 using assms(1) lesspoll˙trans2
      unfolding IsCompactOfCard˙def by auto
  }
  then show thesis using assms(2,3) unfolding IsCompactOfCard˙def by auto
qed
```

A closed subspace of a compact space of any cardinality, is also compact of the same cardinality.

```
theorem compact˙closed:
  assumes K –is compact of cardinal″ Q –in″ T
    and R –is closed in″ T
  shows (K∩R) –is compact of cardinal″ Q –in″ T
proof-
  {
    fix M
    assume AS:M∈Pow(T)K∩R⊆⋃M
    have ⋃T-R∈T using assms(2) IsClosed˙def by auto
    have K-R⊆(⋃T-R) using assms(1) IsCompactOfCard˙def by auto
    with ‹⋃T-R∈T› have K⊆⋃(M ∪–⋃T-R″) and M ∪–⋃T-R″∈Pow(T)
    proof (safe)
      {
        fix x
        assume x∈M
        with AS(1) show x∈T by auto
      }
      {
        fix x
        assume x∈K
        have x∈R∨x∉R by auto
        with ‹x∈K› have x∈K∩R∨x∈K-R by auto
        with AS(2) ‹K-R⊆(⋃T-R)› have x∈⋃M∨x∈(⋃T-R) by auto
        then show x∈⋃(M ∪–⋃T-R″) by auto
      }
    qed
    with assms(1) have ∃N∈Pow(M∪–⋃T-R″). K ⊆ ⋃N ∧ N ≺ Q unfolding Is-
CompactOfCard˙def by auto
    then obtain N where cub:N∈Pow(M∪–⋃T-R″) K⊆⋃N N≺Q by auto
    have N-–⋃T-R″∈Pow(M)K∩R⊆⋃(N-–⋃T-R″) N-–⋃T-R″≺Q
```

```
    proof (safe)
      {
        fix x
        assume x∈Nx∉M
        then show x=⋃T-R using cub(1) by auto
      }
      {
        fix x
        assume x∈Kx∈R
        then have x∉⋃T-Rx∈K by auto
        then show x∈⋃(N-–⋃T-R″) using cub(2) by blast
      }
      have N-–⋃T-R″⊆N by auto
     with cub(3) show N-–⋃T-R″≺Q using subset˙imp˙lepoll lesspoll˙trans1 by blast
    qed
    then have ∃N∈Pow(M). K∩R⊆⋃N ∧ N≺Q by auto
  }
  then have ∀M∈Pow(T). (K ∩ R ⊆ ⋃M ⟶ (∃N∈Pow(M). K ∩ R ⊆ ⋃N ∧ N ≺
Q)) by auto
  then show thesis using IsCompactOfCard˙def assms(1) by auto
qed
```

59.2 Properties of numerability

The properties of numerability deal with cardinals of some sets built from the topology. The properties which are normally used are the ones related to the cardinal of the natural numbers or its successor.

```
definition
  IsFirstOfCard (˙ –is of first type of cardinal″˙ 90) where
  (T –is of first type of cardinal″ Q) ≡ ∀x∈⋃T. (∃B. (B –is a base for″ T) ∧ (–b∈B.
x∈b″ ≺ Q))

definition
  IsSecondOfCard (˙ –is of second type of cardinal″˙ 90) where
  (T –is of second type of cardinal″Q) ≡ (∃B. (B –is a base for″ T) ∧ (B ≺ Q))

definition
  IsSeparableOfCard (˙–is separable of cardinal″˙ 90) where
  T–is separable of cardinal″Q≡ ∃U∈Pow(⋃T). Closure(U,T)=⋃T ∧ U≺Q

definition
  IsFirstCountable (˙ –is first countable″ 90) where
  (T –is first countable″) ≡ T –is of first type of cardinal″ csucc(nat)

definition
  IsSecondCountable (˙ –is second countable″ 90) where
  (T –is second countable″) ≡ (T –is of second type of cardinal″csucc(nat))

definition
```

IsSeparable (˙–is separable″ 90) where
T–is separable″≡ T–is separable of cardinal″csucc(nat)

If a set is of second type of cardinal Q, then it is of first type of that same cardinal.

theorem second˙imp˙first:
assumes T–is of second type of cardinal″Q
shows T–is of first type of cardinal″Q
proof-
from assms have ∃B. (B –is a base for″ T) ∧ (B ≺ Q) using IsSecondOfCard˙def by auto
then obtain B where base:(B –is a base for″ T) ∧ (B ≺ Q) by auto
{
fix x
assume x∈⋃T
have –b∈B. x∈b″⊆B by auto
then have –b∈B. x∈b″≲B using subset˙imp˙lepoll by auto
with base have –b∈B. x∈b″≺Q using lesspoll˙trans1 by auto
with base have (B –is a base for″ T) ∧ –b∈B. x∈b″≺Q by auto
}
then have ∀x∈⋃T. ∃B. (B –is a base for″ T) ∧ –b∈B. x∈b″≺Q by auto
then show thesis using IsFirstOfCard˙def by auto
qed

A set is dense iff it intersects all non-empty, open sets of the topology.

lemma dense˙int˙open:
assumes T–is a topology″ and A⊆⋃T
shows Closure(A,T)=⋃T ⟷ (∀U∈T. U≠0 ⟶ A∩U≠0)
proof
assume AS:Closure(A,T)=⋃T
{
fix U
assume Uopen:U∈T and U≠0
then have U∩⋃T≠0 by auto
with AS have U∩Closure(A,T) ≠0 by auto
with assms Uopen have U∩A≠0 using topology0.cl˙inter˙neigh topology0˙def by blast
}
then show ∀U∈T. U≠0 ⟶ A∩U≠0 by auto
next
assume AS:∀U∈T. U≠0 ⟶ A∩U≠0
{
fix x
assume A:x∈⋃T
then have ∀U∈T. x∈U ⟶ U∩A≠0 using AS by auto
with assms A have x∈Closure(A,T) using topology0.inter˙neigh˙cl topology0˙def by auto
}
then have ⋃T⊆Closure(A,T) by auto

with assms show Closure(A,T)=⋃T using topology0.Top˙3˙L11(1) topology0˙def by blast
qed

59.3 Relations between numerability properties and choice principles

It is known that some statements in topology aren't just derived from choice axioms, but also equivalent to them. Here is an example

The following are equivalent:

- Every topological space of second cardinality csucc(Q) is separable of cardinality csucc(Q).
- The axiom of Q choice.

In the article [4] there is a proof of this statement for Q= ℕ, with more equivalences.

If a topology is of second type of cardinal csucc(Q), then it is separable of the same cardinal. This result makes use of the axiom of choice for the cardinal Q on subsets of ⋃T.

theorem Q˙choice˙imp˙second˙imp˙separable:
assumes T–is of second type of cardinal″csucc(Q)
and –the axiom of″ Q –choice holds for subsets″ ⋃T
and T–is a topology″
shows T–is separable of cardinal″csucc(Q)
proof-
from assms(1) have ∃B. (B –is a base for″ T) ∧ (B ≺ csucc(Q)) using IsSecondOfCard˙def by auto
then obtain B where base:(B –is a base for″ T) ∧ (B ≺ csucc(Q)) by auto
let N=λb∈B. b
let B=B-–0″
have B-–0″⊆B by auto
with base have prec:B-–0″≺csucc(Q) using subset˙imp˙lepoll lesspoll˙trans1 by blast
from base have baseOpen:∀b∈B. Nb∈T using base˙sets˙open by auto
from assms(2) have car:Card(Q) and reg:(∀ M N. (M ≲Q ∧ (∀t∈M. Nt≠0 ∧ Nt⊆⋃T)) ⟶ (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)))
using AxiomCardinalChoice˙def by auto
then have (B ≲Q ∧ (∀t∈B. Nt≠0 ∧ Nt⊆⋃T)) ⟶ (∃f. f:Pi(B,λt. Nt) ∧ (∀t∈B. ft∈Nt)) by blast
with prec have (∀t∈B. Nt⊆⋃T) ⟶ (∃f. f:Pi(B,λt. Nt) ∧ (∀t∈B. ft∈Nt)) using Card˙less˙csucc˙eq˙le car by auto
with baseOpen have ∃f. f:Pi(B,λt. Nt) ∧ (∀t∈B. ft∈Nt) by blast
then obtain f where f:f:Pi(B,λt. Nt) and f2:∀t∈B. ft∈Nt by auto
{
fix U
assume U∈T and U≠0

```
    then obtain b where A1:b∈B-–0″ and b⊆U using Top˙1˙2˙L1 base by blast
    with f2 have fb∈U by auto
    with A1 have –fb. b∈B″∩U≠0 by auto
  }
  then have r:∀U∈T. U≠0 ⟶ –fb. b∈B″∩U≠0 by auto
  have –fb. b∈B″⊆⋃T using f2 baseOpen by auto
  moreover
  with r have Closure(–fb. b∈B″,T)=⋃T using dense˙int˙open assms(3) by auto
  moreover
  have ffun:f:B→range(f) using f range˙of˙fun by auto
  then have f∈surj(B,range(f)) using fun˙is˙surj by auto
 then have des1:range(f)≲B using surj˙fun˙inv˙2[of fBrange(f)Q] prec Card˙less˙csucc˙eq˙le
car
    Card˙is˙Ord by auto
  then have –fb. b∈B″⊆range(f) using apply˙rangeI[OF ffun] by auto
  then have –fb. b∈B″≲range(f) using subset˙imp˙lepoll by auto
  with des1 have –fb. b∈B″≲B using lepoll˙trans by blast
  with prec have –fb. b∈B″≺csucc(Q) using lesspoll˙trans1 by auto
  ultimately show thesis using IsSeparableOfCard˙def by auto
qed
```

The next theorem resolves that the axiom of Q choice for subsets of ⋃T is necessary for second type spaces to be separable of the same cardinal csucc(Q).

```
theorem second˙imp˙separable˙imp˙Q˙choice:
  assumes ∀T. (T–is a topology″ ∧ (T–is of second type of cardinal″csucc(Q))) ⟶
(T–is separable of cardinal″csucc(Q))
  and Card(Q)
  shows –the axiom of″ Q –choice holds″
proof-
  {
    fix N M
    assume AS:M ≲Q ∧  (∀t∈M. Nt≠0)

    then obtain h where inj:h∈inj(M,Q) using lepoll˙def by auto
    then have bij:converse(h):bij(range(h),M) using inj˙bij˙range bij˙converse˙bij by
auto
    let T=–(N(converse(h)i))×–i″. i∈range(h)″
    {
      fix j
      assume AS2:j∈range(h)
      from bij have converse(h):range(h)→M using bij˙def inj˙def by auto
      with AS2 have converse(h)j∈M by simp
      with AS have N(converse(h)j)≠0 by auto
      then have (N(converse(h)j))×–j″≠0 by auto
    }
    then have noEmpty:0∉T by auto
    moreover
    {
```

fix A B
assume AS2:A∈TB∈TA∩B≠0
then obtain j t where A˙def:A=N(converse(h)j)×–j″ and B˙def:B=N(converse(h)t)×–t″
and Range:j∈range(h) t∈range(h) by auto
from AS2(3) obtain x where x∈A∩B by auto
with A˙def B˙def have j=t by auto
with A˙def B˙def have A=B by auto
}
then have (∀A∈T. ∀B∈T. A=B∨ A∩B=0) by auto
ultimately
have Part:T –is a partition of″ ⋃T unfolding IsAPartition˙def by auto
let τ=PTopology ⋃T T
from Part have top:τ –is a topology″ and base:T –is a base for″τ
using Ptopology˙is˙a˙topology by auto
let f=–⟨i,(N(converse(h)i))×–i″⟩. i∈range(h)″
have f:range(h)→T using functionI[of f] Pi˙def by auto
then have f∈surj(range(h),T) unfolding surj˙def using apply˙equality by auto
moreover
have range(h)⊆Q using inj unfolding inj˙def range˙def domain˙def Pi˙def by auto
ultimately have T≲ Q using surj˙fun˙inv[of frange(h)TQ] assms(2) Card˙is˙Ord lepoll˙trans
subset˙imp˙lepoll by auto
then have T≺csucc(Q) using Card˙less˙csucc˙eq˙le assms(2) by auto
with base have (τ–is of second type of cardinal″csucc(Q)) using IsSecondOf-Card˙def by auto
with top have τ–is separable of cardinal″csucc(Q) using assms(1) by auto
then obtain D where sub:D∈Pow(⋃τ) and clos:Closure(D,τ)=⋃τ and cardd:D≺csucc(Q)
using IsSeparableOfCard˙def by auto

then have D≲Q using Card˙less˙csucc˙eq˙le assms(2) by auto
then obtain r where r:r∈inj(D,Q) using lepoll˙def by auto
then have bij2:converse(r):bij(range(r),D) using inj˙bij˙range bij˙converse˙bij by auto
then have surj2:converse(r):surj(range(r),D) using bij˙def by auto
let R=λi∈range(h). –j∈range(r). converse(r)j∈((N(converse(h)i))×–i″)″
{
fix i
assume AS:i∈range(h)
then have T:(N(converse(h)i))×–i″∈T by auto
then have P: (N(converse(h)i))×–i″∈τ using base unfolding IsAbaseFor˙def by blast
with top sub clos have ∀U∈τ. U≠0 ⟶ D∩U≠0 using dense˙int˙open by auto
with P have (N(converse(h)i))×–i″≠0 ⟶ D∩(N(converse(h)i))×–i″≠0 by auto
with T noEmpty have D∩(N(converse(h)i))×–i″≠0 by auto
then obtain x where x∈D and px:x∈(N(converse(h)i))×–i″ by auto
with surj2 obtain j where j∈range(r) and converse(r)j=x unfolding surj˙def by blast
with px have j∈–j∈range(r). converse(r)j∈((N(converse(h)i))×–i″)″ by auto
then have Ri≠0 using beta˙if[of range(h) ˙ i] AS by auto

```
}
then have nonE:∀i∈range(h). Ri≠0 by auto
{
  fix i j
  assume i:i∈range(h) and j:j∈Ri
  from j i have converse(r)j∈((N(converse(h)i))×–i″) using beta˙if by auto
}
then have pp:∀i∈range(h). ∀j∈Ri. converse(r)j∈((N(converse(h)i))×–i″) by auto
let E=–⟨m,fst(converse(r)(μ j. j∈R(hm)))⟩. m∈M″
have ff:function(E) unfolding function˙def by auto
moreover

{
  fix m
  assume M:m∈M
  with inj have hm:hm∈range(h) using apply˙rangeI inj˙def by auto
  {
    fix j
    assume j∈R(hm)
    with hm have j∈range(r) using beta˙if by auto
    from r have r:surj(D,range(r)) using fun˙is˙surj inj˙def by auto
    with ‹j∈range(r)› obtain d where d∈D and rd=j using surj˙def by auto
    then have j∈Q using r inj˙def by auto
  }
  then have subcar:R(hm)⊆Q by blast
  from nonE hm obtain ee where P:ee∈R(hm) by blast
  with subcar have ee∈Q by auto
  then have Ord(ee) using assms(2) Card˙is˙Ord Ord˙in˙Ord by auto
with P have (μ j. j∈R(hm))∈R(hm) using LeastI[where i=ee and P=λj. j∈R(hm)]
  by auto
  with pp hm have converse(r)(μ j. j∈R(hm))∈((N(converse(h)(hm)))×–(hm)″)
by auto
  then have converse(r)(μ j. j∈R(hm))∈((N(m))×–(hm)″) using left˙inverse[OF
inj M]
    by simp
  then have fst(converse(r)(μ j. j∈R(hm)))∈(N(m)) by auto
  }
 ultimately have thesis1:∀m∈M. Em∈(N(m)) using function˙apply˙equality by
auto
{
  fix e
  assume e∈E
  then obtain m where m∈M and e=⟨m,Em⟩ using function˙apply˙equality ff by
auto
  with thesis1 have e∈Sigma(M,λt. Nt) by auto
}
then have E∈Pow(Sigma(M,λt. Nt)) by auto
with ff have E∈Pi(M,λm. Nm) using Pi˙iff by auto
then have (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)) using thesis1 by auto
```

```
  }
  then show thesis using AxiomCardinalChoiceGen˙def assms(2) by auto
qed
```

Here is the equivalence from the two previous results.

```
theorem Q˙choice˙eq˙secon˙imp˙sepa:
  assumes Card(Q)
  shows (∀T. (T–is a topology″ ∧ (T–is of second type of cardinal″csucc(Q))) �longrightarrow
(T–is separable of cardinal″csucc(Q)))
    ⟷(–the axiom of″ Q –choice holds″)
  using Q˙choice˙imp˙second˙imp˙separable choice˙subset˙imp˙choice
  using second˙imp˙separable˙imp˙Q˙choice assms by auto
```

Given a base injective with a set, then we can find a base whose elements are indexed by that set.

```
lemma base˙to˙indexed˙base:
  assumes B ≲Q B –is a base for″T
  shows ∃N. –Ni. i∈Q″–is a base for″T
proof-
  from assms obtain f where f˙def:f∈inj(B,Q) unfolding lepoll˙def by auto
  let ff=–⟨b,fb⟩. b∈B″
  have domain(ff)=B by auto
  moreover
  have relation(ff) unfolding relation˙def by auto
  moreover
  have function(ff) unfolding function˙def by auto
  ultimately
  have fun:ff:B→range(ff) using function˙imp˙Pi[of ff] by auto
  then have injj:ff∈inj(B,range(ff)) unfolding inj˙def
  proof
    {
      fix w x
      assume AS:w∈Bx∈B–⟨b, f  b⟩ . b ∈ B″  w = –⟨b, f  b⟩ . b ∈ B″  x
      then have fw=fx using apply˙equality[OF ˙ fun] by auto
      then have w=x using f˙def inj˙def AS(1,2) by auto
    }
    then show ∀w∈B. ∀x∈B. –⟨b, f  b⟩ . b ∈ B″  w = –⟨b, f  b⟩ . b ∈ B″  x ⟶ w
= x by auto
  qed
  then have bij:ff∈bij(B,range(ff)) using inj˙bij˙range by auto
  from fun have range(ff)=–fb. b∈B″ by auto
  with f˙def have ran:range(ff)⊆Q using inj˙def by auto
  let N=–⟨i,(if i∈range(ff) then converse(ff)i else 0)⟩. i∈Q″
  have FN:function(N) unfolding function˙def by auto
  have B ⊆–Ni. i∈Q″
  proof
    fix t
    assume a:t∈B
    from bij have rr:ff:B→range(ff) unfolding bij˙def inj˙def by auto
```

```
    have ig:fft=ft using a apply˙equality[OF ˙ rr] by auto
    have r:fft∈range(ff) using apply˙type[OF rr a].
    from ig have t:fft∈Q using apply˙type[OF ˙ a] f˙def unfolding inj˙def by auto
     with r have N(fft)=converse(ff)(fft) using function˙apply˙equality[OF ˙ FN] by
auto
    then have N(fft)=t using left˙inverse[OF injj a] by auto
    then have t=N(fft) by auto
    then have ∃i∈Q. t=Ni using t(1) by auto
    then show t∈–Ni. i∈Q″ by simp
  qed
  moreover
  have ∀r∈–Ni. i∈Q″-B. r=0
  proof
    fix r
    assume r∈–Ni. i∈Q″-B
    then obtain j where R:j∈Qr=Njr∉B by auto
    {
      assume AS:j∈range(ff)
       with R(1) have Nj=converse(ff)j using function˙apply˙equality[OF ˙ FN] by
auto
    then have Nj∈B using apply˙funtype[OF inj˙is˙fun[OF bij˙is˙inj[OF bij˙converse˙bij[OF
bij]]] AS]
      by auto
      then have False using R(3,2) by auto
    }
    then have j∉range(ff) by auto
    then show r=0 using function˙apply˙equality[OF ˙ FN] R(1,2) by auto
  qed
  ultimately have –Ni. i∈Q″=B∨–Ni. i∈Q″=B ∪–0″ by blast
  moreover
  have (B ∪–0″)-–0″=B-–0″ by blast
  then have (B ∪–0″)-–0″ –is a base for″T using base˙no˙0[of BT] assms(2) by auto
  then have B ∪–0″ –is a base for″T using base˙no˙0[of B ∪–0″T] by auto
  ultimately
  have –Ni. i∈Q″–is a base for″T using assms(2) by auto
  then show thesis by auto
qed
```

59.4 Relation between numerability and compactness

If the axiom of Q choice holds, then any topology of second type of cardinal csucc(Q) is compact of cardinal csucc(Q)

```
theorem compact˙of˙cardinal˙Q:
  assumes –the axiom of″ Q –choice holds for subsets″ (Pow(Q))
    T–is of second type of cardinal″csucc(Q)
    T–is a topology″
  shows ((⋃T)–is compact of cardinal″csucc(Q)–in″T)
proof-
 from assms(1) have CC:Card(Q) and reg:⋀M N. (M ≲Q ∧ (∀t∈M. Nt≠0∧Nt⊆Pow(Q)))
```

```
⟶ (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)) using
  AxiomCardinalChoice˙def by auto
   from assms(2) obtain R where R≲QR–is a base for″T unfolding IsSecondOf-
Card˙def using Card˙less˙csucc˙eq˙le CC by auto
  with base˙to˙indexed˙base obtain N where base:–Ni. i∈Q″–is a base for″T  by blast
  {
    fix M
    assume A:⋃T⊆⋃MM∈Pow(T)
    let α=λU∈M. –i∈Q. N(i)⊆U″
    have inj:α∈inj(M,Pow(Q)) unfolding inj˙def
    proof
    {
      show (λU∈M. –i ∈ Q . N  i ⊆ U″) ∈ M → Pow(Q) using lam˙type[of MλU. –i
∈ Q . N(i) ⊆ U″%t. Pow(Q)] by auto
      {
        fix w x
        assume AS:w∈Mx∈M–i ∈ Q . N(i) ⊆ w″ = –i ∈ Q . N(i) ⊆ x″
        from AS(1,2) A(2) have w∈Tx∈T by auto
     then have w=Interior(w,T)x=Interior(x,T) using assms(3) topology0.Top˙2˙L3[of
T]
          topology0˙def[of T] by auto
        then have UN:w=(⋃–B∈–N(i). i∈Q″. B⊆w″)x=(⋃–B∈–N(i). i∈Q″. B⊆x″)
          using interior˙set˙base˙topology assms(3) base by auto
        {
          fix b
          assume b∈w
          then have b∈⋃–B∈–N(i). i∈Q″. B⊆w″ using UN(1) by auto
          then obtain S where S:S∈–N(i). i∈Q″ b∈S S⊆w by blast
          then obtain j where j:j∈QS=N(j) by auto
          then have j∈–i ∈ Q . N(i) ⊆ w″ using S(3) by auto
          then have N(j)⊆xb∈N(j)j∈Q using S(2) AS(3) j by auto
          then have b∈(⋃–B∈–N(i). i∈Q″. B⊆x″) by auto
          then have b∈x using UN(2) by auto
        }
        moreover
        {
          fix b
          assume b∈x
          then have b∈⋃–B∈–N(i). i∈Q″. B⊆x″ using UN(2) by auto
          then obtain S where S:S∈–N(i). i∈Q″ b∈S S⊆x by blast
          then obtain j where j:j∈QS=N(j) by auto
          then have j∈–i ∈ Q . N(i) ⊆ x″ using S(3) by auto
          then have j∈–i ∈ Q . N(i) ⊆ w″ using AS(3) by auto
          then have N(j)⊆wb∈N(j)j∈Q using S(2) j(2) by auto
          then have b∈(⋃–B∈–N(i). i∈Q″. B⊆w″) by auto
          then have b∈w using UN(2) by auto
        }
        ultimately have w=x by auto
      }
```

then show ∀w∈M. ∀x∈M. (λU∈M. –i ∈ Q . N i ⊆ U″) w = (λU∈M. –i ∈ Q . N i ⊆ U″) x ⟶ w = x by auto
}
qed
let X=λi∈Q. –αU. U∈–V∈M. N(i)⊆V″″
let M=–i∈Q. Xi≠0″
have subMQ:M⊆Q by auto
then have ddd:M ≲Q using subset˙imp˙lepoll by auto
then have M ≲Q∀i∈M. Xi≠0∀i∈M. Xi⊆Pow(Q) by auto
then have M ≲Q∀i∈M. Xi≠0∀i∈M. Xi ≲ Pow(Q) using subset˙imp˙lepoll by auto
then have (∃f. f:Pi(M,λt. Xt) ∧ (∀t∈M. ft∈Xt)) using reg[of MX] by auto
then obtain f where f:f:Pi(M,λt. Xt)(!!t. t∈M ⟹ ft∈Xt) by auto
{
fix m
assume S:m∈M
from f(2) S obtain YY where YY:(YY∈M) (fm=αYY) by auto
then have Y:(YY∈M)∧(fm=αYY) by auto
moreover
{
fix U
assume U∈M∧(fm=αU)
then have U=YY using inj inj˙def YY by auto
}
then have r:⋀x. x∈M∧(fm=αx) ⟹ x=YY by blast
have ∃!YY. YY∈M ∧ fm=αYY using ex1I[of %Y. Y∈M∧ fm=αY,OF Y r] by auto
}
then have ex1YY:∀m∈M. ∃!YY. YY∈M ∧ fm=αYY by auto
let YYm=–⟨m,(THE YY. YY∈M ∧ fm=αYY)⟩. m∈M″
have aux:⋀m. m∈M ⟹ YYmm=(THE YY. YY∈M ∧ fm=αYY) unfolding apply˙def by auto
have ree:∀m∈M. (YYmm)∈M ∧ fm=α(YYmm)
proof
fix m
assume C:m∈M
then have ∃!YY. YY∈M ∧ fm=αYY using ex1YY by auto
then have (THE YY. YY∈M ∧ fm=αYY)∈M∧fm=α(THE YY. YY∈M ∧ fm=αYY)
using theI[of %Y. Y∈M∧ fm=αY] by blast
then show (YYmm)∈M ∧ fm=α(YYmm) apply (simp only: aux[OF C]) done
qed
have tt:⋀m. m∈M ⟹ N(m)⊆YYmm
proof-
fix m
assume D:m∈M
then have QQ:m∈Q by auto
from D have t:(YYmm)∈M ∧ fm=α(YYmm) using ree by blast
then have fm=α(YYmm) by blast

```
  then have (α(YYmm))∈(λi∈Q. −αU. U∈−V∈M. N(i)⊆V″″)m using f(2)[OF D]
    by auto
  then have (α(YYmm))∈−αU. U∈−V∈M. N(m)⊆V″″ using QQ by auto
  then obtain U where U∈−V∈M. N(m)⊆V″α(YYmm)=αU by auto
  then have r:U∈MN(m)⊆Uα(YYmm)=αU(YYmm)∈M using t by auto
  then have YYmm=U using  inj˙apply˙equality[OF inj] by blast
  then show N(m)⊆YYmm using r by auto
qed
then have (⋃m∈M. N(m))⊆(⋃m∈M. YYmm)
proof-
  {
    fix s
    assume s∈(⋃m∈M. N(m))
    then obtain t where r:t∈Ms∈N(t) by auto
    then have s∈YYmt using tt[OF r(1)] by blast
    then have s∈(⋃m∈M. YYmm) using r(1) by blast
  }
  then show thesis by blast
qed
moreover
{
  fix x
  assume AT:x∈⋃T
  with A obtain U where BB:U∈MU∈Tx∈U by auto
   then obtain j where BC:j∈Q N(j)⊆Ux∈N(j) using point˙open˙base˙neigh[OF
base,of Ux] by auto
  then have Xj≠0 using BB(1) by auto
  then have j∈M using BC(1) by auto
  then have x∈(⋃m∈M. N(m)) using BC(3) by auto
}
then have ⋃T⊆(⋃m∈M. N(m)) by blast
ultimately have covers:⋃T⊆(⋃m∈M. YYmm) using subset˙trans[of ⋃T(⋃m∈M.
N(m))(⋃m∈M. YYmm)]
  by auto
have relation(YYm) unfolding relation˙def by auto
moreover
have f:function(YYm) unfolding function˙def by auto
moreover
have d:domain(YYm)=M by auto
moreover
have r:range(YYm)=YYmM by auto
ultimately
have fun:YYm:M→YYmM using function˙imp˙Pi[of YYm] by auto
have YYm∈surj(M,YYmM) using fun˙is˙surj[OF fun] r by auto
with surj˙fun˙inv[OF this subMQ Card˙is˙Ord[OF CC]]
have YYmM ≲ M by auto
with ddd have Rw:YYmM ≲Q using lepoll˙trans by blast
{
  fix m assume m∈M
```

then have ⟨m,YYmm⟩∈YYm using function˙apply˙Pair[OF f] d by blast
then have YYmm∈YYmM by auto}
then have l1:–YYmm. m∈M″⊆YYmM by blast
{
fix t assume t∈YYmM
then have ∃x∈M. ⟨x,t⟩∈YYm unfolding image˙def by auto
then obtain r where S:r∈M⟨r,t⟩∈YYm by auto
have YYmr=t using apply˙equality[OF S(2) fun] by auto
with S(1) have t∈–YYmm. m∈M″ by auto
}
with l1 have –YYmm. m∈M″=YYmM by blast
with Rw have –YYmm. m∈M″ ≲Q by auto
with covers have –YYmm. m∈M″∈Pow(M)∧⋃T⊆⋃–YYmm. m∈M″∧–YYmm. m∈M″ ≺csucc(Q) using ree
Card˙less˙csucc˙eq˙le[OF CC] by blast
then have ∃N∈Pow(M). ⋃T⊆⋃N∧N≺csucc(Q) by auto
}
then have ∀M∈Pow(T). ⋃T ⊆ ⋃M ⟶ (∃N∈Pow(M). ⋃T ⊆ ⋃N ∧ N ≺ csucc(Q)) by auto
then show thesis using IsCompactOfCard˙def Card˙csucc CC Card˙is˙Ord by auto
qed

In the following proof, we have chosen an infinite cardinal to be able to apply the equation Q × Q ≈ Q. For finite cardinals; both, the assumption and the axiom of choice, are always true.

theorem second˙imp˙compact˙imp˙Q˙choice˙PowQ:
assumes ∀T. (T–is a topology″ ∧ (T–is of second type of cardinal″csucc(Q))) ⟶ ((⋃T)–is compact of cardinal″csucc(Q)–in″T)
and InfCard(Q)
shows –the axiom of″ Q –choice holds for subsets″ (Pow(Q))
proof-
{
fix N M
assume AS:M ≲Q ∧ (∀t∈M. Nt≠0 ∧ Nt⊆Pow(Q))
then obtain h where h∈inj(M,Q) using lepoll˙def by auto

have discTop:Pow(Q×M) –is a topology″ using Pow˙is˙top by auto
{
fix A
assume AS:A∈Pow(Q×M)
have A=⋃—i″. i∈A″ by auto
with AS have ∃T∈Pow(—i″. i∈Q×M″). A=⋃T by auto
then have A∈–⋃U. U∈Pow(—i″. i∈Q×M″)″ by auto
}
moreover
{
fix A
assume AS:A∈–⋃U. U∈Pow(—i″. i∈Q×M″)″
then have A∈Pow(Q×M) by auto

}
ultimately
have base:—x″. x∈Q×M″ –is a base for″ Pow(Q×M) unfolding IsAbaseFor˙def by blast
let f=–⟨i,–i″⟩. i∈Q×M″
have fff:f∈Q×M→—i″. i∈Q×M″ using Pi˙def function˙def by auto
then have f∈inj(Q×M,—i″. i∈Q×M″) unfolding inj˙def using apply˙equality by auto
then have f∈bij(Q×M,—i″. i∈Q×M″) unfolding bij˙def surj˙def using fff
apply˙equality fff by auto
then have Q×M≈—i″. i∈Q×M″ using eqpoll˙def by auto
then have —i″. i∈Q×M″≈Q×M using eqpoll˙sym by auto
then have —i″. i∈Q×M″≲Q×M using eqpoll˙imp˙lepoll by auto
then have —i″. i∈Q×M″≲Q×Q using AS prod˙lepoll˙mono[of QQMQ] lepoll˙refl[of Q]
lepoll˙trans by blast
then have —i″. i∈Q×M″≲Q using InfCard˙square˙eqpoll assms(2) lepoll˙eq˙trans by auto
then have —i″. i∈Q×M″≺csucc(Q) using Card˙less˙csucc˙eq˙le assms(2) InfCard˙is˙Card by auto
then have Pow(Q×M) –is of second type of cardinal″ csucc(Q) using IsSecondOfCard˙def base by auto
then have comp:(Q×M) –is compact of cardinal″csucc(Q)–in″Pow(Q×M) using discTop assms(1) by auto
{
fix W
assume W∈Pow(Q×M)
then have T:W–is closed in″ Pow(Q×M) and (Q×M)∩W=W using IsClosed˙def by auto
with compact˙closed[OF comp T] have (W –is compact of cardinal″csucc(Q)–in″Pow(Q×M)) by auto
}
then have subCompact:∀W∈Pow(Q×M). (W –is compact of cardinal″csucc(Q)–in″Pow(Q×M)) by auto
let cub=⋃—(U)×–t″. U∈Nt″. t∈M″
from AS have (⋃cub)∈Pow((Q)×M) by auto
with subCompact have Ncomp:((⋃cub) –is compact of cardinal″csucc(Q)–in″Pow(Q×M)) by auto
have cond:(cub)∈Pow(Pow(Q×M))∧ ⋃cub⊆⋃cub using AS by auto
have ∃S∈Pow(cub). (⋃cub) ⊆ ⋃S ∧ S ≺ csucc(Q)
proof-
{
have ((⋃cub) –is compact of cardinal″csucc(Q)–in″Pow(Q×M)) using Ncomp by auto
then have ∀M∈Pow(Pow(Q×M)). ⋃cub ⊆ ⋃M ⟶ (∃Na∈Pow(M). ⋃cub ⊆ ⋃Na ∧ Na ≺ csucc(Q))
unfolding IsCompactOfCard˙def by auto
with cond have ∃S∈Pow(cub). ⋃cub ⊆ ⋃S ∧ S ≺ csucc(Q) by auto
}

then show thesis by auto
qed
then have ttt:∃S∈Pow(cub). (⋃cub) ⊆ ⋃S ∧ S ≲ Q using Card˙less˙csucc˙eq˙le assms(2) InfCard˙is˙Card by auto
then obtain S where S˙def:S∈Pow(cub)(⋃cub) ⊆ ⋃S S ≲ Q by auto
{
fix t
assume AA:t∈MNt≠–0″
from AA(1) AS have Nt≠0 by auto
with AA(2) obtain U where G:U∈Nt and notEm:U≠0 by blast
then have U×–t″∈cub using AA by auto
then have U×–t″⊆⋃cub by auto
with G notEm AA have ∃s. ⟨s,t⟩∈⋃cub by auto
}
then have ∀t∈M. (Nt≠–0″)⟶ (∃s. ⟨s,t⟩∈⋃cub) by auto
then have A:∀t∈M. (Nt≠–0″)⟶ (∃s. ⟨s,t⟩∈⋃S) using S˙def(2) by blast
from S˙def(1) have B:∀f∈S. ∃t∈M. ∃U∈Nt. f=U×–t″ by blast
from A B have ∀t∈M. (Nt≠–0″)⟶ (∃U∈Nt. U×–t″∈S) by blast
then have noEmp:∀t∈M. (Nt≠–0″)⟶ (S∩(–U×–t″. U∈Nt″)≠0) by auto
from S˙def(3) obtain r where r:r:inj(S,Q) using lepoll˙def by auto
then have bij2:converse(r):bij(range(r),S) using inj˙bij˙range bij˙converse˙bij by auto
then have surj2:converse(r):surj(range(r),S) using bij˙def by auto
let R=λt∈M. –j∈range(r). converse(r)j∈(–U×–t″. U∈Nt″)″
{
fix t
assume AA:t∈MNt≠–0″
then have (S∩(–U×–t″. U∈Nt″)≠0) using noEmp by auto
then obtain s where ss:s∈Ss∈–U×–t″. U∈Nt″ by blast
then obtain j where converse(r)j=s j∈range(r) using surj2 unfolding surj˙def by blast
then have j∈–j∈range(r). converse(r)j∈(–U×–t″. U∈Nt″)″ using ss by auto
then have Rt≠0 using beta˙if AA by auto
}
then have nonE:∀t∈M. Nt≠–0″⟶Rt≠0 by auto
{
fix t j
assume t∈Mj∈Rt
then have converse(r)j∈–U×–t″. U∈Nt″ using beta˙if by auto
}
then have pp:∀t∈M. ∀j∈Rt. converse(r)j∈–U×–t″. U∈Nt″ by auto
have reg:∀t U V. U×–t″=V×–t″⟶U=V
proof-
{
fix t U V
assume AA:U×–t″=V×–t″
{
fix v
assume v∈V

```
        then have ⟨v,t⟩∈V ×–t″ by auto
        then have ⟨v,t⟩∈U ×–t″ using AA by auto
        then have v∈U by auto
      }
      then have V⊆U by auto
      moreover
      {
        fix u
        assume u∈U
        then have ⟨u,t⟩∈U ×–t″ by auto
        then have ⟨u,t⟩∈V ×–t″ using AA by auto
        then have u∈V by auto
      }
      then have U⊆V by auto
      ultimately have U=V by auto
    }
    then show thesis by auto
  qed

  let E=–⟨t,if Nt=–0″ then 0 else (THE U. converse(r)(μ j. j∈Rt)=U×–t″)⟩. t∈M″
   have ff:function(E) unfolding function˙def by auto
   moreover
   {
     fix t
     assume pm:t∈M
      { assume nonEE:Nt≠–0″
     {
       fix j
       assume j∈Rt
       with pm(1) have j∈range(r) using beta˙if by auto
       from r have r:surj(S,range(r)) using fun˙is˙surj inj˙def by auto
       with ‹j∈range(r)› obtain d where d∈S and rd=j using surj˙def by auto
       then have j∈Q using r inj˙def by auto
       }
     then have sub:Rt⊆Q by blast
     from nonE pm nonEE obtain ee where P:ee∈Rt by blast
     with sub have ee∈Q by auto
     then have Ord(ee) using assms(2) Card˙is˙Ord Ord˙in˙Ord InfCard˙is˙Card by
blast
     with P have (μ j. j∈Rt)∈Rt using LeastI[where i=ee and P=λj. j∈Rt] by auto
     with pp pm have converse(r)(μ j. j∈Rt)∈–U×–t″. U∈Nt″ by auto
     then obtain W where converse(r)(μ j. j∈Rt)=W×–t″ and s:W∈Nt by auto
     then have (THE U. converse(r)(μ j. j∈Rt)=U×–t″)=W using reg by auto
     with s have (THE U. converse(r)(μ j. j∈Rt)=U×–t″)∈Nt by auto
   }
   then have (if Nt=–0″ then 0 else (THE U. converse(r)(μ j. j∈Rt)=U×–t″))∈Nt
by auto
   }
   ultimately have thesis1:∀t∈M. Et∈Nt using function˙apply˙equality by auto
```

```
  {
    fix e
    assume e∈E
    then obtain m where m∈M and e=⟨m,Em⟩ using function˙apply˙equality ff by
auto
    with thesis1 have e∈Sigma(M,λt. Nt) by auto
  }
  then have E∈Pow(Sigma(M,λt. Nt)) by auto
  with ff have E∈Pi(M,λm. Nm) using Pi˙iff by auto
  then have (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)) using thesis1 by auto}
   then show thesis using AxiomCardinalChoice˙def assms(2) InfCard˙is˙Card by
auto
qed
```

The two previous results, state the following equivalence:

```
theorem Q˙choice˙Pow˙eq˙secon˙imp˙comp:
 assumes InfCard(Q)
 shows  (∀T. (T–is a topology″ ∧ (T–is of second type of cardinal″csucc(Q))) ⟶
((⋃T)–is compact of cardinal″csucc(Q)–in″T))
   ⟷(–the axiom of″ Q –choice holds for subsets″ (Pow(Q)))
   using second˙imp˙compact˙imp˙Q˙choice˙PowQ compact˙of˙cardinal˙Q assms by
auto
```

In the next result we will prove that if the space $(\kappa, Pow(\kappa))$, for κ an infinite cardinal, is compact of its successor cardinal; then all topologycal spaces which are of second type of the successor cardinal of κ are also compact of that cardinal.

```
theorem Q˙csuccQ˙comp˙eq˙Q˙choice˙Pow:
 assumes InfCard(Q) (Q)–is compact of cardinal″csucc(Q)–in″Pow(Q)
 shows ∀T. (T–is a topology″ ∧ (T–is of second type of cardinal″csucc(Q))) ⟶
((⋃T)–is compact of cardinal″csucc(Q)–in″T)
proof
 fix T
 {
   assume top:T –is a topology″ and sec:T–is of second type of cardinal″csucc(Q)
    from assms have Card(csucc(Q)) Card(Q) using InfCard˙is˙Card Card˙is˙Ord
Card˙csucc by auto
   moreover
   have ⋃T⊆⋃T by auto
   moreover
   {
     fix M
     assume MT:M∈Pow(T) and cover:⋃T⊆⋃M
      from sec obtain B where B –is a base for″ T B≺csucc(Q) using IsSecondOf-
Card˙def by auto
   with ‹Card(Q)› obtain N where base:–Ni. i∈Q″–is a base for″T using Card˙less˙csucc˙eq˙le
       base˙to˙indexed˙base by blast
     let S=–⟨u,–i∈Q. Ni⊆u″⟩. u∈M″
     have function(S) unfolding function˙def by auto
```

then have S:M→Pow(Q) using Pi˙iff by auto
then have S∈inj(M,Pow(Q)) unfolding inj˙def
proof
{
fix w x
assume AS:w∈Mx∈M–⟨u, –i ∈ Q . N i ⊆ u″⟩ . u ∈ M″ w = –⟨u, –i ∈ Q . N i ⊆ u″⟩ . u ∈ M″ x
with ‹S:M→Pow(Q)› have ASS:–i ∈ Q . N i ⊆ w″=–i ∈ Q . N i ⊆ x″ using apply˙equality by auto
from AS(1,2) MT have w∈Tx∈T by auto
then have w=Interior(w,T)x=Interior(x,T) using top topology0.Top˙2˙L3[of T]
topology0˙def[of T] by auto
then have UN:w=(⋃–B∈–N(i). i∈Q″. B⊆w″)x=(⋃–B∈–N(i). i∈Q″. B⊆x″)
using interior˙set˙base˙topology top base by auto
{
fix b
assume b∈w
then have b∈⋃–B∈–N(i). i∈Q″. B⊆w″ using UN(1) by auto
then obtain S where S:S∈–N(i). i∈Q″ b∈S S⊆w by blast
then obtain j where j:j∈QS=N(j) by auto
then have j∈–i ∈ Q . N(i) ⊆ w″ using S(3) by auto
then have N(j)⊆xb∈N(j)j∈Q using S(2) ASS j by auto
then have b∈(⋃–B∈–N(i). i∈Q″. B⊆x″) by auto
then have b∈x using UN(2) by auto
}
moreover
{
fix b
assume b∈x
then have b∈⋃–B∈–N(i). i∈Q″. B⊆x″ using UN(2) by auto
then obtain S where S:S∈–N(i). i∈Q″ b∈S S⊆x by blast
then obtain j where j:j∈QS=N(j) by auto
then have j∈–i ∈ Q . N(i) ⊆ x″ using S(3) by auto
then have j∈–i ∈ Q . N(i) ⊆ w″ using ASS by auto
then have N(j)⊆wb∈N(j)j∈Q using S(2) j(2) by auto
then have b∈(⋃–B∈–N(i). i∈Q″. B⊆w″) by auto
then have b∈w using UN(2) by auto
}
ultimately have w=x by auto
}
then show ∀w∈M. ∀x∈M. –⟨u, –i ∈ Q . N i ⊆ u″⟩ . u ∈ M″ w = –⟨u, –i ∈ Q . N i ⊆ u″⟩ . u ∈ M″ x ⟶ w = x by auto
qed
then have S∈bij(M,range(S)) using fun˙is˙surj unfolding bij˙def inj˙def surj˙def by force
have range(S)⊆Pow(Q) by auto
then have range(S)∈Pow(Pow(Q)) by auto
moreover

have (⋃(range(S))) –is closed in″ Pow(Q) Q∩(⋃range(S))=(⋃range(S)) using IsClosed˙def by auto
from this(2) compact˙closed[OF assms(2) this(1)] have (⋃range(S))–is compact of cardinal″csucc(Q) –in″Pow(Q)
by auto
moreover
have ⋃(range(S))⊆⋃(range(S)) by auto
ultimately have ∃S∈Pow(range(S)). (⋃(range(S)))⊆⋃S ∧ S≺ csucc(Q) using IsCompactOfCard˙def by auto
then obtain SS where SS˙def:SS⊆range(S) (⋃(range(S)))⊆⋃SS SS≺ csucc(Q) by auto
with ‹S∈bij(M,range(S))› have con:converse(S)∈bij(range(S),M) using bij˙converse˙bij by auto
then have r1:restrict(converse(S),SS)∈bij(SS,converse(S)SS) using restrict˙bij bij˙def SS˙def(1) by auto
then have rr:converse(restrict(converse(S),SS))∈bij(converse(S)SS,SS) using bij˙converse˙bij by auto
{
fix x
assume x∈⋃T
with cover have x∈⋃M by auto
then obtain R where R∈M x∈R by auto
with MT have R∈T x∈R by auto
then have ∃V∈–Ni. i∈Q″. V⊆R ∧ x∈V using point˙open˙base˙neigh base by force
then obtain j where j∈Q Nj⊆R and x˙p:x∈Nj by auto
with ‹R∈M› ‹S:M→Pow(Q)› ‹S∈bij(M,range(S))› have SR∈range(S) ∧ j∈SR using apply˙equality
bij˙def inj˙def by auto
from exI[where P=λt. t∈range(S) ∧ j∈t, OF this] have ∃A∈range(S). j∈A unfolding Bex˙def
by auto
then have j∈(⋃(range(S))) by auto
then have j∈⋃SS using SS˙def(2) by blast
then obtain SR where SR∈SS j∈SR by auto
moreover
have converse(restrict(converse(S),SS))∈surj(converse(S)SS,SS) using rr bij˙def by auto
ultimately obtain RR where converse(restrict(converse(S),SS))RR=SR and p:RR∈converse(S)SS unfolding surj˙def by blast
then have converse(converse(restrict(converse(S),SS)))(converse(restrict(converse(S),SS))RR)=converse(co
by auto
moreover
have converse(restrict(converse(S),SS))∈inj(converse(S)SS,SS) using rr unfolding bij˙def by auto
moreover
ultimately have RR=converse(converse(restrict(converse(S),SS)))SR using left˙inverse[OF ˙ p]
by force

moreover
with r1 have restrict(converse(S),SS)∈SS→converse(S)SS unfolding bij˙def inj˙def by auto
then have relation(restrict(converse(S),SS)) using Pi˙def relation˙def by auto
then have converse(converse(restrict(converse(S),SS)))=restrict(converse(S),SS) using relation˙converse˙converse by auto
ultimately have RR=restrict(converse(S),SS)SR by auto
with ‹SR∈SS› have eq:RR=converse(S)SR unfolding restrict by auto
then have converse(converse(S))RR=converse(converse(S))(converse(S)SR) by auto
moreover
with ‹SR∈SS› have SR∈range(S) using SS˙def(1) by auto
from con left˙inverse[OF ˙ this] have converse(converse(S))(converse(S)SR)=SR unfolding bij˙def
by auto
ultimately have converse(converse(S))RR=SR by auto
then have SRR=SR using relation˙converse˙converse[of S] unfolding relation˙def by auto
moreover
have converse(S):range(S)→M using con bij˙def inj˙def by auto
with ‹SR∈range(S)› have converse(S)SR∈M using apply˙funtype
by auto
with eq have RR∈M by auto
ultimately have SR=–i∈Q. Ni⊆RR″ using ‹S:M→Pow(Q)› apply˙equality by auto
then have Nj⊆RR using ‹j∈SR› by auto
with x˙p have x∈RR by auto
with p have x∈⋃(converse(S)SS) by auto
}
then have ⋃T⊆⋃(converse(S)SS) by blast
moreover
{
from con have converse(S)SS=–converse(S)R. R∈SS″ using image˙function[of converse(S) SS]
SS˙def(1) unfolding range˙def bij˙def inj˙def Pi˙def by auto
have –converse(S)R. R∈SS″⊆–converse(S)R. R∈range(S)″ using SS˙def(1) by auto
moreover
have converse(S):range(S)→M using con unfolding bij˙def inj˙def by auto
then have –converse(S)R. R∈range(S)″⊆M using apply˙funtype by force
ultimately
have (converse(S)SS)⊆M by auto
}
then have converse(S)SS∈Pow(M) by auto
moreover
with rr have converse(S)SS≈SS using eqpoll˙def by auto
then have converse(S)SS≺csucc(Q) using SS˙def(3) eq˙lesspoll˙trans by auto
ultimately
have ∃N∈Pow(M). ⋃T⊆⋃N ∧ N≺csucc(Q) by auto

```
    }
   then have ∀M∈Pow(T). ⋃T⊆⋃M ⟶ (∃N∈Pow(M). ⋃T⊆⋃N ∧ N≺csucc(Q))
by auto
    ultimately have (⋃T)–is compact of cardinal″csucc(Q)–in″T unfolding IsCompactOfCard˙def
      by auto
  }
  then show (T –is a topology″) ∧ (T –is of second type of cardinal″csucc(Q)) ⟶
((⋃T)–is compact of cardinal″csucc(Q) –in″T)
  by auto
qed

theorem Q˙disc˙is˙second˙card˙csuccQ:
  assumes InfCard(Q)
  shows Pow(Q)–is of second type of cardinal″csucc(Q)
proof-
  {
    fix A
    assume AS:A∈Pow(Q)
    have A=⋃—i″. i∈A″ by auto
    with AS have ∃T∈Pow(—i″. i∈Q″). A=⋃T by auto
    then have A∈–⋃U. U∈Pow(—i″. i∈Q″)″ by auto
  }
  moreover
  {
    fix A
    assume AS:A∈–⋃U. U∈Pow(—i″. i∈Q″)″
    then have A∈Pow(Q) by auto
  }
  ultimately
  have base:—x″. x∈Q″ –is a base for″ Pow(Q) unfolding IsAbaseFor˙def by blast
  let f=–⟨i,–i″⟩. i∈Q″
  have f∈Q→—x″. x∈Q″ unfolding Pi˙def function˙def by auto
  then have f∈inj(Q,—x″. x∈Q″) unfolding inj˙def using apply˙equality by auto
  moreover
   from ‹f∈Q→—x″. x∈Q″› have f∈surj(Q,—x″. x∈Q″) unfolding surj˙def using
apply˙equality
    by auto
  ultimately have f∈bij(Q,—x″. x∈Q″) unfolding bij˙def by auto
  then have Q≈—x″. x∈Q″ using eqpoll˙def by auto
  then have —x″. x∈Q″≈Q using eqpoll˙sym by auto
  then have —x″. x∈Q″≲Q using eqpoll˙imp˙lepoll by auto
  then have —x″. x∈Q″≺csucc(Q) using Card˙less˙csucc˙eq˙le assms InfCard˙is˙Card
by auto
  with base show thesis using IsSecondOfCard˙def by auto
qed
```

This previous results give us another equivalence of the axiom of Q choice that is apparently weaker (easier to check) to the previous one.

theorem Q˙disc˙comp˙csuccQ˙eq˙Q˙choice˙csuccQ:
assumes InfCard(Q)
shows (Q–is compact of cardinal″csucc(Q)–in″(Pow(Q))) ⟷ (–the axiom of″Q–choice holds for subsets″(Pow(Q)))
proof
assume Q–is compact of cardinal″csucc(Q) –in″Pow(Q)
with assms show –the axiom of″Q–choice holds for subsets″(Pow(Q)) using Q˙choice˙Pow˙eq˙secon˙imp˙comp Q˙csuccQ˙comp˙eq˙Q˙choice˙Pow
by auto
next
assume –the axiom of″Q–choice holds for subsets″(Pow(Q))
with assms show Q–is compact of cardinal″csucc(Q)–in″(Pow(Q)) using Q˙disc˙is˙second˙card˙csuccQ Q˙choice˙Pow˙eq˙secon˙imp˙comp Pow˙is˙top[of Q]
by force
qed

end

60 Topology 5

theory Topology˙ZF˙5 imports Topology˙ZF˙properties Topology˙ZF˙examples˙1 Topology˙ZF˙4
begin

60.1 Some results for separation axioms

First we will give a global characterization of T_1-spaces; which is interesting because it involves the cardinal $\mathbb{N}$.

lemma (in topology0) T1˙cocardinal˙coarser:
shows (T –is T_1″) ⟷ (CoFinite (⋃T))⊆T
proof
{
assume AS:T –is T_1″
{
fix x assume p:x∈⋃T
{
fix y assume y∈(⋃T)-–x″
with AS p obtain U where U∈T y∈U x∉U using isT1˙def by blast
then have U∈T y∈U U⊆(⋃T)-–x″ by auto
then have ∃U∈T. y∈U ∧ U⊆(⋃T)-–x″ by auto
}
then have ∀y∈(⋃T)-–x″. ∃U∈T. y∈U ∧ U⊆(⋃T)-–x″ by auto
then have ⋃T-–x″∈T using open˙neigh˙open by auto
with p have –x″ –is closed in″T using IsClosed˙def by auto
}
then have pointCl:∀x∈⋃T. –x″ –is closed in″ T by auto
{

fix A
assume AS2:A∈FinPow(⋃T)
let p=–⟨x,–x″⟩. x∈A″
have p∈A→—x″. x∈A″ using Pi˙def unfolding function˙def by auto
then have p:bij(A,—x″. x∈A″) unfolding bij˙def inj˙def surj˙def using apply˙equality
by auto
then have A≈—x″. x∈A″ unfolding eqpoll˙def by auto
with AS2 have Finite(—x″. x∈A″) unfolding FinPow˙def using eqpoll˙imp˙Finite˙iff by auto
then have —x″. x∈A″∈FinPow(–D ∈ Pow(⋃T) . D –is closed in″ T″) using AS2 pointCl unfolding FinPow˙def
by (safe, blast+)
then have (⋃—x″. x∈A″) –is closed in″ T using fin˙union˙cl˙is˙cl by auto
moreover
have ⋃—x″. x∈A″=A by auto
ultimately have A –is closed in″ T by simp
}
then have reg:∀A∈FinPow(⋃T). A –is closed in″ T by auto
{
fix U
assume AS2:U ∈ CoCardinal(⋃T,nat)
then have U∈Pow(⋃T) U=0 ∨ ((⋃T)-U)≺nat using CoCardinal˙def by auto
then have U∈Pow(⋃T) U=0 ∨ Finite(⋃T-U) using lesspoll˙nat˙is˙Finite by auto
then have U∈Pow(⋃T) U∈T∨(⋃T-U) –is closed in″ T using empty˙open topSpaceAssum
reg unfolding FinPow˙def by auto
then have U∈Pow(⋃T) U∈T∨(⋃T-(⋃T-U))∈T using IsClosed˙def by auto
moreover
then have (⋃T-(⋃T-U))=U by blast
ultimately have U∈T by auto
}
then show (CoFinite (⋃T))⊆T using Cofinite˙def by auto
}
{
assume (CoFinite (⋃T))⊆T
then have AS:CoCardinal(⋃T,nat) ⊆ T using Cofinite˙def by auto
{
fix x y
assume AS2:x∈⋃T y∈⋃Tx≠y
have Finite(–y″) by auto
then obtain n where –y″≈n n∈nat using Finite˙def by auto
then have –y″≺nat using n˙lesspoll˙nat eq˙lesspoll˙trans by auto
then have –y″ –is closed in″ CoCardinal(⋃T,nat) using closed˙sets˙cocardinal AS2(2) by auto
then have (⋃T)-–y″∈CoCardinal(⋃T,nat) using union˙cocardinal IsClosed˙def by auto
with AS have (⋃T)-–y″∈T by auto

```
    moreover
    with AS2(1,3) have x∈((⋃T)-–y″) ∧ y∉((⋃T)-–y″) by auto
    ultimately have ∃V∈T. x∈V∧y∉V by(safe,auto)
  }
  then show T –is T1″ using isT1˙def by auto
 }
qed
```

In the previous proof, it is obvious that we don't need to check if ever cofinite set is open. It is enough to check if every singleton is closed.

```
corollary(in topology0) T1˙iff˙singleton˙closed:
 shows (T –is T1″) ⟷ (∀x∈⋃T. –x″–is closed in″T)
proof
 assume AS:T –is T1″
 {
  fix x assume p:x∈⋃T
  {
   fix y assume y∈(⋃T)-–x″
   with AS p obtain U where U∈T y∈U x∉U using isT1˙def by blast
   then have U∈T y∈U U⊆(⋃T)-–x″ by auto
   then have ∃U∈T. y∈U ∧ U⊆(⋃T)-–x″ by auto
  }
  then have ∀y∈(⋃T)-–x″. ∃U∈T. y∈U ∧ U⊆(⋃T)-–x″ by auto
  then have ⋃T-–x″∈T using open˙neigh˙open by auto
  with p have –x″ –is closed in″T using IsClosed˙def by auto
 }
 then show pointCl:∀x∈⋃T. –x″ –is closed in″ T by auto
next
 assume pointCl:∀x∈⋃T. –x″ –is closed in″ T
 {
  fix A
  assume AS2:A∈FinPow(⋃T)
  let p=–⟨x,–x″⟩. x∈A″
  have p∈A→—x″. x∈A″ using Pi˙def unfolding function˙def by auto
   then have p:bij(A,—x″. x∈A″) unfolding bij˙def inj˙def surj˙def using ap-
ply˙equality
   by auto
  then have A≈—x″. x∈A″ unfolding eqpoll˙def by auto
 with AS2 have Finite(—x″. x∈A″) unfolding FinPow˙def using eqpoll˙imp˙Finite˙iff
by auto
 then have —x″. x∈A″∈FinPow(–D ∈ Pow(⋃T) . D –is closed in″ T″) using AS2
pointCl unfolding FinPow˙def
  by (safe, blast+)
  then have (⋃—x″. x∈A″) –is closed in″ T using fin˙union˙cl˙is˙cl by auto
  moreover
  have ⋃—x″. x∈A″=A by auto
  ultimately have A –is closed in″ T by simp
 }
 then have reg:∀A∈FinPow(⋃T). A –is closed in″ T by auto
```

```
  {
    fix U
    assume AS2:U∈CoCardinal(⋃T,nat)
    then have U∈Pow(⋃T) U=0 ∨ ((⋃T)-U)≺nat using CoCardinal˙def by auto
   then have U∈Pow(⋃T) U=0 ∨ Finite(⋃T-U) using lesspoll˙nat˙is˙Finite by auto
      then have U∈Pow(⋃T) U∈T∨(⋃T-U) –is closed in″ T using empty˙open
topSpaceAssum
      reg unfolding FinPow˙def by auto
    then have U∈Pow(⋃T) U∈T∨(⋃T-(⋃T-U))∈T using IsClosed˙def by auto
    moreover
    then have (⋃T-(⋃T-U))=U by blast
    ultimately have U∈T by auto
  }
  then have (CoFinite (⋃T))⊆T using Cofinite˙def by auto
  then show T –is T1″ using T1˙cocardinal˙coarser by auto
qed
```

Secondly, let's show that the CoCardinal X Q topologies for different sets Q are all ordered as the partial order of sets. (The order is linear when considering only cardinals)

```
lemma order˙cocardinal˙top:
  fixes X
  assumes Q1≲Q2
  shows CoCardinal(X,Q1) ⊆ CoCardinal(X,Q2)
proof
  fix x
  assume x ∈ CoCardinal(X,Q1)
  then have x∈Pow(X) x=0∨(X-x)≺Q1 using CoCardinal˙def by auto
  with assms have x∈Pow(X) x=0∨(X-x)≺Q2 using lesspoll˙trans2 by auto
  then show x∈CoCardinal(X,Q2) using CoCardinal˙def by auto
qed

corollary cocardinal˙is˙T1:
  fixes X K
  assumes InfCard(K)
  shows CoCardinal(X,K) –is T1″
proof-
  have nat≤K using InfCard˙def assms by auto
  then have nat⊆K using le˙imp˙subset by auto
  then have nat≲K K≠0using subset˙imp˙lepoll by auto
  then have CoCardinal(X,nat) ⊆ CoCardinal(X,K) ⋃CoCardinal(X,K)=X using
order˙cocardinal˙top
    union˙cocardinal by auto
   then show thesis using topology0.T1˙cocardinal˙coarser topology0˙CoCardinal
assms Cofinite˙def
    by auto
qed
```

In T_2-spaces, filters and nets have at most one limit point.

```
lemma (in topology0) T2˙imp˙unique˙limit˙filter:
  assumes T –is T₂″ 𝔉 –is a filter on″⋃T 𝔉 →_F x 𝔉 →_F y
  shows x=y
proof-
  {
    assume x≠y
    from assms(3,4) have x∈⋃T y∈⋃T using FilterConverges˙def assms(2)
      by auto
    with ‹x≠y› have ∃U∈T. ∃V∈T. x∈U ∧ y∈V ∧ U∩V=0 using assms(1) isT2˙def
by auto
    then obtain U V where x∈U y∈V U∩V=0 U∈T V∈T by auto
   then have U∈–A∈Pow(⋃T). x∈Interior(A,T)″ V∈–A∈Pow(⋃T). y∈Interior(A,T)″
using Top˙2˙L3 by auto
    then have U∈𝔉 V∈𝔉 using FilterConverges˙def assms(2) assms(3,4)
      by auto
    then have U∩V∈𝔉 using IsFilter˙def assms(2) by auto
    with ‹U∩V=0› have 0∈𝔉 by auto
    then have False using IsFilter˙def assms(2) by auto
  }
  then show thesis by auto
qed

lemma (in topology0) T2˙imp˙unique˙limit˙net:
  assumes T –is T₂″ N –is a net on″⋃T N →_N x N →_N y
  shows x=y
proof-
   have (Filter N..(⋃T)) –is a filter on″ (⋃T) (Filter N..(⋃T)) →_F x (Filter
N..(⋃T)) →_F y
    using filter˙of˙net˙is˙filter(1) net˙conver˙filter˙of˙net˙conver assms(2)
    assms(3,4) by auto
  with assms(1) show thesis using T2˙imp˙unique˙limit˙filter by auto
qed
```

In fact, T_2-spaces are characterized by this property. For this proof we build a filter containing the union of two filters.

```
lemma (in topology0) unique˙limit˙filter˙imp˙T2:
  assumes ∀x∈⋃T. ∀y∈⋃T. ∀𝔉. ((𝔉 –is a filter on″⋃T) ∧ (𝔉 →_F x) ∧ (𝔉 →_F y))
⟶ x=y
  shows T –is T₂″
proof-
  {
    fix x y
    assume x∈⋃T y∈⋃T x≠y
    {
      assume ∀U∈T. ∀V∈T. (x∈U ∧ y∈V) ⟶ U∩V≠0
      let Ux=–A∈Pow(⋃T). x∈int(A)″
      let Uy=–A∈Pow(⋃T). y∈int(A)″
      let FF=Ux ∪ Uy ∪ –A∩B. ⟨A,B⟩∈Ux × Uy″
      have sat:FF –satisfies the filter base condition″
```

```
    proof-
      {
        fix A B
        assume A∈FF B∈FF
        {
          assume A∈Ux
          {
            assume B∈Ux
           with ‹x∈⋃T› ‹A∈Ux› have A∩B∈Ux using neigh˙filter(1) IsFilter˙def by
auto
            then have A∩B∈FF by auto
          }
          moreover
          {
            assume B∈Uy
            with ‹A∈Ux› have A∩B∈FF by auto
          }
          moreover
          {
            assume B∈–A∩B. ⟨A,B⟩∈Ux × Uy″
            then obtain AA BB where B=AA∩BB AA∈Ux BB∈Uy by auto
                 with ‹x∈⋃T› ‹A∈Ux› have A∩B=(A∩AA)∩BB A∩AA∈Ux using
neigh˙filter(1) IsFilter˙def by auto
            with ‹BB∈Uy› have A∩B∈–A∩B. ⟨A,B⟩∈Ux × Uy″ by auto
            then have A∩B∈FF by auto
          }
          ultimately have A∩B∈FF using ‹B∈FF› by auto
        }
        moreover
        {
          assume A∈Uy
          {
            assume B∈Uy
           with ‹y∈⋃T› ‹A∈Uy› have A∩B∈Uy using neigh˙filter(1) IsFilter˙def by
auto
            then have A∩B∈FF by auto
          }
          moreover
          {
            assume B∈Ux
            with ‹A∈Uy› have B∩A∈FF by auto
            moreover have A∩B=B∩A by auto
            ultimately have A∩B∈FF by auto
          }
          moreover
          {
            assume B∈–A∩B. ⟨A,B⟩∈Ux × Uy″
            then obtain AA BB where B=AA∩BB AA∈Ux BB∈Uy by auto
                 with ‹y∈⋃T› ‹A∈Uy› have A∩B=AA∩(A∩BB) A∩BB∈Uy using
```

```
neigh˙filter(1) IsFilter˙def by auto
            with ‹AA∈Ux› have A∩B∈–A∩B. ⟨A,B⟩∈Ux × Uy″ by auto
            then have A∩B∈FF by auto
          }
          ultimately have A∩B∈FF using ‹B∈FF› by auto
        }
        moreover
        {
          assume A∈–A∩B. ⟨A,B⟩∈Ux × Uy″
          then obtain AA BB where A=AA∩BB AA∈Ux BB∈Uy by auto
          {
            assume B∈Uy
            with ‹BB∈Uy› ‹y∈⋃T› have B∩BB∈Uy using neigh˙filter(1) IsFilter˙def
by auto
            moreover from ‹A=AA∩BB› have A∩B=AA∩(B∩BB) by auto
            ultimately have A∩B∈FF using ‹AA∈Ux› ‹B∩BB∈Uy› by auto
          }
          moreover
          {
            assume B∈Ux
            with ‹AA∈Ux› ‹x∈⋃T› have B∩AA∈Ux using neigh˙filter(1) IsFilter˙def
by auto
            moreover from ‹A=AA∩BB› have A∩B=(B∩AA)∩BB by auto
            ultimately have A∩B∈FF using ‹B∩AA∈Ux› ‹BB∈Uy› by auto
          }
          moreover
          {
            assume B∈–A∩B. ⟨A,B⟩∈Ux × Uy″
            then obtain AA2 BB2 where B=AA2∩BB2 AA2∈Ux BB2∈Uy by auto
             from ‹B=AA2∩BB2› ‹A=AA∩BB› have A∩B=(AA∩AA2)∩(BB∩BB2)
by auto
            moreover
           from ‹AA∈Ux›‹AA2∈Ux›‹x∈⋃T› have AA∩AA2∈Ux using neigh˙filter(1)
IsFilter˙def by auto
            moreover
           from ‹BB∈Uy›‹BB2∈Uy›‹y∈⋃T› have BB∩BB2∈Uy using neigh˙filter(1)
IsFilter˙def by auto
            ultimately have A∩B∈FF by auto
          }
          ultimately have A∩B∈FF using ‹B∈FF› by auto
        }
        ultimately have A∩B∈FF using ‹A∈FF› by auto
        then have ∃D∈FF. D⊆A∩B unfolding Bex˙def by auto
      }
      then have ∀A∈FF. ∀B∈FF. ∃D∈FF. D⊆A∩B by force
      moreover
      have ⋃T∈Ux using ‹x∈⋃T› neigh˙filter(1) IsFilter˙def by auto
      then have FF≠0 by auto
      moreover
```

```
      {
        assume 0∈FF
        moreover
        have 0∉Ux using ‹x∈⋃T› neigh˙filter(1) IsFilter˙def by auto
        moreover
        have 0∉Uy using ‹y∈⋃T› neigh˙filter(1) IsFilter˙def by auto
        ultimately have 0∈–A∩B. ⟨A,B⟩∈Ux × Uy″ by auto
        then obtain A B where 0=A∩B A∈UxB∈Uy by auto
        then have x∈int(A)y∈int(B) by auto
        moreover
        with ‹0=A∩B› have int(A)∩int(B)=0 using Top˙2˙L1 by auto
        moreover
        have int(A)∈Tint(B)∈T using Top˙2˙L2 by auto
          ultimately have False using ‹∀U∈T. ∀V∈T. x∈U∧y∈V ⟶ U∩V≠0› by
auto
      }
      then have 0∉FF by auto
      ultimately show thesis using SatisfiesFilterBase˙def by auto
    qed
    moreover
    have FF⊆Pow(⋃T) by auto
      ultimately have bas:FF –is a base filter″ –A∈Pow(⋃T). ∃D∈FF. D⊆A″
⋃–A∈Pow(⋃T). ∃D∈FF. D⊆A″=⋃T
      using base˙unique˙filter˙set2[of FF] by auto
   then have fil:–A∈Pow(⋃T). ∃D∈FF. D⊆A″ –is a filter on″ ⋃T using basic˙filter
sat by auto
    have ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈FF. D⊆U) by auto
   then have –A∈Pow(⋃T). ∃D∈FF. D⊆A″ →F x using convergence˙filter˙base2[OF
fil bas(1) ˙ ‹x∈⋃T›] by auto
    moreover
    then have ∀U∈Pow(⋃T). y∈int(U) ⟶ (∃D∈FF. D⊆U) by auto
   then have –A∈Pow(⋃T). ∃D∈FF. D⊆A″ →F y using convergence˙filter˙base2[OF
fil bas(1) ˙ ‹y∈⋃T›] by auto
    ultimately have x=y using assms fil ‹x∈⋃T›‹y∈⋃T› by blast
    with ‹x≠y› have False by auto
   }
   then have ∃U∈T. ∃V∈T. x∈U ∧ y∈V ∧ U∩V=0 by blast
  }
  then show thesis using isT2˙def by auto
qed

lemma (in topology0) unique˙limit˙net˙imp˙T2:
  assumes ∀x∈⋃T. ∀y∈⋃T. ∀N. ((N –is a net on″⋃T) ∧ (N →N x) ∧ (N →N
y)) ⟶ x=y
  shows T –is T2″
proof-
  {
   fix x y 𝔉
   assume x∈⋃T y∈⋃T𝔉 –is a filter on″⋃T𝔉 →F x𝔉 →F y
```

then have (Net(𝔉)) –is a net on″ ⋃T(Net 𝔉) $\rightarrow_N$ x(Net 𝔉) $\rightarrow_N$ y
using filter˙conver˙net˙of˙filter˙conver net˙of˙filter˙is˙net by auto
with ‹x∈⋃T› ‹y∈⋃T› have x=y using assms by blast
}
then have ∀x∈⋃T. ∀y∈⋃T. ∀𝔉. ((𝔉 –is a filter on″⋃T) ∧ (𝔉 $\rightarrow_F$ x) ∧ (𝔉 $\rightarrow_F$ y)) ⟶ x=y by auto
then show thesis using unique˙limit˙filter˙imp˙T2 by auto
qed

This results make easy to check if a space is T_2.

The topology which comes from a filter as in 𝔉 –is a filter on″ ⋃𝔉 ⟹ (𝔉 ∪ –0″) –is a topology″ is not T_2 generally. We will see in this file later on, that the exceptions are a consequence of the spectrum.

corollary filter˙T2˙imp˙card1:
assumes (𝔉∪–0″) –is T_2″ 𝔉 –is a filter on″ ⋃𝔉 x∈⋃𝔉
shows ⋃𝔉=–x″
proof-
{
fix y assume y∈⋃𝔉
then have 𝔉 $\rightarrow_F$ y –in″ (𝔉∪–0″) using lim˙filter˙top˙of˙filter assms(2) by auto
moreover
have 𝔉 $\rightarrow_F$ x –in″ (𝔉∪–0″) using lim˙filter˙top˙of˙filter assms(2,3) by auto
moreover
have ⋃𝔉=⋃(𝔉∪–0″) by auto
ultimately
have y=x using topology0.T2˙imp˙unique˙limit˙filter[OF topology0˙filter[OF assms(2)] assms(1)] assms(2)
by auto
}
then have ⋃𝔉⊆–x″ by auto
with assms(3) show thesis by auto
qed

There are more separation axioms that just T_0, T_1 or T_2

definition
IsRegular (˙–is regular″ 90)
where T–is regular″ ≡ ∀A. A–is closed in″T ⟶ (∀x∈⋃T-A. ∃U∈T. ∃V∈T. A⊆U∧x∈V∧U∩V=0)

definition
isT3 (˙–is T_3″ 90)
where T–is T_3″ ≡ (T–is T_1″) ∧ (T–is regular″)

definition
IsNormal (˙–is normal″ 90)
where T–is normal″ ≡ ∀A. A–is closed in″T ⟶ (∀B. B–is closed in″T ∧ A∩B=0 ⟶
(∃U∈T. ∃V∈T. A⊆U∧B⊆V∧U∩V=0))

```
definition
  isT4 (˙–is T4″ 90)
  where T–is T4″ ≡ (T–is T1″) ∧ (T–is normal″)

lemma (in topology0) T4˙is˙T3:
  assumes T–is T4″ shows T–is T3″
proof-
  from assms have nor:T–is normal″ using isT4˙def by auto
  from assms have T–is T1″ using isT4˙def by auto
  then have Cofinite (⋃T)⊆T using T1˙cocardinal˙coarser by auto
  {
    fix A
    assume AS:A–is closed in″T
    {
      fix x
      assume x∈⋃T-A
      have Finite(–x″) by auto
      then obtain n where –x″≈n n∈nat unfolding Finite˙def by auto
      then have –x″≲n n∈nat using eqpoll˙imp˙lepoll by auto
      then have –x″≺nat using n˙lesspoll˙nat lesspoll˙trans1 by auto
      with ‹x∈⋃T-A› have –x″ –is closed in″ (Cofinite (⋃T)) using Cofinite˙def
        closed˙sets˙cocardinal by auto
    then have ⋃T-–x″∈Cofinite(⋃T) unfolding IsClosed˙def using union˙cocardinal
Cofinite˙def
        by auto
      with ‹Cofinite (⋃T)⊆T› have ⋃T-–x″∈T by auto
      with ‹x∈⋃T-A› have –x″–is closed in″T A∩–x″=0 using IsClosed˙def by auto
    with nor AS have ∃U∈T. ∃V∈T. A⊆U∧–x″⊆V∧U∩V=0 unfolding IsNormal˙def
by blast
      then have ∃U∈T. ∃V∈T. A⊆U∧ x∈V∧U∩V=0 by auto
    }
    then have ∀x∈⋃T-A. ∃U∈T. ∃V∈T. A⊆U∧ x∈V∧U∩V=0 by auto
  }
  then have T–is regular″ using IsRegular˙def by blast
  with ‹T–is T1″› show thesis using isT3˙def by auto
qed

lemma (in topology0) T3˙is˙T2:
  assumes T–is T3″ shows T–is T2″
proof-
  from assms have T–is regular″ using isT3˙def by auto
  from assms have T–is T1″ using isT3˙def by auto
  then have Cofinite (⋃T)⊆T using T1˙cocardinal˙coarser by auto
  {
    fix x y
    assume x∈⋃Ty∈⋃Tx≠y
    have Finite(–x″) by auto
    then obtain n where –x″≈n n∈nat unfolding Finite˙def by auto
```

then have –x″≲n n∈nat using eqpoll˙imp˙lepoll by auto
then have –x″≺nat using n˙lesspoll˙nat lesspoll˙trans1 by auto
with ‹x∈⋃T› have –x″ –is closed in″ (Cofinite (⋃T)) using Cofinite˙def closed˙sets˙cocardinal by auto
then have ⋃T-–x″∈Cofinite(⋃T) unfolding IsClosed˙def using union˙cocardinal Cofinite˙def by auto
with ‹Cofinite (⋃T)⊆T› have ⋃T-–x″∈T by auto
with ‹x∈⋃T›‹y∈⋃T›‹x≠y› have –x″–is closed in″T y∈⋃T-–x″ using IsClosed˙def by auto
with ‹T–is regular″› have ∃U∈T. ∃V∈T. –x″⊆U∧y∈V∧U∩V=0 unfolding IsRegular˙def by force
then have ∃U∈T. ∃V∈T. x∈U∧y∈V∧U∩V=0 by auto
}
then show thesis using isT2˙def by auto
qed

Regularity can be rewritten in terms of existence of certain neighboorhoods.

lemma (in topology0) regular˙imp˙exist˙clos˙neig:
assumes T–is regular″ and U∈T and x∈U
shows ∃V∈T. x∈V ∧ cl(V)⊆U
proof-
from assms(2) have (⋃T-U)–is closed in″T using Top˙3˙L9 by auto moreover
from assms(2,3) have x∈⋃T by auto moreover
note assms(1,3) ultimately obtain A B where A∈T and B∈T and A∩B=0 and (⋃T-U)⊆A and x∈B
unfolding IsRegular˙def by blast
from ‹A∩B=0› ‹B∈T› have B⊆⋃T-A by auto
with ‹A∈T› have cl(B)⊆⋃T-A using Top˙3˙L9 Top˙3˙L13 by auto
moreover from ‹(⋃T-U)⊆A› assms(3) have ⋃T-A⊆U by auto
moreover note ‹x∈B› ‹B∈T›
ultimately have B∈T ∧ x∈B ∧ cl(B)⊆U by auto
then show thesis by auto
qed

lemma (in topology0) exist˙clos˙neig˙imp˙regular:
assumes ∀x∈⋃T. ∀U∈T. x∈U ⟶ (∃V∈T. x∈V∧ cl(V)⊆U)
shows T–is regular″
proof-
{
fix F
assume F–is closed in″T
{
fix x assume x∈⋃T-F
with ‹F–is closed in″T› have x∈⋃T ⋃T-F∈T F⊆⋃T unfolding IsClosed˙def by auto
with assms ‹x∈⋃T-F› have ∃V∈T. x∈V ∧ cl(V)⊆⋃T-F by auto
then obtain V where V∈T x∈V cl(V)⊆⋃T-F by auto
from ‹cl(V)⊆⋃T-F› ‹F⊆⋃T› have F⊆⋃T-cl(V) by auto

moreover from ‹V∈T› have ⋃T-(⋃T-V)=V by auto
then have cl(V)=⋃T-int(⋃T-V) using Top˙3˙L11(2)[of ⋃T-V] by auto
ultimately have F⊆int(⋃T-V) by auto moreover
have int(⋃T-V)⊆⋃T-V using Top˙2˙L1 by auto
then have V∩(int(⋃T-V))=0 by auto moreover
note ‹x∈V›‹V∈T› ultimately
have V∈T int(⋃T-V)∈T F⊆int(⋃T-V) ∧ x∈V ∧ (int(⋃T-V))∩V=0 using Top˙2˙L2
by auto
then have ∃U∈T. ∃V∈T. F⊆U ∧ x∈V ∧ U∩V=0 by auto
}
then have ∀x∈⋃T-F. ∃U∈T. ∃V∈T. F⊆U ∧ x∈V ∧ U∩V=0 by auto
}
then show thesis using IsRegular˙def by blast
qed

lemma (in topology0) regular˙eq:
shows T–is regular″ ⟷ (∀x∈⋃T. ∀U∈T. x∈U ⟶ (∃V∈T. x∈V∧ cl(V)⊆U))
using regular˙imp˙exist˙clos˙neig exist˙clos˙neig˙imp˙regular by force

A Hausdorff space separates compact spaces from points.

theorem (in topology0) T2˙compact˙point:
assumes T–is T_2″ A–is compact in″T x∈⋃T x∉A
shows ∃U∈T. ∃V∈T. A⊆U ∧ x∈V ∧ U∩V=0
proof-
{
assume A=0
then have A⊆0∧x∈⋃T∧(0∩(⋃T)=0) using assms(3) by auto
then have thesis using empty˙open topSpaceAssum unfolding IsATopology˙def by auto
}
moreover
{
assume noEmpty:A≠0
let U=–⟨U,V⟩∈T×T. x∈U∧U∩V=0″
{
fix y assume y∈A
with ‹x∉A› assms(4) have x≠y by auto
moreover from ‹y∈A› have x∈⋃Ty∈⋃T using assms(2,3) unfolding IsCompact˙def by auto
ultimately obtain U V where U∈TV∈TU∩V=0x∈Uy∈V using assms(1) unfolding isT2˙def by blast
then have ∃⟨U,V⟩∈U. y∈V by auto
}
then have ∀y∈A. ∃⟨U,V⟩∈U. y∈V by auto
then have A⊆⋃–snd(B). B∈U″ by auto
moreover have –snd(B). B∈U″∈Pow(T) by auto
ultimately have ∃N∈FinPow(–snd(B). B∈U″). A⊆⋃N using assms(2) unfolding IsCompact˙def by auto

then obtain N where ss:N∈FinPow(–snd(B). B∈U″) A⊆⋃N by auto
with ‹–snd(B). B∈U″∈Pow(T)› have A⊆⋃N N∈Pow(T) unfolding FinPow˙def by auto
then have NN:A⊆⋃N ⋃N∈T using topSpaceAssum unfolding IsATopology˙def by auto
from ss have Finite(N)N⊆–snd(B). B∈U″ unfolding FinPow˙def by auto
then obtain n where n∈nat N≈n unfolding Finite˙def by auto
then have N≲n using eqpoll˙imp˙lepoll by auto
from noEmpty ‹A⊆⋃N› have NnoEmpty:N≠0 by auto
let QQ=–⟨n,–fst(B). B∈–A∈U. snd(A)=n″″⟩. n∈N″
have QQPi:QQ:N→—fst(B). B∈–A∈U. snd(A)=n″″. n∈N″ unfolding Pi˙def function˙def domain˙def by auto
{
fix n assume n∈N
with ‹N⊆–snd(B). B∈U″› obtain B where n=snd(B) B∈U by auto
then have fst(B)∈–fst(B). B∈–A∈U. snd(A)=n″″ by auto
then have –fst(B). B∈–A∈U. snd(A)=n″″≠0 by auto moreover
from ‹n∈N› have ⟨n,–fst(B). B∈–A∈U. snd(A)=n″″⟩∈QQ by auto
with QQPi have QQn=–fst(B). B∈–A∈U. snd(A)=n″″ using apply˙equality by auto
ultimately have QQn≠0 by auto
}
then have ∀n∈N. QQn≠0 by auto
with ‹n∈nat› ‹N≲n› have ∃f. f∈Pi(N,λt. QQt) ∧ (∀t∈N. ft∈QQt) using finite˙choice unfolding AxiomCardinalChoiceGen˙def
by auto
then obtain f where fPI:f∈Pi(N,λt. QQt) (∀t∈N. ft∈QQt) by auto
from fPI(1) NnoEmpty have range(f)≠0 unfolding Pi˙def range˙def domain˙def converse˙def by (safe,blast)
{
fix t assume t∈N
then have ft∈QQt using fPI(2) by auto
with ‹t∈N› have ft∈⋃(QQN) QQt⊆⋃(QQN) using func˙imagedef QQPi by auto
}
then have reg:∀t∈N. ft∈⋃(QQN) ∀t∈N. QQt⊆⋃(QQN) by auto
{
fix tt assume tt∈f
with fPI(1) have tt∈Sigma(N, ()(QQ)) unfolding Pi˙def by auto
then have tt∈(⋃xa∈N. ⋃y∈QQxa. –⟨xa,y⟩″) unfolding Sigma˙def by auto
then obtain xa y where xa∈N y∈QQxa tt=⟨xa,y⟩ by auto
with reg(2) have y∈⋃(QQN) by blast
with ‹tt=⟨xa,y⟩› ‹xa∈N› have tt∈(⋃xa∈N. ⋃y∈⋃(QQN). –⟨xa,y⟩″) by auto
then have tt∈N×(⋃(QQN)) unfolding Sigma˙def by auto
}
then have ffun:f:N→⋃(QQN) using fPI(1) unfolding Pi˙def by auto
then have f∈surj(N,range(f)) using fun˙is˙surj by auto
with ‹N≲n› ‹n∈nat› have range(f)≲N using surj˙fun˙inv˙2 nat˙into˙Ord by auto
with ‹N≲n› have range(f)≲n using lepoll˙trans by blast

with ‹n∈nat› have Finite(range(f)) using n˙lesspoll˙nat lesspoll˙nat˙is˙Finite lesspoll˙trans1 by auto
moreover from ffun have rr:range(f)⊆⋃(QQN) unfolding Pi˙def by auto
then have range(f)⊆T by auto
ultimately have range(f)∈FinPow(T) unfolding FinPow˙def by auto
then have ⋂range(f)∈T using fin˙inter˙open˙open ‹range(f)≠0› by auto moreover
{
fix S assume S∈range(f)
with rr have S∈⋃(QQN) by blast
then have ∃B∈(QQN). S ∈ B using Union˙iff by auto
then obtain B where B∈(QQN) S∈B by auto
then have ∃rr∈N. ⟨rr,B⟩∈QQ unfolding image˙def by auto
then have ∃rr∈N. B=–fst(B). B∈–A∈U. snd(A)=rr˝˝ by auto
with ‹S∈B› obtain rr where ⟨S,rr⟩∈U by auto
then have x∈S by auto
}
then have x∈⋂range(f) using ‹range(f)≠0› by auto moreover
{
fix y assume y∈(⋃N)∩(⋂range(f))
then have reg:(∀S∈range(f). y∈S)∧(∃t∈N. y∈t) by auto
then obtain t where t∈N y∈t by auto
then have ⟨t, –fst(B). B∈–A∈U. snd(A)=t˝˝⟩∈QQ by auto
then have ft∈range(f) using apply˙rangeI ffun by auto
with reg have yft:y∈ft by auto
with ‹t∈N› fPI(2) have ft∈QQt by auto
with ‹t∈N› have ft∈–fst(B). B∈–A∈U. snd(A)=t˝˝ using apply˙equality QQPi by auto
then have ⟨ft,t⟩∈U by auto
then have ft∩t=0 by auto
with ‹y∈t› yft have False by auto
}
then have (⋃N)∩(⋂range(f))=0 by blast moreover
note NN
ultimately have thesis by auto
}
ultimately show thesis by auto
qed

A Hausdorff space separates compact spaces from other compact spaces.

theorem (in topology0) T2˙compact˙compact:
assumes T–is T_2˝ A–is compact in˝T B–is compact in˝T A∩B=0
shows ∃U∈T. ∃V∈T. A⊆U ∧ B⊆V ∧ U∩V=0
proof-
{
assume B=0
then have A⊆⋃T∧B⊆0∧((⋃T)∩0=0) using assms(2) unfolding IsCompact˙def by auto moreover
have 0∈T using empty˙open topSpaceAssum by auto moreover
have ⋃T∈T using topSpaceAssum unfolding IsATopology˙def by auto ultimately

```
    have thesis by auto
  }
  moreover
  {
    assume noEmpty:B≠0
    let U=–⟨U,V⟩∈T×T. A⊆U ∧ U∩V=0″
    {
      fix y assume y∈B
      then have y∈⋃T using assms(3) unfolding IsCompact˙def by auto
     with ‹y∈B› have ∃U∈T. ∃V∈T. A⊆U ∧ y∈V ∧ U∩V=0 using T2˙compact˙point
assms(1,2,4) by auto
      then have ∃⟨U,V⟩∈U. y∈V by auto
    }
    then have ∀y∈B. ∃⟨U,V⟩∈U. y∈V by auto
    then have B⊆⋃–snd(B). B∈U″ by auto
    moreover have –snd(B). B∈U″∈Pow(T) by auto
   ultimately have ∃N∈FinPow(–snd(B). B∈U″). B⊆⋃N using assms(3) unfolding
IsCompact˙def by auto
    then obtain N where ss:N∈FinPow(–snd(B). B∈U″) B⊆⋃N by auto
     with ‹–snd(B). B∈U″∈Pow(T)› have B⊆⋃N N∈Pow(T) unfolding FinPow˙def
by auto
    then have NN:B⊆⋃N ⋃N∈T using topSpaceAssum unfolding IsATopology˙def
by auto
    from ss have Finite(N)N⊆–snd(B). B∈U″ unfolding FinPow˙def by auto
    then obtain n where n∈nat N≈n unfolding Finite˙def by auto
    then have N≲n using eqpoll˙imp˙lepoll by auto
    from noEmpty ‹B⊆⋃N› have NnoEmpty:N≠0 by auto
    let QQ=–⟨n,–fst(B). B∈–A∈U. snd(A)=n″″⟩. n∈N″
      have QQPi:QQ:N→—fst(B). B∈–A∈U. snd(A)=n″″. n∈N″ unfolding Pi˙def
function˙def domain˙def by auto
    {
      fix n assume n∈N
      with ‹N⊆–snd(B). B∈U″› obtain B where n=snd(B) B∈U by auto
      then have fst(B)∈–fst(B). B∈–A∈U. snd(A)=n″″ by auto
      then have –fst(B). B∈–A∈U. snd(A)=n″″≠0 by auto moreover
      from ‹n∈N› have ⟨n,–fst(B). B∈–A∈U. snd(A)=n″″⟩∈QQ by auto
     with QQPi have QQn=–fst(B). B∈–A∈U. snd(A)=n″″ using apply˙equality by
auto
      ultimately have QQn≠0 by auto
    }
    then have ∀n∈N. QQn≠0 by auto
     with ‹n∈nat› ‹N≲n› have ∃f. f∈Pi(N,λt. QQt) ∧ (∀t∈N. ft∈QQt) using fi-
nite˙choice unfolding AxiomCardinalChoiceGen˙def
      by auto
    then obtain f where fPI:f∈Pi(N,λt. QQt) (∀t∈N. ft∈QQt) by auto
    from fPI(1) NnoEmpty have range(f)≠0 unfolding Pi˙def range˙def domain˙def
converse˙def by (safe,blast)
    {
      fix t assume t∈N
```

then have ft∈QQt using fPI(2) by auto
with ‹t∈N› have ft∈⋃(QQN) QQt⊆⋃(QQN) using func˙imagedef QQPi by auto
}
then have reg:∀t∈N. ft∈⋃(QQN) ∀t∈N. QQt⊆⋃(QQN) by auto
{
fix tt assume tt∈f
with fPI(1) have tt∈Sigma(N, ()(QQ)) unfolding Pi˙def by auto
then have tt∈(⋃xa∈N. ⋃y∈QQxa. −⟨xa,y⟩″) unfolding Sigma˙def by auto
then obtain xa y where xa∈N y∈QQxa tt=⟨xa,y⟩ by auto
with reg(2) have y∈⋃(QQN) by blast
with ‹tt=⟨xa,y⟩› ‹xa∈N› have tt∈(⋃xa∈N. ⋃y∈⋃(QQN). −⟨xa,y⟩″) by auto
then have tt∈N×(⋃(QQN)) unfolding Sigma˙def by auto
}
then have ffun:f:N→⋃(QQN) using fPI(1) unfolding Pi˙def by auto
then have f∈surj(N,range(f)) using fun˙is˙surj by auto
with ‹N≲n› ‹n∈nat› have range(f)≲N using surj˙fun˙inv˙2 nat˙into˙Ord by auto
with ‹N≲n› have range(f)≲n using lepoll˙trans by blast
with ‹n∈nat› have Finite(range(f)) using n˙lesspoll˙nat lesspoll˙nat˙is˙Finite lesspoll˙trans1 by auto
moreover from ffun have rr:range(f)⊆⋃(QQN) unfolding Pi˙def by auto
then have range(f)⊆T by auto
ultimately have range(f)∈FinPow(T) unfolding FinPow˙def by auto
then have ⋂range(f)∈T using fin˙inter˙open˙open ‹range(f)≠0› by auto moreover
{
fix S assume S∈range(f)
with rr have S∈⋃(QQN) by blast
then have ∃B∈(QQN). S ∈ B using Union˙iff by auto
then obtain B where B∈(QQN) S∈B by auto
then have ∃rr∈N. ⟨rr,B⟩∈QQ unfolding image˙def by auto
then have ∃rr∈N. B=−fst(B). B∈−A∈U. snd(A)=rr″″ by auto
with ‹S∈B› obtain rr where ⟨S,rr⟩∈U by auto
then have A⊆S by auto
}
then have A⊆⋂range(f) using ‹range(f)≠0› by auto moreover
{
fix y assume y∈(⋃N)∩(⋂range(f))
then have reg:(∀S∈range(f). y∈S)∧(∃t∈N. y∈t) by auto
then obtain t where t∈N y∈t by auto
then have ⟨t, −fst(B). B∈−A∈U. snd(A)=t″″⟩∈QQ by auto
then have ft∈range(f) using apply˙rangeI ffun by auto
with reg have yft:y∈ft by auto
with ‹t∈N› fPI(2) have ft∈QQt by auto
with ‹t∈N› have ft∈−fst(B). B∈−A∈U. snd(A)=t″″ using apply˙equality QQPi by auto
then have ⟨ft,t⟩∈U by auto
then have ft∩t=0 by auto
with ‹y∈t› yft have False by auto
}

then have $(\bigcap \text{range}(f))\cap(\bigcup N)=0$ by blast moreover
note NN
ultimately have thesis by auto
}
ultimately show thesis by auto
qed

A compact Hausdorff space is normal.

corollary (in topology0) T2˙compact˙is˙normal:
assumes T–is T_2″ $(\bigcup T)$–is compact in″T
shows T–is normal″ unfolding IsNormal˙def
proof-
from assms(2) have car˙nat:$(\bigcup T)$–is compact of cardinal″nat–in″T using Compact˙is˙card˙nat by auto
{
fix A B assume A–is closed in″T B–is closed in″TA∩B=0
then have com:$((\bigcup T)\cap A)$–is compact of cardinal″nat–in″T $((\bigcup T)\cap B)$–is compact of cardinal″nat–in″T using compact˙closed[OF car˙nat]
by auto
from ‹A–is closed in″T›‹B–is closed in″T› have $(\bigcup T)\cap A=A(\bigcup T)\cap B=B$ unfolding IsClosed˙def by auto
with com have A–is compact of cardinal″nat–in″T B–is compact of cardinal″nat–in″T by auto
then have A–is compact in″TB–is compact in″T using Compact˙is˙card˙nat by auto
with ‹A∩B=0› have ∃U∈T. ∃V∈T. A⊆U ∧ B⊆V ∧ U∩V=0 using T2˙compact˙compact assms(1) by auto
}
then show ∀A. A –is closed in″ T ⟶ (∀B. B –is closed in″ T ∧ A ∩ B = 0 ⟶ (∃U∈T. ∃V∈T. A ⊆ U ∧ B ⊆ V ∧ U ∩ V = 0))
by auto
qed

60.2 Hereditability

A topological property is hereditary if whenever a space has it, every subspace also has it.

definition IsHer (˙–is hereditary″ 90)
where P –is hereditary″ ≡ ∀T. T–is a topology″ ∧ P(T) ⟶ (∀A∈Pow$(\bigcup T)$. P(T–restricted to″A))

lemma subspace˙of˙subspace:
assumes A⊆BB⊆$\bigcup T$
shows T–restricted to″A=(T–restricted to″B)–restricted to″A
proof
from assms have S:∀S∈T. A∩(B∩S)=A∩S by auto
then show T –restricted to″ A ⊆ T –restricted to″ B –restricted to″ A unfolding RestrictedTo˙def

by auto
from S show T –restricted to″ B –restricted to″ A ⊆ T –restricted to″ A unfolding RestrictedTo˙def
by auto
qed

The separation properties T_0, T_1, T_2 y T_3 are hereditary.

theorem regular˙here:
assumes T–is regular″ A∈Pow(⋃T) shows (T–restricted to″A)–is regular″
proof-
{
fix C
assume A:C–is closed in″(T–restricted to″A)
{fix y assume y∈⋃(T–restricted to″A)y∉C
with A have (⋃(T–restricted to″A))-C∈(T–restricted to″A)C⊆⋃(T–restricted to″A) y∈⋃(T–restricted to″A)y∉C unfolding IsClosed˙def
by auto
moreover
with assms(2) have ⋃(T–restricted to″A)=A unfolding RestrictedTo˙def by auto
ultimately have A-C∈T–restricted to″A y∈Ay∉CC∈Pow(A) by auto
then obtain S where S∈T A∩S=A-C y∈Ay∉C unfolding RestrictedTo˙def by auto
then have y∈A-CA∩S=A-C by auto
with ‹C∈Pow(A)› have y∈A∩SC=A-A∩S by auto
then have y∈S C=A-S by auto
with assms(2) have y∈S C⊆⋃T-S by auto
moreover
from ‹S∈T› have ⋃T-(⋃T-S)=S by auto
moreover
with ‹S∈T› have (⋃T-S) –is closed in″T using IsClosed˙def by auto
ultimately have y∈⋃T-(⋃T-S) (⋃T-S) –is closed in″T by auto
with assms(1) have ∀y∈⋃T-(⋃T-S). ∃U∈T. ∃V∈T. (⋃T-S)⊆U∧y∈V∧U∩V=0 unfolding IsRegular˙def by auto
with ‹y∈⋃T-(⋃T-S)› have ∃U∈T. ∃V∈T. (⋃T-S)⊆U∧y∈V∧U∩V=0 by auto
then obtain U V where U∈TV∈T ⋃T-S⊆Uy∈VU∩V=0 by auto
then have A∩U∈(T–restricted to″A)A∩V∈(T–restricted to″A) C⊆Uy∈V(A∩U)∩(A∩V)=0
unfolding RestrictedTo˙def using ‹C⊆⋃T-S› by auto
moreover
with ‹C∈Pow(A)›‹y∈A› have C⊆A∩Uy∈A∩V by auto
ultimately have ∃U∈(T–restricted to″A). ∃V∈(T–restricted to″A). C⊆U∧y∈V∧U∩V=0 by auto
}
then have ∀x∈⋃(T–restricted to″A)-C. ∃U∈(T–restricted to″A). ∃V∈(T–restricted to″A). C⊆U∧x∈V∧U∩V=0 by auto
}
then have ∀C. C–is closed in″(T–restricted to″A) ⟶ (∀x∈⋃(T–restricted to″A)-C. ∃U∈(T–restricted to″A). ∃V∈(T–restricted to″A). C⊆U∧x∈V∧U∩V=0)
by blast
then show thesis using IsRegular˙def by auto

qed

corollary here˙regular:
shows IsRegular –is hereditary″ using regular˙here IsHer˙def by auto

theorem T1˙here:
assumes T–is T_1″ A∈Pow(⋃T) shows (T–restricted to″A)–is T_1″
proof-
from assms(2) have un:⋃(T–restricted to″A)=A unfolding RestrictedTo˙def by auto
{
fix x y
assume x∈Ay∈Ax≠y
with ‹A∈Pow(⋃T)› have x∈⋃Ty∈⋃Tx≠y by auto
then have ∃U∈T. x∈U∧y∉U using assms(1) isT1˙def by auto
then obtain U where U∈Tx∈Uy∉U by auto
with ‹x∈A› have A∩U∈(T–restricted to″A) x∈A∩U y∉A∩U unfolding RestrictedTo˙def by auto
then have ∃U∈(T–restricted to″A). x∈U∧y∉U by blast
}
with un have ∀x y. x∈⋃(T–restricted to″A) ∧ y∈⋃(T–restricted to″A) ∧ x≠y ⟶ (∃U∈(T–restricted to″A). x∈U∧y∉U)
by auto
then show thesis using isT1˙def by auto
qed

corollary here˙T1:
shows isT1 –is hereditary″ using T1˙here IsHer˙def by auto

lemma here˙and:
assumes P –is hereditary″ Q –is hereditary″
shows (λT. P(T) ∧ Q(T)) –is hereditary″ using assms unfolding IsHer˙def by auto

corollary here˙T3:
shows isT3 –is hereditary″ using here˙and[OF here˙T1 here˙regular] unfolding IsHer˙def isT3˙def.

lemma T2˙here:
assumes T–is T_2″ A∈Pow(⋃T) shows (T–restricted to″A)–is T_2″
proof-
from assms(2) have un:⋃(T–restricted to″A)=A unfolding RestrictedTo˙def by auto
{
fix x y
assume x∈Ay∈Ax≠y
with ‹A∈Pow(⋃T)› have x∈⋃Ty∈⋃Tx≠y by auto
then have ∃U∈T. ∃V∈T. x∈U∧y∈V∧U∩V=0 using assms(1) isT2˙def by auto
then obtain U V where U∈T V∈Tx∈Uy∈VU∩V=0 by auto
with ‹x∈A›‹y∈A› have A∩U∈(T–restricted to″A)A∩V∈(T–restricted to″A)

x∈A∩U y∈A∩V (A∩U)∩(A∩V)=0unfolding RestrictedTo˙def by auto
then have ∃U∈(T–restricted to″A). ∃V∈(T–restricted to″A). x∈U∧y∈V∧U∩V=0
unfolding Bex˙def by auto
}
with un have ∀x y. x∈⋃(T–restricted to″A) ∧ y∈⋃(T–restricted to″A) ∧ x≠y
⟶ (∃U∈(T–restricted to″A). ∃V∈(T–restricted to″A). x∈U∧y∈V∧U∩V=0)
by auto
then show thesis using isT2˙def by auto
qed

corollary here˙T2:
shows isT2 –is hereditary″ using T2˙here IsHer˙def by auto

lemma T0˙here:
assumes T–is T_0″ A∈Pow(⋃T) shows (T–restricted to″A)–is T_0″
proof-
from assms(2) have un:⋃(T–restricted to″A)=A unfolding RestrictedTo˙def by
auto
{
fix x y
assume x∈Ay∈Ax≠y
with ‹A∈Pow(⋃T)› have x∈⋃Ty∈⋃Tx≠y by auto
then have ∃U∈T. (x∈U∧y∉U)∨(y∈U∧x∉U) using assms(1) isT0˙def by auto
then obtain U where U∈T (x∈U∧y∉U)∨(y∈U∧x∉U) by auto
with ‹x∈A›‹y∈A› have A∩U∈(T–restricted to″A) (x∈A∩U∧y∉A∩U)∨(y∈A∩U∧x∉A∩U)
unfolding RestrictedTo˙def by auto
then have ∃U∈(T–restricted to″A). (x∈U∧y∉U)∨(y∈U∧x∉U) unfolding Bex˙def
by auto
}
with un have ∀x y. x∈⋃(T–restricted to″A) ∧ y∈⋃(T–restricted to″A) ∧ x≠y
⟶ (∃U∈(T–restricted to″A). (x∈U∧y∉U)∨(y∈U∧x∉U))
by auto
then show thesis using isT0˙def by auto
qed

corollary here˙T0:
shows isT0 –is hereditary″ using T0˙here IsHer˙def by auto

60.3 Spectrum and anti-properties

The spectrum of a topological property is a class of sets such that all topologies defined over that set have that property.

The spectrum of a property gives us the list of sets for which the property doesn't give any topological information. Being in the spectrum of a topological property is an invariant in the category of sets and function; mening that equipollent sets are in the same spectra.

definition Spec (˙ –is in the spectrum of″ ˙ 99)
where Spec(K,P) ≡ ∀T. ((T–is a topology″ ∧ ⋃T≈K) ⟶ P(T))

```
lemma equipollent˙spect:
  assumes A≈B B –is in the spectrum of″ P
  shows  A –is in the spectrum of″ P
proof-
  from assms(2) have ∀T. ((T–is a topology″ ∧ ⋃T≈B) ⟶ P(T)) using Spec˙def
by auto
  then have ∀T. ((T–is a topology″ ∧ ⋃T≈A) ⟶ P(T)) using eqpoll˙trans[OF ˙
assms(1)] by auto
  then show thesis using Spec˙def by auto
qed

theorem eqpoll˙iff˙spec:
  assumes A≈B
  shows (B –is in the spectrum of″ P) ⟷ (A –is in the spectrum of″ P)
proof
  assume B –is in the spectrum of″ P
  with assms equipollent˙spect show A –is in the spectrum of″ P by auto
next
  assume A –is in the spectrum of″ P
  moreover
  from assms have B≈A using eqpoll˙sym by auto
  ultimately show B –is in the spectrum of″ P using equipollent˙spect by auto
qed
```

From the previous statement, we see that the spectrum could be formed only by representative of clases of sets. If AC holds, this means that the spectrum can be taken as a set or class of cardinal numbers.

Here is an example of the spectrum. The proof lies in the indiscrite filter –A″ that can be build for any set. In this proof, we see that without choice, there is no way to define the sepctrum of a property with cardinals because if a set is not comparable with any ordinal, its cardinal is defined as 0 without the set being empty.

```
theorem T4˙spectrum:
  shows (A –is in the spectrum of″ isT4) ⟷ A ≲ 1
proof
  assume A –is in the spectrum of″ isT4
  then have reg:∀T. ((T–is a topology″ ∧ ⋃T≈A) ⟶ (T –is T₄″)) using Spec˙def
by auto
  {
    assume A≠0
    then obtain x where x∈A by auto
    then have x∈⋃–A″ by auto
    moreover
    then have –A″ –is a filter on″⋃–A″ using IsFilter˙def by auto
    moreover
     then have (–A″∪–0″) –is a topology″ ∧ ⋃(–A″∪–0″)=A using top˙of˙filter by
auto
```

then have top:(–A″∪–0″) –is a topology″ ⋃(–A″∪–0″)≈A using eqpoll˙refl by auto
then have (–A″∪–0″) –is T_4″ using reg by auto
then have (–A″∪–0″) –is T_2″ using topology0.T3˙is˙T2 topology0.T4˙is˙T3 topology0˙def top by auto
ultimately have ⋃–A″=–x″ using filter˙T2˙imp˙card1[of –A″x] by auto
then have A=–x″ by auto
then have A≈1 using singleton˙eqpoll˙1 by auto
}
moreover
have A=0 ⟶ A≈0 by auto
ultimately have A≈1∨A≈0 by blast
then show A≲1 using empty˙lepollI eqpoll˙imp˙lepoll eq˙lepoll˙trans by auto
next
assume A≲1
have A=0∨A≠0 by auto
then obtain E where A=0∨E∈A by auto
then have A≈0∨E∈A by auto
with ‹A≲1› have A≈0∨A=–E″ using lepoll˙1˙is˙sing by auto
then have A≈0∨A≈1 using singleton˙eqpoll˙1 by auto
{
fix T
assume AS:T–is a topology″⋃T≈A
{
assume A≈0
with AS have T–is a topology″ and empty:⋃T=0 using eqpoll˙trans eqpoll˙0˙is˙0 by auto
then have T–is T_2″ using isT2˙def by auto
then have T–is T_1″ using T2˙is˙T1 by auto
moreover
from empty have T⊆–0″ by auto
with AS(1) have T=–0″ using empty˙open by auto
from empty have rr:∀A. A–is closed in″T ⟶ A=0 using IsClosed˙def by auto
have ∃U∈T. ∃V∈T. 0⊆U∧0⊆V∧U∩V=0 using empty˙open AS(1) by auto
with rr have ∀A. A–is closed in″T ⟶ (∀B. B–is closed in″T ∧ A∩B=0 ⟶ (∃U∈T. ∃V∈T. A⊆U∧B⊆V∧U∩V=0))
by blast
then have T–is normal″ using IsNormal˙def by auto
with ‹T–is T_1″› have T–is T_4″ using isT4˙def by auto
}
moreover
{
assume A≈1
with AS have T–is a topology″ and NONempty:⋃T≈1 using eqpoll˙trans[of ⋃TA1] by auto
then have ⋃T≲1 using eqpoll˙imp˙lepoll by auto
moreover
{
assume ⋃T=0

```
      then have 0≈⋃T by auto
      with NONempty have 0≈1 using eqpoll˙trans by blast
      then have 0=1 using eqpoll˙0˙is˙0 eqpoll˙sym by auto
      then have False by auto
    }
    then have ⋃T≠0 by auto
    then obtain R where R∈⋃T by blast
    ultimately have ⋃T=–R″ using lepoll˙1˙is˙sing by auto
    {
      fix x y
      assume x–is closed in″Ty–is closed in″T x∩y=0
      then have x⊆⋃Ty⊆⋃T using IsClosed˙def by auto
      then have x=0∨y=0 using ‹x∩y=0› ‹⋃T=–R″› by force
      {
        assume x=0
        then have x⊆0y⊆⋃T using ‹y⊆⋃T› by auto
        moreover
        have 0∈T⋃T∈T using AS(1) IsATopology˙def empty˙open by auto
        ultimately have ∃U∈T. ∃V∈T. x⊆U∧y⊆V∧U∩V=0 by auto
      }
      moreover
      {
        assume x≠0
        with ‹x=0∨y=0› have y=0 by auto
        then have x⊆⋃Ty⊆0 using ‹x⊆⋃T› by auto
        moreover
        have 0∈T⋃T∈T using AS(1) IsATopology˙def empty˙open by auto
        ultimately have ∃U∈T. ∃V∈T. x⊆U∧y⊆V∧U∩V=0 by auto
      }
      ultimately
      have (∃U∈T. ∃V∈T. x ⊆ U ∧ y ⊆ V ∧ U ∩ V = 0) by blast
    }
    then have T–is normal″ using IsNormal˙def by auto
    moreover
    {
      fix x y
      assume x∈⋃Ty∈⋃Tx≠y
      with ‹⋃T=–R″› have False by auto
      then have ∃U∈T. x∈U∧y∉U by auto
    }
    then have T–is T1″ using isT1˙def by auto
    ultimately have T–is T4″ using isT4˙def by auto
  }
  ultimately have T–is T4″ using ‹A≈0∨A≈1› by auto
}
then have ∀T. (T–is a topology″ ∧ ⋃T ≈ A) ⟶ (T–is T4″) by auto
then show A –is in the spectrum of″ isT4 using Spec˙def by auto
qed
```

If the topological properties are related, then so are the spectra.

lemma P˙imp˙Q˙spec˙inv:
assumes ∀T. T–is a topology″ ⟶ (Q(T) ⟶ P(T)) A –is in the spectrum of″ Q
shows A –is in the spectrum of″ P
proof-
from assms(2) have ∀T. T–is a topology″ ∧ ⋃T ≈ A ⟶ Q(T) using Spec˙def by auto
with assms(1) have ∀T. T–is a topology″ ∧ ⋃T ≈ A ⟶ P(T) by auto
then show thesis using Spec˙def by auto
qed

Since we already now the spectrum of T_4; if we now the spectrum of T_0, it should be easier to compute the spectrum of T_1, T_2 and T_3.

theorem T0˙spectrum:
shows (A –is in the spectrum of″ isT0) ⟷ A ≲ 1
proof
assume A –is in the spectrum of″ isT0
then have reg:∀T. ((T–is a topology″ ∧ ⋃T≈A) ⟶ (T –is T_0″)) using Spec˙def by auto
{
assume A≠0
then obtain x where x∈A by auto
then have x∈⋃–A″ by auto
moreover
then have –A″ –is a filter on″⋃–A″ using IsFilter˙def by auto
moreover
then have (–A″∪–0″) –is a topology″ ∧ ⋃(–A″∪–0″)=A using top˙of˙filter by auto
then have (–A″∪–0″) –is a topology″ ∧ ⋃(–A″∪–0″)≈A using eqpoll˙refl by auto
then have (–A″∪–0″) –is T_0″ using reg by auto
{
fix y
assume y∈Ax≠y
with ‹(–A″∪–0″) –is T_0″› obtain U where U∈(–A″∪–0″) and dis:(x ∈ U ∧ y ∉ U) ∨ (y ∈ U ∧ x ∉ U) using isT0˙def by auto
then have U=A by auto
with dis ‹y∈A› ‹x∈⋃–A″› have False by auto
}
then have ∀y∈A. y=x by auto
with ‹x∈⋃–A″› have A=–x″ by blast
then have A≈1 using singleton˙eqpoll˙1 by auto
}
moreover
have A=0 ⟶ A≈0 by auto
ultimately have A≈1∨A≈0 by blast
then show A≲1 using empty˙lepollI eqpoll˙imp˙lepoll eq˙lepoll˙trans by auto
next
assume A≲1
{

fix T
assume T–is a topology″
then have (T–is T_4″)⟶(T–is T_0″) using topology0.T4˙is˙T3 topology0.T3˙is˙T2 T2˙is˙T1 T1˙is˙T0
topology0˙def by auto
}
then have ∀T. T–is a topology″ ⟶ ((T–is T_4″)⟶(T–is T_0″)) by auto
then have (A –is in the spectrum of″ isT4) ⟶ (A –is in the spectrum of″ isT0)
using P˙imp˙Q˙spec˙inv[of λT. (T–is T_4″)λT. T–is T_0″] by auto
then show (A –is in the spectrum of″ isT0) using T4˙spectrum ‹A≲1› by auto
qed

theorem T1˙spectrum:
shows (A –is in the spectrum of″ isT1) ⟷ A ≲ 1
proof-
note T2˙is˙T1 topology0.T3˙is˙T2 topology0.T4˙is˙T3
then have (A –is in the spectrum of″ isT4) ⟶ (A –is in the spectrum of″ isT1)
using P˙imp˙Q˙spec˙inv[of isT4isT1] topology0˙def by auto
moreover
note T1˙is˙T0
then have (A –is in the spectrum of″ isT1) ⟶ (A –is in the spectrum of″isT0)
using P˙imp˙Q˙spec˙inv[of isT1isT0] by auto
moreover
note T0˙spectrum T4˙spectrum
ultimately show thesis by blast
qed

theorem T2˙spectrum:
shows (A –is in the spectrum of″ isT2) ⟷ A ≲ 1
proof-
note topology0.T3˙is˙T2 topology0.T4˙is˙T3
then have (A –is in the spectrum of″ isT4) ⟶ (A –is in the spectrum of″ isT2)
using P˙imp˙Q˙spec˙inv[of isT4isT2] topology0˙def by auto
moreover
note T2˙is˙T1
then have (A –is in the spectrum of″ isT2) ⟶ (A –is in the spectrum of″isT1)
using P˙imp˙Q˙spec˙inv[of isT2isT1] by auto
moreover
note T1˙spectrum T4˙spectrum
ultimately show thesis by blast
qed

theorem T3˙spectrum:
shows (A –is in the spectrum of″ isT3) ⟷ A ≲ 1
proof-
note topology0.T4˙is˙T3
then have (A –is in the spectrum of″ isT4) ⟶ (A –is in the spectrum of″ isT3)
using P˙imp˙Q˙spec˙inv[of isT4isT3] topology0˙def by auto
moreover

note topology0.T3˙is˙T2
then have (A –is in the spectrum of″ isT3) ⟶ (A –is in the spectrum of″isT2)
using P˙imp˙Q˙spec˙inv[of isT3isT2] topology0˙def by auto
moreover
note T2˙spectrum T4˙spectrum
ultimately show thesis by blast
qed

theorem compact˙spectrum:
shows (A –is in the spectrum of″ (λT. (⋃T) –is compact in″T)) ⟷ Finite(A)
proof
assume A –is in the spectrum of″ (λT. (⋃T) –is compact in″T)
then have reg:∀T. T–is a topology″ ∧ ⋃T≈A ⟶ ((⋃T) –is compact in″T) using Spec˙def by auto
have Pow(A)–is a topology″ ∧ ⋃Pow(A)=A using Pow˙is˙top by auto
then have Pow(A)–is a topology″ ∧ ⋃Pow(A)≈A using eqpoll˙refl by auto
with reg have A–is compact in″Pow(A) by auto
moreover
have —x″. x∈A″∈Pow(Pow(A)) by auto
moreover
have ⋃—x″. x∈A″=A by auto
ultimately have ∃N∈FinPow(—x″. x∈A″). A⊆⋃N using IsCompact˙def by auto
then obtain N where N∈FinPow(—x″. x∈A″) A⊆⋃N by auto
then have N⊆—x″. x∈A″ Finite(N) A⊆⋃N using FinPow˙def by auto
{
fix t
assume t∈—x″. x∈A″
then obtain x where x∈At=–x″ by auto
with ‹A⊆⋃N› have x∈⋃N by auto
then obtain B where B∈Nx∈B by auto
with ‹N⊆—x″. x∈A″› have B=–x″ by auto
with ‹t=–x″›‹B∈N› have t∈N by auto
}
with ‹N⊆—x″. x∈A″› have N=—x″. x∈A″ by auto
with ‹Finite(N)› have Finite(—x″. x∈A″) by auto
let B=–⟨x,–x″⟩. x∈A″
have B:A→—x″. x∈A″ unfolding Pi˙def function˙def by auto
then have B:bij(A,—x″. x∈A″) unfolding bij˙def inj˙def surj˙def using apply˙equality by auto
then have A≈—x″. x∈A″ using eqpoll˙def by auto
with ‹Finite(—x″. x∈A″)› show Finite(A) using eqpoll˙imp˙Finite˙iff by auto
next
assume Finite(A)
{
fix T assume T–is a topology″ ⋃T≈A
with ‹Finite(A)› have Finite(⋃T) using eqpoll˙imp˙Finite˙iff by auto
then have Finite(Pow(⋃T)) using Finite˙Pow by auto
moreover
have T⊆Pow(⋃T) by auto

ultimately have Finite(T) using subset˙Finite by auto
{
fix M
assume M∈Pow(T)⋃T⊆⋃M
with ‹Finite(T)› have Finite(M) using subset˙Finite by auto
with ‹⋃T⊆⋃M› have ∃N∈FinPow(M). ⋃T⊆⋃N using FinPow˙def by auto
}
then have (⋃T)–is compact in″T unfolding IsCompact˙def by auto
}
then show A –is in the spectrum of″ (λT. (⋃T) –is compact in″T) using Spec˙def by auto
qed

It is, at least for some people, surprising that the spectrum of some properties cannot be completely determined in ZF.

theorem compactK˙spectrum:
assumes –the axiom of″K–choice holds for subsets″(Pow(K)) Card(K)
shows (A –is in the spectrum of″ (λT. ((⋃T)–is compact of cardinal″ csucc(K)–in″T))) ⟷ (A≲K)
proof
assume A –is in the spectrum of″ (λT. ((⋃T)–is compact of cardinal″ csucc(K)–in″T))
then have reg:∀T. T–is a topology″∧⋃T≈A ⟶ ((⋃T)–is compact of cardinal″ csucc(K)–in″T) using Spec˙def by auto
then have A–is compact of cardinal″ csucc(K) –in″ Pow(A) using Pow˙is˙top[of A] by auto
then have ∀M∈Pow(Pow(A)). A⊆⋃M ⟶ (∃N∈Pow(M). A⊆⋃N ∧ N≺csucc(K)) unfolding IsCompactOfCard˙def by auto
moreover
have —x″. x∈A″∈Pow(Pow(A)) by auto
moreover
have A=⋃—x″. x∈A″ by auto
ultimately have ∃N∈Pow(—x″. x∈A″). A⊆⋃N ∧ N≺csucc(K) by auto
then obtain N where N∈Pow(—x″. x∈A″) A⊆⋃N N≺csucc(K) by auto
then have N⊆—x″. x∈A″ N≺csucc(K) A⊆⋃N using FinPow˙def by auto
{
fix t
assume t∈—x″. x∈A″
then obtain x where x∈At=–x″ by auto
with ‹A⊆⋃N› have x∈⋃N by auto
then obtain B where B∈Nx∈B by auto
with ‹N⊆—x″. x∈A″› have B=–x″ by auto
with ‹t=–x″›‹B∈N› have t∈N by auto
}
with ‹N⊆—x″. x∈A″› have N=—x″. x∈A″ by auto
let B=–⟨x,–x″⟩. x∈A″
from ‹N=—x″. x∈A″› have B:A→ N unfolding Pi˙def function˙def by auto
with ‹N=—x″. x∈A″› have B:inj(A,N) unfolding inj˙def using apply˙equality by auto
then have A≲N using lepoll˙def by auto

```
  with ‹N≺csucc(K)› have A≺csucc(K) using lesspoll˙trans1 by auto
  then show A≲K using Card˙less˙csucc˙eq˙le assms(2) by auto
next
  assume A≲K
  {
    fix T
    assume T–is a topology″⋃T≈A
    have Pow(⋃T)–is a topology″ using Pow˙is˙top by auto
    {
      fix B
      assume AS:B∈Pow(⋃T)
      then have —i″. i∈B″⊆—i″ .i∈⋃T″ by auto
      moreover
      have B=⋃—i″. i∈B″ by auto
      ultimately have ∃S∈Pow(—i″. i∈⋃T″). B=⋃S by auto
      then have B∈–⋃U. U∈Pow(—i″. i∈⋃T″)″ by auto
    }
    moreover
    {
      fix B
      assume AS:B∈–⋃U. U∈Pow(—i″. i∈⋃T″)″
      then have B∈Pow(⋃T) by auto
    }
    ultimately
     have base:—x″. x∈⋃T″ –is a base for″Pow(⋃T) unfolding IsAbaseFor˙def by
auto
    let f=–⟨i,–i″⟩. i∈⋃T″
    have f:f:⋃T→ —x″. x∈⋃T″ using Pi˙def function˙def by auto
    moreover
    {
      fix w x
      assume as:w∈⋃Tx∈⋃Tfw=fx
      with f have fw=–w″ fx=–x″ using apply˙equality by auto
      with as(3) have w=x by auto
    }
    with f have f:inj(⋃T,—x″. x∈⋃T″) unfolding inj˙def by auto
    moreover
    {
      fix xa
      assume xa∈—x″. x∈⋃T″
      then obtain x where x∈⋃Txa=–x″ by auto
      with f have fx=xa using apply˙equality by auto
      with ‹x∈⋃T› have ∃x∈⋃T. fx=xa by auto
    }
    then have ∀xa∈—x″. x∈⋃T″. ∃x∈⋃T. fx=xa by blast
    ultimately have f:bij(⋃T,—x″. x∈⋃T″) unfolding bij˙def surj˙def by auto
    then have ⋃T≈—x″. x∈⋃T″ using eqpoll˙def by auto
    then have —x″. x∈⋃T″≈⋃T using eqpoll˙sym by auto
    with ‹⋃T≈A› have —x″. x∈⋃T″≈A using eqpoll˙trans by blast
```

then have —x″. x∈⋃T″≲A using eqpoll˙imp˙lepoll by auto
with ‹A≲K› have —x″. x∈⋃T″≲K using lepoll˙trans by blast
then have —x″. x∈⋃T″≺csucc(K) using assms(2) Card˙less˙csucc˙eq˙le by auto
with base have Pow(⋃T) –is of second type of cardinal″csucc(K) unfolding IsSecondOfCard˙def by auto
moreover
have ⋃Pow(⋃T)=⋃T by auto
with calculation assms(1) ‹Pow(⋃T)–is a topology″› have (⋃T) –is compact of cardinal″csucc(K)–in″Pow(⋃T)
using compact˙of˙cardinal˙Q[of KPow(⋃T)] by auto
moreover
have T⊆Pow(⋃T) by auto
ultimately have (⋃T) –is compact of cardinal″csucc(K)–in″T using compact˙coarser by auto
}
then show A –is in the spectrum of″ (λT. ((⋃T)–is compact of cardinal″csucc(K) –in″T)) using Spec˙def by auto
qed

theorem compactK˙spectrum˙reverse:
assumes ∀A. (A –is in the spectrum of″ (λT. ((⋃T)–is compact of cardinal″ csucc(K)–in″T))) ⟷ (A≲K) InfCard(K)
shows –the axiom of″K–choice holds for subsets″(Pow(K))
proof-
have K≲K using lepoll˙refl by auto
then have K –is in the spectrum of″ (λT. ((⋃T)–is compact of cardinal″ csucc(K)–in″T)) using assms(1) by auto
moreover
have Pow(K)–is a topology″ using Pow˙is˙top by auto
moreover
have ⋃Pow(K)=K by auto
then have ⋃Pow(K)≈K using eqpoll˙refl by auto
ultimately
have K –is compact of cardinal″ csucc(K)–in″Pow(K) using Spec˙def by auto
then show thesis using Q˙disc˙comp˙csuccQ˙eq˙Q˙choice˙csuccQ assms(2) by auto
qed

This last theorem states that if one of the forms of the axiom of choice related to this compactness property fails, then the spectrum will be different. Notice that even for Lindelf spaces that will happend.

The spectrum gives us the posibility to define what an anti-property means. A space is anti-P if the only subspaces which have the property are the ones in the spectrum of P. This concept tries to put together spaces that are completely opposite to spaces where P(T).

definition
antiProperty (˙–is anti-″˙ 50)
where T–is anti-″P ≡ ∀A∈Pow(⋃T). P(T–restricted to″A) ⟶ (A –is in the spectrum of″ P)

abbreviation
ANTI(P) ≡ λT. (T–is anti-″P)

A first, very simple, but very useful result is the following: when the properties are related and the spectra are equal, then the anti-properties are related in the oposite direction.

theorem (in topology0) eq˙spect˙rev˙imp˙anti:
assumes ∀T. T–is a topology″ ⟶ P(T) ⟶ Q(T) ∀A. (A–is in the spectrum of″Q) ⟶ (A–is in the spectrum of″P)
and T–is anti-″Q
shows T–is anti-″P
proof-
{
fix A
assume A∈Pow(⋃T)P(T–restricted to″A)
with assms(1) have Q(T–restricted to″A) using Top˙1˙L4 by auto
with assms(3) ‹A∈Pow(⋃T)› have A–is in the spectrum of″Q using antiProperty˙def by auto
with assms(2) have A–is in the spectrum of″P by auto
}
then show thesis using antiProperty˙def by auto
qed

If a space can be P(T)∧Q(T) only in case the underlying set is in the spectrum of P; then Q(T)⟶ANTI(P,T) when Q is hereditary.

theorem Q˙P˙imp˙Spec:
assumes ∀T. ((T–is a topology″∧P(T)∧Q(T))⟶ ((⋃T)–is in the spectrum of″P))
and Q–is hereditary″
shows ∀T. T–is a topology″ ⟶ (Q(T)⟶(T–is anti-″P))
proof
fix T
{
assume T–is a topology″
{
assume Q(T)
{
assume ¬(T–is anti-″P)
then obtain A where A∈Pow(⋃T) P(T–restricted to″A)¬(A–is in the spectrum of″P)
unfolding antiProperty˙def by auto
from ‹Q(T)›‹T–is a topology″›‹A∈Pow(⋃T)› assms(2) have Q(T–restricted to″A)
unfolding IsHer˙def by auto
moreover
note ‹P(T–restricted to″A)› assms(1)
moreover

from ‹T–is a topology″› have (T–restricted to″A)–is a topology″ using topology0.Top˙1˙L4
topology0˙def by auto
moreover
from ‹A∈Pow(⋃T)› have ⋃(T–restricted to″A)=A unfolding RestrictedTo˙def by auto
ultimately have A–is in the spectrum of″P by auto
with ‹¬(A–is in the spectrum of″P)› have False by auto
}
then have T–is anti-″P by auto
}
then have Q(T)⟶(T–is anti-″P) by auto
}
then show (T –is a topology″) ⟶ (Q(T) ⟶ (T–is anti-″P)) by auto
qed

If a topologycal space has an hereditary property, then it has its double-anti property.

theorem (in topology0)her˙P˙imp˙anti2P:
assumes P–is hereditary″ P(T)
shows T–is anti-″ANTI(P)
proof-
{
assume ¬(T–is anti-″ANTI(P))
then have ∃A∈Pow(⋃T). ((T–restricted to″A)–is anti-″P)∧¬(A–is in the spectrum of″ANTI(P))
unfolding antiProperty˙def[of ˙ ANTI(P)] by auto
then obtain A where A˙def:A∈Pow(⋃T)¬(A–is in the spectrum of″ANTI(P))(T–restricted to″A)–is anti-″P
by auto
from ‹A∈Pow(⋃T)› have tot:⋃(T–restricted to″A)=A unfolding RestrictedTo˙def by auto
from A˙def have reg:∀B∈Pow(⋃(T–restricted to″A)). P((T–restricted to″A)–restricted to″B) ⟶ (B–is in the spectrum of″P)
unfolding antiProperty˙def by auto
have ∀B∈Pow(A). (T–restricted to″A)–restricted to″B=T–restricted to″B using subspace˙of˙subspace ‹A∈Pow(⋃T)› by auto
then have ∀B∈Pow(A). P(T–restricted to″B) ⟶ (B–is in the spectrum of″P) using reg tot
by force
moreover
have ∀B∈Pow(A). P(T–restricted to″B) using assms ‹A∈Pow(⋃T)› unfolding IsHer˙def using topSpaceAssum by blast
ultimately have reg2:∀B∈Pow(A). (B–is in the spectrum of″P) by auto
from ‹¬(A–is in the spectrum of″ANTI(P))› have ∃T. T–is a topology″ ∧ ⋃T≈A ∧ ¬(T–is anti-″P)
unfolding Spec˙def by auto
then obtain S where S–is a topology″ ⋃S≈A ¬(S–is anti-″P) by auto
from ‹¬(S–is anti-″P)› have ∃B∈Pow(⋃S). P(S–restricted to″B) ∧ ¬(B–is in

the spectrum of″P) unfolding antiProperty˙def by auto
then obtain B where B˙def:¬(B–is in the spectrum of″P) B∈Pow(⋃S) by auto
then have B≲⋃S using subset˙imp˙lepoll by auto
with ‹⋃S≈A› have B≲A using lepoll˙eq˙trans by auto
then obtain f where f∈inj(B,A) unfolding lepoll˙def by auto
then have f∈bij(B,range(f)) using inj˙bij˙range by auto
then have B≈range(f) unfolding eqpoll˙def by auto
with B˙def(1) have ¬(range(f)–is in the spectrum of″P) using eqpoll˙iff˙spec by auto
moreover
with ‹f∈inj(B,A)› have range(f)⊆A unfolding inj˙def Pi˙def by auto
with reg2 have range(f)–is in the spectrum of″P by auto
ultimately have False by auto
}
then show thesis by auto
qed

The anti-properties are always hereditary

theorem anti˙here:
shows ANTI(P)–is hereditary″
proof-
{
fix T
assume T –is a topology″ANTI(P,T)
{
fix A
assume A∈Pow(⋃T)
then have ⋃(T–restricted to″A)=A unfolding RestrictedTo˙def by auto
moreover
{
fix B
assume B∈Pow(A)P((T–restricted to″A)–restricted to″B)
with ‹A∈Pow(⋃T)› have B∈Pow(⋃T)P(T–restricted to″B) using subspace˙of˙subspace by auto
with ‹ANTI(P,T)› have B–is in the spectrum of″P unfolding antiProperty˙def by auto
}
ultimately have ∀B∈Pow(⋃(T–restricted to″A)). (P((T–restricted to″A)–restricted to″B)) ⟶ (B–is in the spectrum of″P)
by auto
then have ANTI(P,(T–restricted to″A)) unfolding antiProperty˙def by auto
}
then have ∀A∈Pow(⋃T). ANTI(P,(T–restricted to″A)) by auto
}
then show thesis using IsHer˙def by auto
qed

corollary (in topology0) anti˙imp˙anti3:
assumes T–is anti-″P

shows T–is anti-″ANTI(ANTI(P))
using anti˙here her˙P˙imp˙anti2P assms by auto

In the article [5], we can find some results on anti-properties.

theorem (in topology0) anti˙T0:
shows (T–is anti-″isT0) ⟷ T=–0,⋃T″
proof
assume T=–0,⋃T″
{
fix A
assume A∈Pow(⋃T)(T–restricted to″A) –is T_0″
{
fix B
assume B∈T–restricted to″A
then obtain S where S∈T and B=A∩S unfolding RestrictedTo˙def by auto
with ‹T=–0,⋃T″› have S∈–0,⋃T″ by auto
then have S=0∨S=⋃T by auto
with ‹B=A∩S›‹A∈Pow(⋃T)› have B=0∨B=A by auto
}
moreover
{
have 0∈–0,⋃T″ ⋃T∈–0,⋃T″ by auto
with ‹T=–0,⋃T″› have 0∈T(⋃T)∈T by auto
then have A∩0∈(T–restricted to″A) A∩(⋃T)∈(T–restricted to″A) using RestrictedTo˙def by auto
moreover
from ‹A∈Pow(⋃T)› have A∩(⋃T)=A by auto
ultimately have 0∈(T–restricted to″A) A∈(T–restricted to″A) by auto
}
ultimately have (T–restricted to″A)=–0,A″ by auto
with ‹(T–restricted to″A) –is T_0″› have –0,A″ –is T_0″ by auto
{
assume A≠0
then obtain x where x∈A by blast
{
fix y
assume y∈Ax≠y
with ‹–0,A″ –is T_0″› obtain U where U∈–0,A″ and dis:(x ∈ U ∧ y ∉ U) ∨ (y ∈ U ∧ x ∉ U) using isT0˙def by auto
then have U=A by auto
with dis ‹y∈A› ‹x∈A› have False by auto
}
then have ∀y∈A. y=x by auto
with ‹x∈A› have A=–x″ by blast
then have A≈1 using singleton˙eqpoll˙1 by auto
then have A≲1 using eqpoll˙imp˙lepoll by auto
then have A–is in the spectrum of″isT0 using T0˙spectrum by auto
}
moreover

```
    {
      assume A=0
      then have A≈0 by auto
      then have A≲1 using empty˙lepollI eq˙lepoll˙trans by auto
      then have A–is in the spectrum of″isT0 using T0˙spectrum by auto
    }
    ultimately have A–is in the spectrum of″isT0 by auto
  }
  then show T–is anti-″isT0 using antiProperty˙def by auto
next
  assume T–is anti-″isT0
  then have ∀A∈Pow(⋃T). (T–restricted to″A)–is T0″ ⟶ (A–is in the spectrum
of″isT0) using antiProperty˙def by auto
  then have reg:∀A∈Pow(⋃T). (T–restricted to″A)–is T0″ ⟶ (A≲1) using T0˙spectrum
by auto
  {
    assume ∃A∈T. A≠0∧ A≠⋃T
    then obtain A where A∈TA≠0A≠⋃T by auto
    then obtain x y where x∈A y∈⋃T-A by blast
    with ‹A∈T› have s:–x,y″∈Pow(⋃T) x≠y by auto
    note s
    moreover
    {
      fix b1 b2
      assume b1∈⋃(T–restricted to″–x,y″)b2∈⋃(T–restricted to″–x,y″)b1≠b2
      moreover
       from s have ⋃(T–restricted to″–x,y″)=–x,y″ unfolding RestrictedTo˙def by
auto
      ultimately have (b1=x∧b2=y)∨(b1=y∧b2=x) by auto
      with ‹x≠y› have (b1∈–x″∧b2∉–x″) ∨ (b2∈–x″∧b1∉–x″) by auto
      moreover
      from ‹y∈⋃T-A›‹x∈A› have –x″=–x,y″∩A by auto
       with ‹A∈T› have –x″∈(T–restricted to″–x,y″) unfolding RestrictedTo˙def by
auto
      ultimately have ∃U∈(T–restricted to″–x,y″). (b1∈U∧b2∉U) ∨ (b2∈U∧b1∉U)
by auto
    }
    then have (T–restricted to″–x,y″)–is T0″ using isT0˙def by auto
    ultimately have –x,y″≲1 using reg by auto
    moreover
    have x∈–x,y″ by auto
    ultimately have –x,y″=–x″ using lepoll˙1˙is˙sing[of –x,y″x] by auto
    moreover
    have y∈–x,y″ by auto
    ultimately have y∈–x″ by auto
    then have y=x by auto
    with ‹x≠y› have False by auto
  }
  then have T⊆–0,⋃T″ by auto
```

```
  moreover
  from topSpaceAssum have 0∈T⋃T∈T using IsATopology˙def empty˙open by auto
  ultimately show T=–0,⋃T″ by auto
qed

lemma indiscrete˙spectrum:
  shows (A –is in the spectrum of″(λT. T=–0,⋃T″)) ⟷ A≲1
proof
  assume (A –is in the spectrum of″(λT. T=–0,⋃T″))
  then have reg:∀T. ((T–is a topology″ ∧ ⋃T≈A) ⟶ T =–0,⋃T″) using Spec˙def
by auto
  moreover
  have ⋃Pow(A)=A by auto
  then have ⋃Pow(A)≈A by auto
  moreover
  have Pow(A) –is a topology″ using Pow˙is˙top by auto
  ultimately have P:Pow(A)=–0,A″ by auto
  {
    assume A≠0
    then obtain x where x∈A by blast
    then have –x″∈Pow(A) by auto
    with P have –x″=A by auto
    then have A≈1 using singleton˙eqpoll˙1 by auto
    then have A≲1 using eqpoll˙imp˙lepoll by auto
  }
  moreover
  {
    assume A=0
    then have A≈0 by auto
    then have A≲1 using empty˙lepollI eq˙lepoll˙trans by auto
  }
  ultimately show A≲1 by auto
next
  assume A≲1
  {
    fix T
    assume T–is a topology″⋃T≈A
    {
      assume A=0
      with ‹⋃T≈A› have ⋃T≈0 by auto
      then have ⋃T=0 using eqpoll˙0˙is˙0 by auto
      then have T⊆–0″ by auto
      with ‹T–is a topology″› have T=–0″ using empty˙open by auto
      then have T=–0,⋃T″ by auto
    }
    moreover
    {
      assume A≠0
      then obtain E where E∈A by blast
```

```
      with ‹A≲1› have A=–E″ using lepoll˙1˙is˙sing by auto
      then have A≈1 using singleton˙eqpoll˙1 by auto
      with ‹⋃T≈A› have NONempty:⋃T≈1 using eqpoll˙trans by blast
      then have ⋃T≲1 using eqpoll˙imp˙lepoll by auto
      moreover
      {
        assume ⋃T=0
        then have 0≈⋃T by auto
        with NONempty have 0≈1 using eqpoll˙trans by blast
        then have 0=1 using eqpoll˙0˙is˙0 eqpoll˙sym by auto
        then have False by auto
      }
      then have ⋃T≠0 by auto
      then obtain R where R∈⋃T by blast
      ultimately have ⋃T=–R″ using lepoll˙1˙is˙sing by auto
      moreover
      have T⊆Pow(⋃T) by auto
      ultimately have T⊆Pow(–R″) by auto
      then have T⊆–0,–R″″ by blast
      moreover
      with ‹T–is a topology″› have 0∈T⋃T∈T using IsATopology˙def by auto
      moreover
      note ‹⋃T=–R″›
      ultimately have T=–0,⋃T″ by auto
    }
    ultimately have T=–0,⋃T″ by auto
  }
  then show A –is in the spectrum of″(λT. T=–0,⋃T″) using Spec˙def by auto
qed

theorem (in topology0) anti˙indiscrete:
  shows (T–is anti-″(λT. T=–0,⋃T″)) ⟷ T–is T0″
proof
  assume T–is T0″
  {
    fix A
    assume A∈Pow(⋃T)T–restricted to″A=–0,⋃(T–restricted to″A)″
   then have un:⋃(T–restricted to″A)=A T–restricted to″A=–0,A″ using Restrict-
edTo˙def by auto
    from ‹T–is T0″›‹A∈Pow(⋃T)› have (T–restricted to″A)–is T0″ using T0˙here
by auto
    {
      assume A=0
      then have A≈0 by auto
      then have A≲1 using empty˙lepollI eq˙lepoll˙trans by auto
    }
    moreover
    {
      assume A≠0
```

then obtain E where E∈A by blast
{
fix y
assume y∈Ay≠E
with ‹E∈A› un have y∈⋃(T–restricted to″A)E∈⋃(T–restricted to″A) by auto
with ‹(T–restricted to″A)–is T_0″›‹y≠E› have ∃U∈(T–restricted to″A). (E∈U∧y∉U)∨(E∉U∧y∈U)
unfolding isT0˙def by blast
then obtain U where U∈(T–restricted to″A) (E∈U∧y∉U)∨(E∉U∧y∈U) by auto
with ‹T–restricted to″A=–0,A″› have U=0∨U=A by auto
with ‹(E∈U∧y∉U)∨(E∉U∧y∈U)›‹y∈A›‹E∈A› have False by auto
}
then have ∀y∈A. y=E by auto
with ‹E∈A› have A=–E″ by blast
then have A≈1 using singleton˙eqpoll˙1 by auto
then have A≲1 using eqpoll˙imp˙lepoll by auto
}
ultimately have A≲1 by auto
then have A–is in the spectrum of″(λT. T=–0,⋃T″) using indiscrete˙spectrum by auto
}
then show T–is anti-″(λT. T=–0,⋃T″) unfolding antiProperty˙def by auto
next
assume T–is anti-″(λT. T=–0,⋃T″)
then have ∀A∈Pow(⋃T). (T–restricted to″A)=–0,⋃(T–restricted to″A)″ ⟶ (A –is in the spectrum of″ (λT. T=–0,⋃T″)) using antiProperty˙def by auto
then have ∀A∈Pow(⋃T). (T–restricted to″A)=–0,⋃(T–restricted to″A)″ ⟶ A≲1 using indiscrete˙spectrum by auto
moreover
have ∀A∈Pow(⋃T). ⋃(T–restricted to″A)=A unfolding RestrictedTo˙def by auto
ultimately have reg:∀A∈Pow(⋃T). (T–restricted to″A)=–0,A″ ⟶ A≲1 by auto
{
fix x y
assume x∈⋃Ty∈⋃Tx≠y
{
assume ∀U∈T. (x∈U∧y∈U)∨(x∉U∧y∉U)
then have T–restricted to″–x,y″⊆–0,–x,y″″ unfolding RestrictedTo˙def by auto
moreover
from ‹x∈⋃T›‹y∈⋃T› have emp:0∈T–x,y″∩0=0 and tot: –x,y″=–x,y″∩⋃T ⋃T∈T using topSpaceAssum empty˙open IsATopology˙def by auto
from emp have 0∈T–restricted to″–x,y″ unfolding RestrictedTo˙def by auto
moreover
from tot have –x,y″∈T–restricted to″–x,y″ unfolding RestrictedTo˙def by auto
ultimately have T–restricted to″–x,y″=–0,–x,y″″ by auto
with reg ‹x∈⋃T›‹y∈⋃T› have –x,y″≲1 by auto
moreover
have x∈–x,y″ by auto

```
    ultimately have –x,y″=–x″ using lepoll˙1˙is˙sing[of –x,y″x] by auto
    moreover
    have y∈–x,y″ by auto
    ultimately have y∈–x″ by auto
    then have y=x by auto
    then have False using ‹x≠y› by auto
  }
  then have ∃U∈T. (x∉U∨y∉U)∧(x∈U∨y∈U) by auto
  then have ∃U∈T. (x∈U∧y∉U)∨(x∉U∧y∈U) by auto
 }
 then have ∀x y. x∈⋃T∧y∈⋃T∧ x≠y ⟶ (∃U∈T. (x∈U∧y∉U)∨(y∈U∧x∉U)) by
auto
 then show T –is T0″ using isT0˙def by auto
qed
```

The conclusion is that being T_0 is just the opposite to being indiscrete.

Next, let's compute the anti-T_i for $i = 1,\ 2,\ 3$ or 4. Surprisingly, they are all the same. Meaning, that the total negation of T_1 is enough to negate all of these axioms.

```
theorem anti˙T1:
 shows (T–is anti-″isT1) ⟷ (IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″))
proof
 assume T–is anti-″isT1
 let r=–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″
 have antisym(r) unfolding antisym˙def by auto
 moreover
 have trans(r) unfolding trans˙def by auto
 moreover
 {
  fix A B
  assume A∈TB∈T
  {
   assume ¬(A⊆B∨B⊆A)
   then have A-B≠0B-A≠0 by auto
   then obtain x y where x∈Ax∉By∈By∉A x≠y by blast
   then have –x,y″∩A=–x″–x,y″∩B=–y″ by auto
   moreover
   from ‹A∈T›‹B∈T› have –x,y″∩A∈T–restricted to″–x,y″–x,y″∩B∈T–restricted
to″–x,y″ unfolding
    RestrictedTo˙def by auto
   ultimately have open˙set:–x″∈T–restricted to″–x,y″–y″∈T–restricted to″–x,y″
by auto
   have x∈⋃Ty∈⋃T using ‹A∈T›‹B∈T›‹x∈A›‹y∈B› by auto
   then have sub:–x,y″∈Pow(⋃T) by auto
   then have tot:⋃(T–restricted to″–x,y″)=–x,y″ unfolding RestrictedTo˙def by
auto
   {
    fix s t
```

assume s∈⋃(T–restricted to″–x,y″)t∈⋃(T–restricted to″–x,y″)s≠t
with tot have s∈–x,y″t∈–x,y″s≠t by auto
then have (s=x∧t=y)∨(s=y∧t=x) by auto
with open˙set have ∃U∈(T–restricted to″–x,y″). s∈U∧t∉U using ‹x≠y› by auto
}
then have (T–restricted to″–x,y″)–is T_1″ unfolding isT1˙def by auto
with sub ‹T–is anti-″isT1› tot have –x,y″ –is in the spectrum of″isT1 using antiProperty˙def
by auto
then have –x,y″≲1 using T1˙spectrum by auto
moreover
have x∈–x,y″ by auto
ultimately have –x″=–x,y″ using lepoll˙1˙is˙sing[of –x,y″x] by auto
moreover
have y∈–x,y″ by auto
ultimately
have y∈–x″ by auto
then have x=y by auto
then have False using ‹x∈A›‹y∉A› by auto
}
then have A⊆B∨B⊆A by auto
}
then have r –is total on″T using IsTotal˙def by auto
ultimately
show IsLinOrder(T,r) using IsLinOrder˙def by auto
next
assume IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″)
then have ordTot:∀S∈T. ∀B∈T. S⊆B∨B⊆S unfolding IsLinOrder˙def IsTotal˙def by auto
{
fix A
assume A∈Pow(⋃T) and T1:(T–restricted to″A) –is T_1″
then have tot:⋃(T–restricted to″A)=A unfolding RestrictedTo˙def by auto
{
fix U V
assume U∈T–restricted to″AV∈T–restricted to″A
then obtain AU AV where AU∈TAV∈TU=A∩AUV=A∩AV unfolding RestrictedTo˙def by auto
with ordTot have U⊆V∨V⊆U by auto
}
then have ordTotSub:∀S∈T–restricted to″A. ∀B∈T–restricted to″A. S⊆B∨B⊆S by auto
{
assume A=0
then have A≈0 by auto
moreover
have 0≲1 using empty˙lepollI by auto
ultimately have A≲1 using eq˙lepoll˙trans by auto

```
      then have A–is in the spectrum of″isT1 using T1˙spectrum by auto
    }
    moreover
    {
      assume A≠0
      then obtain t where t∈A by blast
      {
        fix y
        assume y∈Ay≠t
        with ‹t∈A› tot T1 obtain U where U∈(T–restricted to″A)y∈Ut∉U unfolding
isT1˙def
          by auto
        from ‹y≠t› have t≠y by auto
          with ‹y∈A›‹t∈A› tot T1 obtain V where V∈(T–restricted to″A)t∈Vy∉V
unfolding isT1˙def
          by auto
        with ‹y∈U›‹t∉U› have ¬(U⊆V∨V⊆U) by auto
        with ordTotSub ‹U∈(T–restricted to″A)›‹V∈(T–restricted to″A)› have False
by auto
      }
      then have ∀y∈A. y=t by auto
      with ‹t∈A› have A=–t″ by blast
      then have A≈1 using singleton˙eqpoll˙1 by auto
      then have A≲1 using eqpoll˙imp˙lepoll by auto
      then have A–is in the spectrum of″isT1 using T1˙spectrum by auto
    }
    ultimately
    have A–is in the spectrum of″isT1 by auto
  }
  then show T–is anti-″isT1 using antiProperty˙def by auto
qed

corollary linordtop˙here:
  shows (λT. IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″))–is hereditary″
  using anti˙T1 anti˙here[of isT1] by auto

theorem (in topology0) anti˙T4:
 shows (T–is anti-″isT4) ⟷ (IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″))
proof
  assume T–is anti-″isT4
  let r=–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″
  have antisym(r) unfolding antisym˙def by auto
  moreover
  have trans(r) unfolding trans˙def by auto
  moreover
  {
    fix A B
    assume A∈TB∈T
    {
```

assume ¬(A⊆B∨B⊆A)
then have A-B≠0B-A≠0 by auto
then obtain x y where x∈Ax∉By∈By∉A x≠y by blast
then have –x,y″∩A=–x″–x,y″∩B=–y″ by auto
moreover
from ‹A∈T›‹B∈T› have –x,y″∩A∈T–restricted to″–x,y″–x,y″∩B∈T–restricted to″–x,y″ unfolding
RestrictedTo˙def by auto
ultimately have open˙set:–x″∈T–restricted to″–x,y″–y″∈T–restricted to″–x,y″ by auto
have x∈⋃Ty∈⋃T using ‹A∈T›‹B∈T›‹x∈A›‹y∈B› by auto
then have sub:–x,y″∈Pow(⋃T) by auto
then have tot:⋃(T–restricted to″–x,y″)=–x,y″ unfolding RestrictedTo˙def by auto
{
fix s t
assume s∈⋃(T–restricted to″–x,y″)t∈⋃(T–restricted to″–x,y″)s≠t
with tot have s∈–x,y″t∈–x,y″s≠t by auto
then have (s=x∧t=y)∨(s=y∧t=x) by auto
with open˙set have ∃U∈(T–restricted to″–x,y″). s∈U∧t∉U using ‹x≠y› by auto
}
then have (T–restricted to″–x,y″)–is T_1″ unfolding isT1˙def by auto
moreover
{
fix s
assume AS:s–is closed in″(T–restricted to″–x,y″)
{
fix t
assume AS2:t–is closed in″(T–restricted to″–x,y″)s∩t=0
have (T–restricted to″–x,y″)–is a topology″ using Top˙1˙L4 by auto
with tot have 0∈(T–restricted to″–x,y″)–x,y″∈(T–restricted to″–x,y″) using empty˙open
union˙open[where $\mathcal{A}$=T–restricted to″–x,y″] by auto
moreover
note open˙set
moreover
have T–restricted to″–x,y″⊆Pow(⋃(T–restricted to″–x,y″)) by blast
with tot have T–restricted to″–x,y″⊆Pow(–x,y″) by auto
ultimately have T–restricted to″–x,y″=–0,–x″,–y″,–x,y″″ by blast
moreover have –0,–x″,–y″,–x,y″″=Pow(–x,y″) by blast
ultimately have P:T–restricted to″–x,y″=Pow(–x,y″) by simp
with tot have –A∈Pow(–x,y″). A–is closed in″(T–restricted to″–x,y″)″=–A ∈ Pow(–x, y″) . A ⊆ –x, y″ ∧ –x, y″ - A ∈ Pow(–x, y″)″ using IsClosed˙def by simp
with P have S:–A∈Pow(–x,y″). A–is closed in″(T–restricted to″–x,y″)″=T–restricted to″–x,y″ by auto
from AS AS2(1) have s∈Pow(–x,y″) t∈Pow(–x,y″) using IsClosed˙def tot by auto

moreover
note AS2(1) AS
ultimately have s∈–A∈Pow(–x,y″). A–is closed in″(T–restricted to″–x,y″)″t∈–A∈Pow(–x,y″). A–is closed in″(T–restricted to″–x,y″)″
by auto
with S AS2(2) have s∈T–restricted to″–x,y″ t∈T–restricted to″–x,y″s∩t=0 by auto
then have ∃U∈(T–restricted to″–x,y″). ∃V∈(T–restricted to″–x,y″). s⊆U∧t⊆V∧U∩V=0 by auto
}
then have ∀t. t–is closed in″(T–restricted to″–x,y″)∧s∩t=0 ⟶ (∃U∈(T–restricted to″–x,y″). ∃V∈(T–restricted to″–x,y″). s⊆U∧t⊆V∧U∩V=0)
by auto
}
then have ∀s. s–is closed in″(T–restricted to″–x,y″) ⟶ (∀t. t–is closed in″(T–restricted to″–x,y″)∧s∩t=0 ⟶ (∃U∈(T–restricted to″–x,y″). ∃V∈(T–restricted to″–x,y″). s⊆U∧t⊆V∧U∩V=0))
by auto
then have (T–restricted to″–x,y″)–is normal″ using IsNormal˙def by auto
ultimately have (T–restricted to″–x,y″)–is T_4″ using isT4˙def by auto
with sub ‹T–is anti-″isT4› tot have –x,y″ –is in the spectrum of″isT4 using antiProperty˙def
by auto
then have –x,y″≲1 using T4˙spectrum by auto
moreover
have x∈–x,y″ by auto
ultimately have –x″=–x,y″ using lepoll˙1˙is˙sing[of –x,y″x] by auto
moreover
have y∈–x,y″ by auto
ultimately
have y∈–x″ by auto
then have x=y by auto
then have False using ‹x∈A›‹y∉A› by auto
}
then have A⊆B∨B⊆A by auto
}
then have r –is total on″T using IsTotal˙def by auto
ultimately
show IsLinOrder(T,r) using IsLinOrder˙def by auto
next
assume IsLinOrder(T, –⟨U,V⟩ ∈ Pow(⋃T) × Pow(⋃T) . U ⊆ V″)
then have T–is anti-″isT1 using anti˙T1 by auto
moreover
have ∀T. T–is a topology″ ⟶ (T–is T_4″) ⟶ (T–is T_1″) using topology0.T4˙is˙T3

topology0.T3˙is˙T2 T2˙is˙T1 topology0˙def by auto
moreover
have ∀A. (A –is in the spectrum of″ isT1) ⟶ (A –is in the spectrum of″ isT4) using T1˙spectrum T4˙spectrum

by auto
ultimately show T–is anti-″isT4 using eq˙spect˙rev˙imp˙anti[of isT4isT1] by auto
qed

theorem (in topology0) anti˙T3:
shows (T–is anti-″isT3) ⟷ (IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″))
proof
assume T–is anti-″isT3
moreover
have ∀T. T–is a topology″ ⟶ (T–is T_4″) ⟶ (T–is T_3″) using topology0.T4˙is˙T3

topology0˙def by auto
moreover
have ∀A. (A –is in the spectrum of″ isT3) ⟶ (A –is in the spectrum of″ isT4)
using T3˙spectrum T4˙spectrum
by auto
ultimately have T–is anti-″isT4 using eq˙spect˙rev˙imp˙anti[of isT4isT3] by auto
then show IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″) using anti˙T4 by
auto
next
assume IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″)
then have T–is anti-″isT1 using anti˙T1 by auto
moreover
have ∀T. T–is a topology″ ⟶ (T–is T_3″) ⟶ (T–is T_1″) using
topology0.T3˙is˙T2 T2˙is˙T1 topology0˙def by auto
moreover
have ∀A. (A –is in the spectrum of″ isT1) ⟶ (A –is in the spectrum of″ isT3)
using T1˙spectrum T3˙spectrum
by auto
ultimately show T–is anti-″isT3 using eq˙spect˙rev˙imp˙anti[of isT3isT1] by auto
qed

theorem (in topology0) anti˙T2:
shows (T–is anti-″isT2) ⟷ (IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″))
proof
assume T–is anti-″isT2
moreover
have ∀T. T–is a topology″ ⟶ (T–is T_4″) ⟶ (T–is T_2″) using topology0.T4˙is˙T3

topology0.T3˙is˙T2 topology0˙def by auto
moreover
have ∀A. (A –is in the spectrum of″ isT2) ⟶ (A –is in the spectrum of″ isT4)
using T2˙spectrum T4˙spectrum
by auto
ultimately have T–is anti-″isT4 using eq˙spect˙rev˙imp˙anti[of isT4isT2] by auto
then show IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″) using anti˙T4 by
auto
next
assume IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″)

then have T–is anti-″isT1 using anti˙T1 by auto
moreover
have ∀T. T–is a topology″ ⟶ (T–is T_2″) ⟶ (T–is T_1″) using T2˙is˙T1 by auto
moreover
have ∀A. (A –is in the spectrum of″ isT1) ⟶ (A –is in the spectrum of″ isT2)
using T1˙spectrum T2˙spectrum
by auto
ultimately show T–is anti-″isT2 using eq˙spect˙rev˙imp˙anti[of isT2isT1] by auto
qed

lemma linord˙spectrum:
shows (A–is in the spectrum of″(λT. IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V″))) ⟷ A≲1
proof
assume A–is in the spectrum of″(λT. IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V″))
then have reg:∀T. T–is a topology″∧⋃T≈A ⟶ IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V″)
using Spec˙def by auto
{
assume A=0
moreover
have 0≲1 using empty˙lepollI by auto
ultimately have A≲1 using eq˙lepoll˙trans by auto
}
moreover
{
assume A≠0
then obtain x where x∈A by blast
moreover
{
fix y
assume y∈A
have Pow(A) –is a topology″ using Pow˙is˙top by auto
moreover
have ⋃Pow(A)=A by auto
then have ⋃Pow(A)≈A by auto
note reg
ultimately have IsLinOrder(Pow(A),–⟨U,V⟩∈Pow(⋃Pow(A))×Pow(⋃Pow(A)).
U⊆V″) by auto
then have IsLinOrder(Pow(A),–⟨U,V⟩∈Pow(A)×Pow(A). U⊆V″) by auto
with ‹x∈A›‹y∈A› have –x″⊆–y″∨–y″⊆–x″ unfolding IsLinOrder˙def IsTotal˙def
by auto
then have x=y by auto
}
ultimately have A=–x″ by blast
then have A≈1 using singleton˙eqpoll˙1 by auto
then have A≲1 using eqpoll˙imp˙lepoll by auto
}

ultimately show A≲1 by auto
next
assume A≲1
then have ind:A–is in the spectrum of"(λT. T=–0,⋃T") using indiscrete˙spectrum by auto
{
fix T
assume AS:T–is a topology" T=–0,⋃T"
have trans(–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V") unfolding trans˙def by auto
moreover
have antisym(–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V") unfolding antisym˙def by auto
moreover
have –⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V"–is total on"T
proof-
{
fix aa b
assume aa∈Tb∈T
with AS(2) have aa∈–0,⋃T"b∈–0,⋃T" by auto
then have aa=0∨aa=⋃Tb=0∨b=⋃T by auto
then have aa⊆b∨b⊆aa by auto
then have ⟨aa, b⟩ ∈ Collect(Pow(⋃T) × Pow(⋃T), split((⊆))) ∨ ⟨b, aa⟩ ∈ Collect(Pow(⋃T) × Pow(⋃T), split((⊆)))
using ‹aa∈T›‹b∈T› by auto
}
then show thesis using IsTotal˙def by auto
qed
ultimately have IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V") unfolding IsLinOrder˙def by auto
}
then have ∀T. T –is a topology" ⟶ T = –0, ⋃T" ⟶ IsLinOrder(T, –⟨U,V⟩ ∈ Pow(⋃T) × Pow(⋃T) . U ⊆ V") by auto
then show A–is in the spectrum of"(λT. IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V"))
using P˙imp˙Q˙spec˙inv[of λT. T=–0,⋃T"λT. IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V")]
ind by auto
qed

theorem (in topology0) anti˙linord:
shows (T–is anti-"(λT. IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V"))) ⟷ T–is T_1"
proof
assume AS:T–is anti-"(λT. IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V"))
{
assume ¬(T–is T_1")
then obtain x y where x∈⋃Ty∈⋃Tx≠y∀U∈T. x∉U∨y∈U unfolding isT1˙def by auto
{

assume –x″∈T–restricted to″–x,y″
then obtain U where U∈T –x″=–x,y″∩U unfolding RestrictedTo˙def by auto
moreover
have x∈–x″ by auto
ultimately have U∈Tx∈U by auto
moreover
{
assume y∈U
then have y∈–x,y″∩U by auto
with ‹–x″=–x,y″∩U› have y∈–x″ by auto
with ‹x≠y› have False by auto
}
then have y∉U by auto
moreover
note ‹∀ U∈T. x∉U∨y∈U›
ultimately have False by auto
}
then have –x″∉T–restricted to″–x,y″ by auto
moreover
have tot:⋃(T–restricted to″–x,y″)=–x,y″ using ‹x∈⋃T›‹y∈⋃T› unfolding RestrictedTo˙def by auto
moreover
have T–restricted to″–x,y″⊆Pow(⋃(T–restricted to″–x,y″)) by auto
ultimately have T–restricted to″–x,y″⊆Pow(–x,y″)-—x″″ by auto
moreover
have Pow(–x,y″)=–0,–x,y″,–x″,–y″″ by blast
ultimately have T–restricted to″–x,y″⊆–0,–x,y″,–y″″ by auto
moreover
have IsLinOrder(–0,–x,y″,–y″″,–⟨U,V⟩∈Pow(–x,y″)×Pow(–x,y″). U⊆V″)
proof-
have antisym(Collect(Pow(–x, y″) × Pow(–x, y″), split((⊆)))) using antisym˙def by auto
moreover
have trans(Collect(Pow(–x, y″) × Pow(–x, y″), split((⊆)))) using trans˙def by auto
moreover
have Collect(Pow(–x, y″) × Pow(–x, y″), split((⊆))) –is total on″ –0, –x, y″, –y″″ using IsTotal˙def by auto
ultimately show IsLinOrder(–0,–x,y″,–y″″,–⟨U,V⟩∈Pow(–x,y″)×Pow(–x,y″). U⊆V″) using IsLinOrder˙def by auto
qed
ultimately have IsLinOrder(T–restricted to″–x,y″,–⟨U,V⟩∈Pow(–x,y″)×Pow(–x,y″). U⊆V″) using ord˙linear˙subset
by auto
with tot have IsLinOrder(T–restricted to″–x,y″,–⟨U,V⟩∈Pow(⋃(T–restricted to″–x,y″))×Pow(⋃(T–restricted to″–x,y″)). U⊆V″)
by auto
then have IsLinOrder(T–restricted to″–x,y″,Collect(Pow(⋃(T –restricted to″ –x,y″)) × Pow(⋃(T –restricted to″ –x,y″)), split((⊆)))) by auto

moreover
from ‹x∈⋃T›‹y∈⋃T› have –x,y″∈Pow(⋃T) by auto
moreover
note AS
ultimately have –x,y″–is in the spectrum of″(λT. IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″)) unfolding antiProperty˙def
by simp
then have –x,y″≲1 using linord˙spectrum by auto
moreover
have x∈–x,y″ by auto
ultimately have –x″=–x,y″ using lepoll˙1˙is˙sing[of –x,y″x] by auto
moreover
have y∈–x,y″ by auto
ultimately
have y∈–x″ by auto
then have x=y by auto
then have False using ‹x≠y› by auto
}
then show T –is T_1″ by auto
next
assume T1:T –is T_1″
{
fix A
assume A˙def:A∈Pow(⋃T)IsLinOrder((T–restricted to″A) ,–⟨U,V⟩∈Pow(⋃(T–restricted to″A))×Pow(⋃(T–restricted to″A)). U⊆V″)
{
fix x
assume AS1:x∈A
{
fix y
assume AS:y∈Ax≠y
with AS1 have –x,y″∈Pow(⋃T) using ‹A∈Pow(⋃T)› by auto
from ‹x∈A›‹y∈A› have –x,y″∈Pow(A) by auto
from ‹–x,y″∈Pow(⋃T)› have T11:(T–restricted to″–x,y″)–is T_1″ using T1˙here T1 by auto
moreover
have tot:⋃(T–restricted to″–x,y″)=–x,y″ unfolding RestrictedTo˙def using ‹–x,y″∈Pow(⋃T)› by auto
moreover
note AS(2)
ultimately obtain U where x∈Uy∉UU∈(T–restricted to″–x,y″) unfolding isT1˙def by auto
moreover
from AS(2) tot T11 obtain V where y∈Vx∉VV∈(T–restricted to″–x,y″) unfolding isT1˙def by auto
ultimately have x∈U-Vy∈V-UU∈(T–restricted to″–x,y″)V∈(T–restricted to″–x,y″) by auto
then have ¬(U⊆V∨V⊆U)U∈(T–restricted to″–x,y″)V∈(T–restricted to″–x,y″) by auto

then have ¬(–⟨U,V⟩∈Pow(⋃(T–restricted to″–x,y″))×Pow(⋃(T–restricted to″–x,y″)). U⊆V″ –is total on″ (T–restricted to″–x,y″))
unfolding IsTotal˙def by auto
then have ¬(IsLinOrder((T–restricted to″–x,y″),–⟨U,V⟩∈Pow(⋃(T–restricted to″–x,y″))×Pow(⋃(T–restricted to″–x,y″)). U⊆V″))
unfolding IsLinOrder˙def by auto
moreover
{
have (T–restricted to″A) –is a topology″ using Top˙1˙L4 by auto
moreover
note A˙def(2) linordtop˙here
ultimately have ∀B∈Pow(⋃(T–restricted to″A)). IsLinOrder((T–restricted to″A)–restricted to″B ,–⟨U,V⟩∈Pow(⋃((T–restricted to″A)–restricted to″B))×Pow(⋃((T–restricted to″A)–restricted to″B)). U⊆V″)
unfolding IsHer˙def by auto
moreover
have tot:⋃(T–restricted to″A)=A unfolding RestrictedTo˙def using ‹A∈Pow(⋃T)› by auto
ultimately have ∀B∈Pow(A). IsLinOrder((T–restricted to″A)–restricted to″B ,–⟨U,V⟩∈Pow(⋃((T–restricted to″A)–restricted to″B))×Pow(⋃((T–restricted to″A)–restricted to″B)). U⊆V″) by auto
moreover
have ∀B∈Pow(A). (T–restricted to″A)–restricted to″B=T–restricted to″B using subspace˙of˙subspace ‹A∈Pow(⋃T)› by auto
ultimately
have ∀B∈Pow(A). IsLinOrder((T–restricted to″B) ,–⟨U,V⟩∈Pow(⋃((T–restricted to″A)–restricted to″B))×Pow(⋃((T–restricted to″A)–restricted to″B)). U⊆V″) by auto
moreover
have ∀B∈Pow(A). ⋃((T–restricted to″A)–restricted to″B)=B using ‹A∈Pow(⋃T)› unfolding RestrictedTo˙def by auto
ultimately have ∀B∈Pow(A). IsLinOrder((T–restricted to″B) ,–⟨U,V⟩∈Pow(B)×Pow(B). U⊆V″) by auto
with ‹–x,y″∈Pow(A)› have IsLinOrder((T–restricted to″–x,y″) ,–⟨U,V⟩∈Pow(–x,y″)×Pow(–x,y″). U⊆V″) by auto
}
ultimately have False using tot by auto
}
then have A=–x″ using AS1 by auto
then have A≈1 using singleton˙eqpoll˙1 by auto
then have A≲1 using eqpoll˙imp˙lepoll by auto
then have A–is in the spectrum of″(λT. IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″)) using linord˙spectrum
by auto
}
moreover
{
assume A=0
then have A≈0 by auto

moreover
have 0≲1 using empty˙lepollI by auto
ultimately have A≲1 using eq˙lepoll˙trans by auto
then have A–is in the spectrum of″(λT. IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″)) using linord˙spectrum
by auto
}
ultimately have A–is in the spectrum of″(λT. IsLinOrder(T,–⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V″)) by blast
}
then show T–is anti-″(λT. IsLinOrder(T, –⟨U,V⟩ ∈ Pow(⋃T) × Pow(⋃T) . U ⊆ V″)) unfolding antiProperty˙def
by auto
qed

In conclusion, T_1 is also an anti-property.

Let's define some anti-properties that we'll use in the future.

definition
IsAntiComp (˙–is anti-compact″)
where T–is anti-compact″ ≡ T–is anti-″(λT. (⋃T)–is compact in″T)

definition
IsAntiLin (˙–is anti-lindeloef″)
where T–is anti-lindeloef″ ≡ T–is anti-″(λT. ((⋃T)–is lindeloef in″T))

Anti-compact spaces are also called pseudo-finite spaces in literature before the concept of anti-property was defined.

end

61 Topology 6

theory Topology˙ZF˙6 imports Topology˙ZF˙4 Topology˙ZF˙2 Topology˙ZF˙1

begin

This theory deals with the relations between continuous functions and convergence of filters. At the end of the file there some results about the building of functions in cartesian products.

61.1 Image filter

First of all, we will define the appropriate tools to work with functions and filters together.

We define the image filter as the collections of supersets of of images of sets from a filter.

```
definition
  ImageFilter (˙[˙]..˙ 98)
   where 𝔉 –is a filter on″ X ⟹ f:X→Y ⟹ f[𝔉]..Y ≡ –A∈Pow(Y). ∃D∈–f(B)
.B∈𝔉″. D⊆A″
```

Note that in the previous definition, it is necessary to state Y as the final set because f is also a function to every superset of its range. X can be changed by domain(f) without any change in the definition.

```
lemma base˙image˙filter:
  assumes 𝔉 –is a filter on″ X f:X→Y
  shows –fB .B∈𝔉″ –is a base filter″ (f[𝔉]..Y) and (f[𝔉]..Y) –is a filter on″ Y
proof-
  {
    assume 0 ∈ –fB .B∈𝔉″
    then obtain B where B∈𝔉 and f˙B:fB=0 by auto
    then have B∈Pow(X) using assms(1) IsFilter˙def by auto
    then have fB=–fb. b∈B″ using image˙fun assms(2) by auto
    with f˙B have –fb. b∈B″=0 by auto
    then have B=0 by auto
    with ‹B∈𝔉› have False using IsFilter˙def assms(1) by auto
  }
  then have 0∉–fB .B∈𝔉″ by auto
  moreover
  from assms(1) obtain S where S∈𝔉 using IsFilter˙def by auto
  then have fS∈–fB .B∈𝔉″ by auto
  then have nA:–fB .B∈𝔉″≠0 by auto
  moreover
  {
    fix A B
    assume A∈–fB .B∈𝔉″ and B∈–fB .B∈𝔉″
    then obtain AB BB where A=fAB B=fBB AB∈𝔉 BB∈𝔉 by auto
    then have A∩B=(fAB)∩(fBB) by auto
    then have I: f(AB∩BB)⊆A∩B by auto
    moreover
    from assms(1) I ‹AB∈𝔉›‹BB∈𝔉› have AB∩BB∈𝔉 using IsFilter˙def by auto
    ultimately have ∃D∈–fB .B∈𝔉″. D⊆A∩B by auto
  }
  then have ∀A∈–fB .B∈𝔉″. ∀B∈–fB .B∈𝔉″. ∃D∈–fB .B∈𝔉″. D⊆A∩B by auto
  ultimately have sbc:–fB .B∈𝔉″ –satisfies the filter base condition″
    using SatisfiesFilterBase˙def by auto
  moreover
  {
    fix t
    assume t∈–fB . B∈𝔉″
    then obtain B where B∈𝔉 and im˙def:fB=t by auto
    with assms(1) have B∈Pow(X) unfolding IsFilter˙def by auto
    with im˙def assms(2) have t=–fx. x∈B″ using image˙fun by auto
    with assms(2) ‹B∈Pow(X)› have t⊆Y using apply˙funtype by auto
    }
```

then have nB:–fB . B∈𝔉″⊆Pow(Y) by auto
ultimately
have ((–fB .B∈𝔉″ –is a base filter″ –A ∈ Pow(Y) . ∃D∈–fB .B∈𝔉″. D ⊆ A″) ∧ (⋃–A ∈ Pow(Y) . ∃D∈–fB .B∈𝔉″. D ⊆ A″=Y)) using base˙unique˙filter˙set2
by force
then have –fB .B∈𝔉″ –is a base filter″ –A ∈ Pow(Y) . ∃D∈–fB .B∈𝔉″. D ⊆ A″ by auto
with assms show –fB .B∈𝔉″ –is a base filter″ (f[𝔉]..Y) using ImageFilter˙def by auto
moreover
note sbc
moreover
{
from nA obtain D where I: D∈–fB .B∈𝔉″ by blast
moreover from I nB have D⊆Y by auto
ultimately have Y∈–A∈Pow(Y). ∃D∈–fB .B∈𝔉″. D⊆A″ by auto
}
then have ⋃–A∈Pow(Y). ∃D∈–fB .B∈𝔉″. D⊆A″=Y by auto
ultimately show (f[𝔉]..Y) –is a filter on″ Y using basic˙filter
ImageFilter˙def assms by auto
qed

61.2 Continuous at a point vs. globally continuous

In this section we show that continuity of a function implies local continuity (at a point) and that local continuity at all points implies (global) continuity.

If a function is continuous, then it is continuous at every point.

lemma cont˙global˙imp˙continuous˙x:
assumes x∈⋃τ_1 IsContinuous(τ_1,τ_2,f) f:(⋃τ_1)→(⋃τ_2) x∈⋃τ_1
shows ∀U∈τ_2. f(x)∈U ⟶ (∃V∈τ_1. x∈V ∧ f(V)⊆U)
proof-
{
fix U
assume AS:U∈τ_2 f(x)∈U
then have f-(U)∈τ_1 using assms(2) IsContinuous˙def by auto
moreover
from assms(3) have f(f-(U))⊆U using function˙image˙vimage fun˙is˙fun
by auto
moreover
from assms(3) assms(4) AS(2) have x∈f-(U) using func1˙1˙L15 by auto
ultimately have ∃V∈τ_1. x∈V ∧ fV⊆U by auto
}
then show ∀U∈τ_2. f(x)∈U ⟶ (∃V∈τ_1. x∈V ∧ f(V)⊆U) by auto
qed

A function that is continuous at every point of its domain is continuous.

lemma ccontinuous˙all˙x˙imp˙cont˙global:

```
  assumes ∀x∈⋃τ1. ∀U∈τ2. fx∈U ⟶ (∃V∈τ1. x∈V ∧ fV⊆U) f∈(⋃τ1)→(⋃τ2)
and
    τ1 –is a topology″
  shows IsContinuous(τ1,τ2,f)
proof-
  {
    fix U
    assume U∈τ2
    {
      fix x
      assume AS: x∈f-U
      note ‹U∈τ2›
      moreover
      from assms(2) have f - U ⊆ ⋃τ1 using func1˙1˙L6A by auto
      with AS have x∈⋃τ1 by auto
      with assms(1) have ∀U∈τ2. fx∈U ⟶ (∃V∈τ1. x∈V ∧ fV⊆U) by auto
      moreover
      from AS assms(2) have fx∈U using func1˙1˙L15 by auto
      ultimately have ∃V∈τ1. x∈V ∧ fV⊆U by auto
      then obtain V where I: V∈τ1 x∈V f(V)⊆U by auto
      moreover
      from I have V⊆⋃τ1 by auto
      moreover
      from assms(2) ‹V⊆⋃τ1› have V⊆f-(fV) using func1˙1˙L9 by auto
      ultimately have V ⊆ f-(U) by blast
      with ‹V∈τ1› ‹x∈V› have ∃V∈τ1. x∈V ∧ V ⊆ f-(U) by auto
    } hence ∀x∈f-(U). ∃V∈τ1. x∈V ∧ V⊆f-(U) by auto
    with assms(3) have f-(U) ∈ τ1 using topology0.open˙neigh˙open topology0˙def
      by auto
  }
  hence ∀U∈τ2. f-U∈τ1 by auto
  then show thesis using IsContinuous˙def by auto
qed
```

61.3 Continuous functions and filters

In this section we consider the relations between filters and continuity.

If the function is continuous then if the filter converges to a point the image filter converges to the image point.

```
lemma (in two˙top˙spaces0) cont˙imp˙filter˙conver˙preserved:
  assumes 𝔉 –is a filter on″ X1 f –is continuous″ 𝔉 →F x –in″ τ1
  shows (f[𝔉]..X2) →F (f(x)) –in″ τ2
proof -
  from assms(1) assms(3) have x∈X1
    using topology0.FilterConverges˙def topol˙cntxs˙valid(1) X1˙def by auto
  have topology0(τ2) using topol˙cntxs˙valid(2) by simp
  moreover from assms(1) have (f[𝔉]..X2) –is a filter on″ (⋃τ2) and –fB .B∈𝔉″ –is
a base filter″ (f[𝔉]..X2)
```

using base˙image˙filter fmapAssum X1˙def X2˙def by auto
moreover have $\forall$ U∈Pow($\bigcup\tau_2$). (fx)∈Interior(U,τ_2) $\longrightarrow$ ($\exists$ D∈–fB .B∈$\mathfrak{F}$″. D⊆U)
proof -
{ fix U
assume U∈Pow(X_2) (fx)∈Interior(U,τ_2)
with ‹x∈X_1› have xim: x∈f-(Interior(U,τ_2)) and sub: f-(Interior(U,τ_2))∈Pow(X_1)

using func1˙1˙L6A fmapAssum func1˙1˙L15 fmapAssum by auto
note sub
moreover
have Interior(U,τ_2)∈τ_2 using topology0.Top˙2˙L2 topol˙cntxs˙valid(2) by auto
with assms(2) have f-(Interior(U,τ_2))∈τ_1 unfolding isContinuous˙def IsContinuous˙def
by auto
with xim have x∈Interior(f-(Interior(U,τ_2)),τ_1)
using topology0.Top˙2˙L3 topol˙cntxs˙valid(1) by auto
moreover from assms(1) assms(3) have –U∈Pow(X_1). x∈Interior(U,τ_1)″⊆$\mathfrak{F}$
using topology0.FilterConverges˙def topol˙cntxs˙valid(1) X1˙def by auto
ultimately have f-(Interior(U,τ_2))∈$\mathfrak{F}$ by auto
moreover have f(f-(Interior(U,τ_2)))⊆Interior(U,τ_2)
using function˙image˙vimage fun˙is˙fun fmapAssum by auto
then have f(f-(Interior(U,τ_2)))⊆U
using topology0.Top˙2˙L1 topol˙cntxs˙valid(2) by auto
ultimately have $\exists$ D∈–f(B) .B∈$\mathfrak{F}$″. D⊆U by auto
} thus thesis by auto
qed
moreover from fmapAssum ‹x∈X_1› have f(x) ∈ X_2
by (rule apply˙funtype)
hence f(x) ∈ $\bigcup\tau_2$ by simp
ultimately show thesis by (rule topology0.convergence˙filter˙base2)
qed

Continuity in filter at every point of the domain implies global continuity.

lemma (in two˙top˙spaces0) filter˙conver˙preserved˙imp˙cont:
assumes $\forall$ x∈$\bigcup\tau_1$. $\forall\mathfrak{F}$. ($\mathfrak{F}$ –is a filter on″ X_1) $\wedge$ ($\mathfrak{F}$ $\rightarrow_F$ x –in″ τ_1)) $\longrightarrow$ ((f[$\mathfrak{F}$]..X_2) $\rightarrow_F$ (fx) –in″ τ_2)
shows f–is continuous″
proof-
{
fix x
assume as2: x∈$\bigcup\tau_1$
with assms have reg:
$\forall\mathfrak{F}$. ($\mathfrak{F}$ –is a filter on″ X_1) $\wedge$ ($\mathfrak{F}$ $\rightarrow_F$ x –in″ τ_1)) $\longrightarrow$ ((f[$\mathfrak{F}$]..X_2) $\rightarrow_F$ (fx) –in″ τ_2)
by auto
let Neig = –U ∈ Pow($\bigcup\tau_1$) . x ∈ Interior(U, τ_1)″
from as2 have NFil: Neig–is a filter on″X_1 and NCon: Neig $\rightarrow_F$ x –in″ τ_1
using topol˙cntxs˙valid(1) topology0.neigh˙filter by auto
{

fix U
assume U∈τ_2 fx∈U
then have U∈Pow(⋃τ_2) fx∈Interior(U,τ_2) using topol˙cntxs˙valid(2) topology0.Top˙2˙L3 by auto
moreover
from NCon NFil reg have (f[Neig]..X_2) $\rightarrow_F$ (fx) –in″ τ_2 by auto
moreover have (f[Neig]..X_2) –is a filter on″ X_2
using base˙image˙filter(2) NFil fmapAssum by auto
ultimately have U∈(f[Neig]..X_2)
using topology0.FilterConverges˙def topol˙cntxs˙valid(2) unfolding X1˙def X2˙def
by auto
moreover
from fmapAssum NFil have –fB .B∈Neig″ –is a base filter″ (f[Neig]..X_2)
using base˙image˙filter(1) X1˙def X2˙def by auto
ultimately have ∃V∈–fB .B∈Neig″. V⊆U using basic˙element˙filter by blast
then obtain B where B∈Neig fB⊆U by auto
moreover
have Interior(B,τ_1)⊆B using topology0.Top˙2˙L1 topol˙cntxs˙valid(1) by auto
hence fInterior(B,τ_1) ⊆ f(B) by auto
moreover have Interior(B,τ_1)∈τ_1
using topology0.Top˙2˙L2 topol˙cntxs˙valid(1) by auto
ultimately have ∃V∈τ_1. x∈V ∧ fV⊆U by force
}
hence ∀U∈τ_2. fx∈U ⟶ (∃V∈τ_1. x∈V ∧ fV⊆U) by auto
}
hence ∀x∈⋃τ_1. ∀U∈τ_2. fx∈U ⟶ (∃V∈τ_1. x∈V ∧ fV⊆U) by auto
then show thesis
using ccontinuous˙all˙x˙imp˙cont˙global fmapAssum X1˙def X2˙def isContinuous˙def tau1˙is˙top
by auto
qed

end

62 Topology 7

theory Topology˙ZF˙7 imports Topology˙ZF˙5
begin

62.1 Connection Properties

Another type of topological properties are the connection properties. These properties establish if the space is formed of several pieces or just one.

A space is connected iff there is no clopen set other that the empty set and the total set.

definition IsConnected (˙–is connected″ 70)

where T –is connected″ ≡ ∀U. (U∈T ∧ (U –is closed in″T)) ⟶ U=0∨U=⋃T

lemma indiscrete˙connected:
shows –0,X″ –is connected″
unfolding IsConnected˙def IsClosed˙def by auto

The anti-property of connectedness is called total-diconnectedness.

definition IsTotDis (˙ –is totally-disconnected″ 70)
where IsTotDis ≡ ANTI(IsConnected)

lemma conn˙spectrum:
shows (A–is in the spectrum of″IsConnected) ⟷ A≲1
proof
assume A–is in the spectrum of″IsConnected
then have ∀T. (T–is a topology″∧⋃T≈A) ⟶ (T–is connected″) using Spec˙def by auto
moreover
have Pow(A)–is a topology″ using Pow˙is˙top by auto
moreover
have ⋃(Pow(A))=A by auto
then have ⋃(Pow(A))≈A by auto
ultimately have Pow(A) –is connected″ by auto
{
assume A≠0
then obtain E where E∈A by blast
then have –E″∈Pow(A) by auto
moreover
have A-–E″∈Pow(A) by auto
ultimately have –E″∈Pow(A)∧–E″–is closed in″Pow(A) unfolding IsClosed˙def by auto
with ‹Pow(A) –is connected″› have –E″=A unfolding IsConnected˙def by auto
then have A≈1 using singleton˙eqpoll˙1 by auto
then have A≲1 using eqpoll˙imp˙lepoll by auto
}
moreover
{
assume A=0
then have A≲1 using empty˙lepollI[of 1] by auto
}
ultimately show A≲1 by auto
next
assume A≲1
{
fix T
assume T–is a topology″⋃T≈A
{
assume ⋃T=0
with ‹T–is a topology″› have T=–0″ using empty˙open by auto
then have T–is connected″ unfolding IsConnected˙def by auto

```
    }
    moreover
    {
      assume ⋃T≠0
      moreover
      from ‹A≲1›‹⋃T≈A› have ⋃T≲1 using eq˙lepoll˙trans by auto
      ultimately
      obtain E where ⋃T=–E″ using lepoll˙1˙is˙sing by blast
      moreover
      have T⊆Pow(⋃T) by auto
      ultimately have T⊆Pow(–E″) by auto
      then have T⊆–0,–E″″ by blast
      with ‹T–is a topology″› have –0″⊆T T⊆–0,–E″″ using empty˙open by auto
      then have T–is connected″ unfolding IsConnected˙def by auto
    }
    ultimately have T–is connected″ by auto
  }
  then show A–is in the spectrum of″IsConnected unfolding Spec˙def by auto
qed
```

The discrete space is a first example of totally-disconnected space.

```
lemma discrete˙tot˙dis:
  shows Pow(X) –is totally-disconnected″
proof-
  {
    fix A assume A∈Pow(X) and con:(Pow(X)–restricted to″A)–is connected″
     have res:(Pow(X)–restricted to″A)=Pow(A) unfolding RestrictedTo˙def using
‹A∈Pow(X)›
      by blast
    {
      assume A=0
      then have A≲1 using empty˙lepollI[of 1] by auto
      then have A–is in the spectrum of″IsConnected using conn˙spectrum by auto
    }
    moreover
    {
      assume A≠0
      then obtain E where E∈A by blast
      then have –E″∈Pow(A) by auto
      moreover
      have A-–E″∈Pow(A) by auto
     ultimately have –E″∈Pow(A)∧–E″–is closed in″Pow(A) unfolding IsClosed˙def
by auto
      with con res have –E″=A unfolding IsConnected˙def by auto
      then have A≈1 using singleton˙eqpoll˙1 by auto
      then have A≲1 using eqpoll˙imp˙lepoll by auto
      then have A–is in the spectrum of″IsConnected using conn˙spectrum by auto
    }
    ultimately have A–is in the spectrum of″IsConnected by auto
```

```
  }
  then show thesis unfolding IsTotDis˙def antiProperty˙def by auto
qed
```

An space is hyperconnected iff every two non-empty open sets meet.

```
definition IsHConnected (˙–is hyperconnected″90)
  where T–is hyperconnected″ ≡∀ U V. U∈T ∧ V∈T ∧ U∩V=0 ⟶ U=0∨V=0
```

Every hyperconnected space is connected.

```
lemma HConn˙imp˙Conn:
  assumes T–is hyperconnected″
  shows T–is connected″
proof-
  {
    fix U
    assume U∈TU –is closed in″T
    then have ⋃T-U∈TU∈T using IsClosed˙def by auto
    moreover
    have (⋃T-U)∩U=0 by auto
    moreover
    note assms
    ultimately
    have U=0∨(⋃T-U)=0 using IsHConnected˙def by auto
    with ‹U∈T› have U=0∨U=⋃T by auto
  }
  then show thesis using IsConnected˙def by auto
qed

lemma Indiscrete˙HConn:
  shows –0,X″–is hyperconnected″
  unfolding IsHConnected˙def by auto
```

A first example of an hyperconnected space but not indiscrete, is the cofinite topology on the natural numbers.

```
lemma Cofinite˙nat˙HConn:
  assumes ¬(X≺nat)
  shows (CoFinite X)–is hyperconnected″
proof-
  {
    fix U V
    assume U∈(CoFinite X)V∈(CoFinite X)U∩V=0
    then have eq:(X-U)≺nat∨U=0(X-V)≺nat∨V=0 unfolding Cofinite˙def
      CoCardinal˙def by auto
    from ‹U∩V=0› have un:(X-U)∪(X-V)=X by auto
    {
      assume AS:(X-U)≺nat(X-V)≺nat
      from un have X≺nat using less˙less˙imp˙un˙less[OF AS InfCard˙nat] by auto
      then have False using assms by auto
    }
```

```
    with eq(1,2) have U=0∨V=0 by auto
  }
  then show (CoFinite X)–is hyperconnected″ using IsHConnected˙def by auto
qed

lemma HConn˙spectrum:
  shows (A–is in the spectrum of″IsHConnected) ⟷ A≲1
proof
  assume A–is in the spectrum of″IsHConnected
   then have ∀T. (T–is a topology″∧⋃T≈A) ⟶ (T–is hyperconnected″) using
Spec˙def by auto
  moreover
  have Pow(A)–is a topology″ using Pow˙is˙top by auto
  moreover
  have ⋃(Pow(A))=A by auto
  then have ⋃(Pow(A))≈A by auto
  ultimately
  have HC˙Pow:Pow(A)–is hyperconnected″ by auto
  {
    assume A=0
    then have A≲1 using empty˙lepollI by auto
  }
  moreover
  {
    assume A≠0
    then obtain e where e∈A by blast
    then have –e″∈Pow(A) by auto
    moreover
    have A-–e″∈Pow(A) by auto
    moreover
    have –e″∩(A-–e″)=0 by auto
    moreover
    note HC˙Pow
    ultimately have A-–e″=0 unfolding IsHConnected˙def by blast
    with ‹e∈A› have A=–e″ by auto
    then have A≈1 using singleton˙eqpoll˙1 by auto
    then have A≲1 using eqpoll˙imp˙lepoll by auto
  }
  ultimately show A≲1 by auto
next
  assume A≲1
  {
    fix T
    assume T–is a topology″⋃T≈A
    {
      assume ⋃T=0
      with ‹T–is a topology″› have T=–0″ using empty˙open by auto
      then have T–is hyperconnected″ unfolding IsHConnected˙def by auto
    }
```

```
    moreover
    {
      assume ⋃T≠0
      moreover
      from ‹A≲1›‹⋃T≈A› have ⋃T≲1 using eq˙lepoll˙trans by auto
      ultimately
      obtain E where ⋃T=–E″ using lepoll˙1˙is˙sing by blast
      moreover
      have T⊆Pow(⋃T) by auto
      ultimately have T⊆Pow(–E″) by auto
      then have T⊆–0,–E″″ by blast
      with ‹T–is a topology″› have –0″⊆T T⊆–0,–E″″ using empty˙open by auto
      then have T–is hyperconnected″ unfolding IsHConnected˙def by auto
    }
    ultimately have T–is hyperconnected″ by auto
  }
  then show A–is in the spectrum of″IsHConnected unfolding Spec˙def by auto
qed
```

In the following results we will show that anti-hyperconnectedness is a separation property between T_1 and T_2. We will show also that both implications are proper.

First, the closure of a point in every topological space is always hyperconnected. This is the reason why every anti-hyperconnected space must be T_1: every singleton must be closed.

```
lemma (in topology0)cl˙point˙imp˙HConn:
  assumes x∈⋃T
  shows (T–restricted to″Closure(–x″,T))–is hyperconnected″
proof-
  from assms have sub:Closure(–x″,T)⊆⋃T using Top˙3˙L11 by auto
  then have tot:⋃(T–restricted to″Closure(–x″,T))=Closure(–x″,T) unfolding Re-
strictedTo˙def by auto
  {
    fix A B
   assume AS:A∈(T–restricted to″Closure(–x″,T))B∈(T–restricted to″Closure(–x″,T))A∩B=0
   then have B⊆⋃((T–restricted to″Closure(–x″,T)))A⊆⋃((T–restricted to″Closure(–x″,T)))
      by auto
    with tot have B⊆Closure(–x″,T)A⊆Closure(–x″,T) by auto
   from AS(1,2) obtain UA UB where UAUB:UA∈TUB∈TA=UA∩Closure(–x″,T)B=UB∩Closure(–x″,T)
      unfolding RestrictedTo˙def by auto
   then have Closure(–x″,T)-A=Closure(–x″,T)-(UA∩Closure(–x″,T)) Closure(–x″,T)-B=Closure(–x″,T)-(UB∩
      by auto
   then have Closure(–x″,T)-A=Closure(–x″,T)-(UA) Closure(–x″,T)-B=Closure(–x″,T)-(UB)
      by auto
   with sub have Closure(–x″,T)-A=Closure(–x″,T)∩(⋃T-UA) Closure(–x″,T)-B=Closure(–x″,T)∩(⋃T-UB)
by auto
    moreover
      from UAUB have (⋃T-UA)–is closed in″T(⋃T-UB)–is closed in″T using
```

```
Top˙3˙L9 by auto
    moreover
    have Closure(–x″,T)–is closed in″T using cl˙is˙closed assms by auto
    ultimately have (Closure(–x″,T)-A)–is closed in″T(Closure(–x″,T)-B)–is closed
in″T
      using Top˙3˙L5(1) by auto
    moreover
    {
      have x∈Closure(–x″,T) using cl˙contains˙set assms by auto
      moreover
      from AS(3) have x∉A∨x∉B by auto
      ultimately have x∈(Closure(–x″,T)-A)∨x∈(Closure(–x″,T)-B) by auto
    }
   ultimately have Closure(–x″,T)⊆(Closure(–x″,T)-A) ∨ Closure(–x″,T)⊆(Closure(–x″,T)-B)
      using Top˙3˙L13 by auto
    then have A∩Closure(–x″,T)=0 ∨ B∩Closure(–x″,T)=0 by auto
   with ‹B⊆Closure(–x″,T)›‹A⊆Closure(–x″,T)› have A=0∨B=0 using cl˙contains˙set
assms by blast
  }
  then show thesis unfolding IsHConnected˙def by auto
qed
```

A consequence is that every totally-disconnected space is T_1.

```
lemma (in topology0) tot˙dis˙imp˙T1:
  assumes T–is totally-disconnected″
  shows T–is T1″
proof-
  {
    fix x y
    assume y∈⋃Tx∈⋃Ty≠x
   then have (T–restricted to″Closure(–x″,T))–is hyperconnected″ using cl˙point˙imp˙HConn
by auto
   then have (T–restricted to″Closure(–x″,T))–is connected″ using HConn˙imp˙Conn
by auto
    moreover
    from ‹x∈⋃T› have Closure(–x″,T)⊆⋃T using Top˙3˙L11(1) by auto
    moreover
    note assms
     ultimately have Closure(–x″,T)–is in the spectrum of″IsConnected unfolding
IsTotDis˙def antiProperty˙def
      by auto
    then have Closure(–x″,T)≲1 using conn˙spectrum by auto
    moreover
    from ‹x∈⋃T› have x∈Closure(–x″,T) using cl˙contains˙set by auto
     ultimately have Closure(–x″,T)=–x″ using lepoll˙1˙is˙sing[of Closure(–x″,T) x]
by auto
    then have –x″–is closed in″T using Top˙3˙L8 ‹x∈⋃T› by auto
    then have ⋃T-–x″∈T unfolding IsClosed˙def by auto
    moreover
```

from ‹y∈⋃T›‹y≠x› have y∈⋃T-–x″∧x∉⋃T-–x″ by auto
ultimately have ∃U∈T. y∈U∧x∉U by force
}
then show thesis unfolding isT1˙def by auto
qed

In the literature, there exists a class of spaces called sober spaces; where the only non-empty closed hyperconnected subspaces are the closures of points and closures of different singletons are different.

definition IsSober (˙–is sober″90)
where T–is sober″ ≡ ∀A∈Pow(⋃T)-–0″. (A–is closed in″T ∧ ((T–restricted to″A)–is hyperconnected″)) ⟶ (∃x∈⋃T. A=Closure(–x″,T) ∧ (∀y∈⋃T. A=Closure(–y″,T) ⟶ y=x))

Being sober is weaker than being anti-hyperconnected.

theorem (in topology0) anti˙HConn˙imp˙sober:
assumes T–is anti-″IsHConnected
shows T–is sober″
proof-
{
fix A assume A∈Pow(⋃T)-–0″A–is closed in″T(T–restricted to″A)–is hyperconnected″
with assms have A–is in the spectrum of″IsHConnected unfolding antiProperty˙def by auto
then have A≲1 using HConn˙spectrum by auto
moreover
with ‹A∈Pow(⋃T)-–0″› have A≠0 by auto
then obtain x where x∈A by auto
ultimately have A=–x″ using lepoll˙1˙is˙sing by auto
with ‹A–is closed in″T› have –x″–is closed in″T by auto
moreover from ‹x∈A› ‹A∈Pow(⋃T)-–0″› have –x″∈Pow(⋃T) by auto
ultimately
have Closure(–x″,T)=–x″ unfolding Closure˙def ClosedCovers˙def by auto
with ‹A=–x″› have A=Closure(–x″,T) by auto
moreover
{
fix y assume y∈⋃TA=Closure(–y″,T)
then have –y″⊆Closure(–y″,T) using cl˙contains˙set by auto
with ‹A=Closure(–y″,T)› have y∈A by auto
with ‹A=–x″› have y=x by auto
}
then have ∀y∈⋃T. A=Closure(–y″,T) ⟶ y=x by auto
moreover note ‹–x″∈Pow(⋃T)›
ultimately have ∃x∈⋃T. A=Closure(–x″,T)∧(∀y∈⋃T. A=Closure(–y″,T) ⟶ y=x) by auto
}
then show thesis using IsSober˙def by auto
qed

Every sober space is T_0.

```
lemma (in topology0) sober˙imp˙T0:
  assumes T–is sober″
  shows T–is T0″
proof-
  {
    fix x y
    assume AS:x∈⋃Ty∈⋃Tx≠y∀U∈T. x∈U ⟷ y∈U
    from ‹x∈⋃T› have clx:Closure(–x″,T) –is closed in″T using cl˙is˙closed by auto
     with ‹x∈⋃T› have (⋃T-Closure(–x″,T))∈T using Top˙3˙L11(1) unfolding Is-
Closed˙def by auto
    moreover
    from ‹x∈⋃T› have x∈Closure(–x″,T) using cl˙contains˙set by auto
    moreover
    note AS(1,4)
    ultimately have y∉(⋃T-Closure(–x″,T)) by auto
    with AS(2) have y∈Closure(–x″,T) by auto
    with clx have ineq1:Closure(–y″,T)⊆Closure(–x″,T) using Top˙3˙L13 by auto
    from ‹y∈⋃T› have cly:Closure(–y″,T) –is closed in″T using cl˙is˙closed by auto
     with ‹y∈⋃T› have (⋃T-Closure(–y″,T))∈T using Top˙3˙L11(1) unfolding Is-
Closed˙def by auto
    moreover
    from ‹y∈⋃T› have y∈Closure(–y″,T) using cl˙contains˙set by auto
    moreover
    note AS(2,4)
    ultimately have x∉(⋃T-Closure(–y″,T)) by auto
    with AS(1) have x∈Closure(–y″,T) by auto
    with cly have Closure(–x″,T)⊆Closure(–y″,T) using Top˙3˙L13 by auto
    with ineq1 have eq:Closure(–x″,T)=Closure(–y″,T) by auto
   have Closure(–x″,T)∈Pow(⋃T)-–0″ using Top˙3˙L11(1) ‹x∈⋃T› ‹x∈Closure(–x″,T)›
by auto
    moreover note assms clx
    ultimately have ∃t∈⋃T.( Closure(–x″,T) = Closure(–t″, T) ∧ (∀y∈⋃T. Clo-
sure(–x″,T) = Closure(–y″, T) ⟶ y = t))
      unfolding IsSober˙def using cl˙point˙imp˙HConn[OF ‹x∈⋃T›] by auto
    then obtain t where t˙def:t∈⋃TClosure(–x″,T) = Closure(–t″, T)∀y∈⋃T. Clo-
sure(–x″,T) = Closure(–y″, T) ⟶ y = t
      by blast
    with eq have y=t using ‹y∈⋃T› by auto
    moreover from t˙def ‹x∈⋃T› have x=t by blast
    ultimately have y=x by auto
    with ‹x≠y› have False by auto
  }
  then have ∀x y. x∈⋃T∧y∈⋃T∧x≠y ⟶ (∃U∈T. (x∈U∧y∉U)∨(y∈U∧x∉U)) by
auto
  then show thesis using isT0˙def by auto
qed
```

Every T_2 space is anti-hyperconnected.

```
theorem (in topology0) T2˙imp˙anti˙HConn:
  assumes T–is T₂″
  shows T–is anti-″IsHConnected
proof-
  {
    fix TT
    assume TT–is a topology″ TT–is hyperconnected″TT–is T₂″
    {
      assume ⋃TT=0
      then have ⋃TT≲1 using empty˙lepollI by auto
      then have (⋃TT)–is in the spectrum of″IsHConnected using HConn˙spectrum
by auto
    }
    moreover
    {
      assume ⋃TT≠0
      then obtain x where x∈⋃TT by blast
      {
        fix y
        assume y∈⋃TTx≠y
        with ‹TT–is T₂″›‹x∈⋃TT› obtain U V where U∈TTV∈TTx∈Uy∈VU∩V=0
unfolding isT2˙def by blast
        with ‹TT–is hyperconnected″› have False using IsHConnected˙def by auto
      }
      with ‹x∈⋃TT› have ⋃TT=–x″ by auto
      then have ⋃TT≈1 using singleton˙eqpoll˙1 by auto
      then have ⋃TT≲1 using eqpoll˙imp˙lepoll by auto
      then have (⋃TT)–is in the spectrum of″IsHConnected using HConn˙spectrum
by auto
    }
    ultimately have (⋃TT)–is in the spectrum of″IsHConnected by blast
  }
  then have ∀T. ((T–is a topology″∧(T–is hyperconnected″)∧(T–is T₂″))⟶((⋃T)–is
in the spectrum of″IsHConnected))
    by auto
  moreover
  note here˙T2
  ultimately
  have ∀T.  T–is a topology″ ⟶ ((T–is T₂″)⟶(T–is anti-″IsHConnected)) using
Q˙P˙imp˙Spec[where P=IsHConnected and Q=isT2]
    by auto
  then show thesis using assms topSpaceAssum by auto
qed
```

Every anti-hyperconnected space is T_1.

```
theorem anti˙HConn˙imp˙T1:
  assumes T–is anti-″IsHConnected
  shows T–is T₁″
proof-
```

```
  {
    fix x y
    assume x∈⋃Ty∈⋃Tx≠y
    {
      assume AS:∀U∈T. x∉U∨y∈U
      from ‹x∈⋃T›‹y∈⋃T› have –x,y″∈Pow(⋃T) by auto
      then have sub:(T–restricted to″–x,y″)⊆Pow(–x,y″) using RestrictedTo˙def by
auto
      {
        fix U V
        assume H:U∈T–restricted to″–x,y″ V∈(T–restricted to″–x,y″)U∩V=0
        with AS have x∈U⟶y∈Ux∈V⟶y∈V unfolding RestrictedTo˙def by auto
        with H(1,2) sub have x∈U⟶U=–x,y″x∈V⟶V=–x,y″ by auto
        with H sub have x∈U⟶(U=–x,y″∧V=0)x∈V⟶(V=–x,y″∧U=0) by auto
        then have (x∈U∨x∈V)⟶(U=0∨V=0) by auto
        moreover
        from sub H have (x∉U∧x∉V)⟶ (U=0∨V=0) by blast
        ultimately have U=0∨V=0 by auto
      }
       then have (T–restricted to″–x,y″)–is hyperconnected″ unfolding IsHCon-
nected˙def by auto
      with assms‹–x,y″∈Pow(⋃T)› have –x,y″–is in the spectrum of″IsHConnected
unfolding antiProperty˙def
        by auto
      then have –x,y″≲1 using HConn˙spectrum by auto
      moreover
      have x∈–x,y″ by auto
      ultimately have –x,y″=–x″ using lepoll˙1˙is˙sing[of –x,y″x] by auto
      moreover
      have y∈–x,y″ by auto
      ultimately have y∈–x″ by auto
      then have y=x by auto
      with ‹x≠y› have False by auto
    }
    then have ∃U∈T. x∈U∧y∉U by auto
  }
  then show thesis using isT1˙def by auto
qed
```

There is at least one topological space that is T_1, but not anti-hyperconnected. This space is the cofinite topology on the natural numbers.

```
lemma Cofinite˙not˙anti˙HConn:
  shows ¬((CoFinite nat)–is anti-″IsHConnected) and (CoFinite nat)–is T1″
proof-
  {
    assume (CoFinite nat)–is anti-″IsHConnected
    moreover
   have ⋃(CoFinite nat)=nat unfolding Cofinite˙def using union˙cocardinal by auto
    moreover
```

have (CoFinite nat)–restricted to″nat=(CoFinite nat) using subspace˙cocardinal unfolding Cofinite˙def
by auto
moreover
have ¬(nat≺nat) by auto
then have (CoFinite nat)–is hyperconnected″ using Cofinite˙nat˙HConn[of nat] by auto
ultimately have nat–is in the spectrum of″IsHConnected unfolding antiProperty˙def by auto
then have nat≲1 using HConn˙spectrum by auto
moreover
have 1∈nat by auto
then have 1≺nat using n˙lesspoll˙nat by auto
ultimately have nat≺nat using lesspoll˙trans1 by auto
then have False by auto
}
then show ¬((CoFinite nat)–is anti-″IsHConnected) by auto
next
show (CoFinite nat)–is T_1″ using cocardinal˙is˙T1 InfCard˙nat unfolding Cofinite˙def by auto
qed

The join-topology build from the cofinite topology on the natural numbers, and the excluded set topology on the natural numbers excluding –0,1″; is just the union of both.

lemma join˙top˙cofinite˙excluded˙set:
shows (joinT –CoFinite nat,ExcludedSet(nat,–0,1″)″)=(CoFinite nat)∪ ExcludedSet(nat,–0,1″)
proof-
have coftop:(CoFinite nat)–is a topology″ unfolding Cofinite˙def using CoCar˙is˙topology InfCard˙nat by auto
moreover
have ExcludedSet(nat,–0,1″)–is a topology″ using excludedset˙is˙topology by auto
moreover
have exuni:⋃ExcludedSet(nat,–0,1″)=nat using union˙excludedset by auto
moreover
have cofuni:⋃(CoFinite nat)=nat using union˙cocardinal unfolding Cofinite˙def by auto
ultimately have (joinT –CoFinite nat,ExcludedSet(nat,–0,1″)″) = (THE T. (CoFinite nat)∪ExcludedSet(nat,–0,1″) –is a subbase for″ T)
using joinT˙def by auto
moreover
have ⋃(CoFinite nat)∈CoFinite nat using CoCar˙is˙topology[OF InfCard˙nat] unfolding Cofinite˙def IsATopology˙def
by auto
with cofuni have n:nat∈CoFinite nat by auto
have Pa:(CoFinite nat)∪ExcludedSet(nat,–0,1″) –is a subbase for″–⋃A. A∈Pow(–⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,–0,1″))″)″
using Top˙subbase(2) by auto

have –⋃A. A∈Pow(–⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,–0,1″))″)″=(THE T. (CoFinite nat)∪ExcludedSet(nat,–0,1″) –is a subbase for″ T)
using same˙subbase˙same˙top[where B=(CoFinite nat)∪ExcludedSet(nat,–0,1″), OF ˙ Pa] the˙equality[where a=–⋃A. A∈Pow(–⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,–0,1″))″)″ and P=λT. ((CoFinite nat)∪ExcludedSet(nat,–0,1″))–is a subbase for″T,
OF Pa] by auto
ultimately have equal:(joinT –CoFinite nat,ExcludedSet(nat,–0,1″)″) =–⋃A. A∈Pow(–⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,–0,1″))″)″
by auto
{
fix U assume U∈–⋃A. A∈Pow(–⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,–0,1″))″)″
then obtain AU where U=⋃AU and base:AU∈Pow(–⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,–0,1″))″)
by auto
have (CoFinite nat)⊆Pow(⋃(CoFinite nat)) by auto
moreover
have ExcludedSet(nat,–0,1″)⊆Pow(⋃ExcludedSet(nat,–0,1″)) by auto
moreover
note cofuni exuni
ultimately have sub:(CoFinite nat)∪ExcludedSet(nat,–0,1″)⊆Pow(nat) by auto
from base have ∀S∈AU. S∈–⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,–0,1″))″ by blast
then have ∀S∈AU. ∃B∈FinPow((CoFinite nat)∪ExcludedSet(nat,–0,1″)). S=⋂B by blast
then have eq:∀S∈AU. ∃B∈Pow((CoFinite nat)∪ExcludedSet(nat,–0,1″)). S=⋂B unfolding FinPow˙def by blast
{
fix S assume S∈AU
with eq obtain B where B∈Pow((CoFinite nat)∪ExcludedSet(nat,–0,1″))S=⋂B by auto
with sub have B∈Pow(Pow(nat)) by auto
{
fix x assume x∈⋂B
then have ∀N∈B. x∈NB≠0 by auto
with ‹B∈Pow(Pow(nat))› have x∈nat by blast
}
with ‹S=⋂B› have S∈Pow(nat) by auto
}
then have ∀S∈AU. S∈Pow(nat) by blast
with ‹U=⋃AU› have U∈Pow(nat) by auto
{
assume 0∈U∨1∈U
with ‹U=⋃AU› obtain S where S∈AU0∈S∨1∈S by auto
with base obtain BS where S=⋂BS and bsbase:BS∈FinPow((CoFinite nat)∪ExcludedSet(nat,–0,1″)) by auto
with ‹0∈S∨1∈S› have ∀M∈BS. 0∈M∨1∈M by auto
then have ∀M∈BS. M∉ExcludedSet(nat,–0,1″)-–nat″ unfolding Excluded-Point˙def ExcludedSet˙def by auto
moreover

note bsbase n
ultimately have BS∈FinPow(CoFinite nat) unfolding FinPow˙def by auto
moreover
from ‹0∈S∨1∈S› have S≠0 by auto
with ‹S=⋂BS› have BS≠0 by auto
moreover
note coftop
ultimately have ⋂BS∈CoFinite nat using topology0.fin˙inter˙open˙open[OF topology0˙CoCardinal[OF InfCard˙nat]]
unfolding Cofinite˙def by auto
with ‹S=⋂BS› have S∈CoFinite nat by auto
with ‹0∈S∨1∈S› have nat-S≺nat unfolding Cofinite˙def CoCardinal˙def by auto
moreover
from ‹U=⋃AU›‹S∈AU› have S⊆U by auto
then have nat-U⊆nat-S by auto
then have nat-U≲nat-S using subset˙imp˙lepoll by auto
ultimately
have nat-U≺nat using lesspoll˙trans1 by auto
with ‹U∈Pow(nat)› have U∈CoFinite nat unfolding Cofinite˙def CoCardinal˙def by auto
with ‹U∈Pow(nat)› have U∈ (CoFinite nat)∪ ExcludedSet(nat,–0,1″) by auto
}
with ‹U∈Pow(nat)› have U∈(CoFinite nat)∪ ExcludedSet(nat,–0,1″) unfolding ExcludedSet˙def by blast
}
then have (–⋃A . A ∈ Pow(–⋂B . B ∈ FinPow((CoFinite nat) ∪ ExcludedSet(nat,–0,1″))″)″) ⊆ (CoFinite nat)∪ ExcludedSet(nat,–0,1″)
by blast
moreover
{
fix U
assume U∈(CoFinite nat)∪ ExcludedSet(nat,–0,1″)
then have –U″∈FinPow((CoFinite nat) ∪ ExcludedSet(nat,–0,1″)) unfolding FinPow˙def by auto
then have –U″∈Pow(–⋂B . B ∈ FinPow((CoFinite nat) ∪ ExcludedSet(nat,–0,1″))″) by blast
moreover
have U=⋃–U″ by auto
ultimately have U∈–⋃A . A ∈ Pow(–⋂B . B ∈ FinPow((CoFinite nat) ∪ ExcludedSet(nat,–0,1″))″)″ by blast
}
then have (CoFinite nat)∪ ExcludedSet(nat,–0,1″)⊆–⋃A . A ∈ Pow(–⋂B . B ∈ FinPow((CoFinite nat) ∪ ExcludedSet(nat,–0,1″))″)″
by auto
ultimately have (CoFinite nat)∪ ExcludedSet(nat,–0,1″)=–⋃A . A ∈ Pow(–⋂B . B ∈ FinPow((CoFinite nat) ∪ ExcludedSet(nat,–0,1″))″)″
by auto
with equal show thesis by auto
qed

The previous topology in not T_2, but is anti-hyperconnected.

theorem join˙Cofinite˙ExclPoint˙not˙T2:
shows
¬((joinT –CoFinite nat, ExcludedSet(nat,–0,1″)″)–is T_2″) and
(joinT –CoFinite nat,ExcludedSet(nat,–0,1″)″)–is anti-″ IsHConnected
proof-
have (CoFinite nat) ⊆ (CoFinite nat) ∪ ExcludedSet(nat,–0,1″) by auto
have ⋃((CoFinite nat)∪ ExcludedSet(nat,–0,1″))=(⋃(CoFinite nat))∪ (⋃ExcludedSet(nat,–0,1″))
by auto
moreover
have ...=nat unfolding Cofinite˙def using union˙cocardinal union˙excludedset by auto
ultimately have tot:⋃((CoFinite nat)∪ ExcludedSet(nat,–0,1″))=nat by auto
{
assume (joinT –CoFinite nat,ExcludedSet(nat,–0,1″)″) –is T_2″
then have t2:((CoFinite nat)∪ ExcludedSet(nat,–0,1″))–is T_2″ using join˙top˙cofinite˙excluded˙set
by auto
with tot have ∃U∈((CoFinite nat)∪ ExcludedSet(nat,–0,1″)). ∃V∈((CoFinite nat)∪ ExcludedSet(nat,–0,1″)). 0∈U∧1∈V∧U∩V=0 using isT2˙def by auto
then obtain U V where U ∈ (CoFinite nat) ∨ (0 ∉ U∧1∉U)V ∈ (CoFinite nat) ∨ (0 ∉ V∧1∉V)0∈U1∈VU∩V=0
unfolding ExcludedSet˙def by auto
then have U∈(CoFinite nat)V∈(CoFinite nat) by auto
with ‹0∈U›‹1∈V› have U∩V≠0 using Cofinite˙nat˙HConn IsHConnected˙def by auto
with ‹U∩V=0› have False by auto
}
then show ¬((joinT –CoFinite nat,ExcludedSet(nat,–0,1″)″)–is T_2″) by auto
{
fix A assume AS:A∈Pow(⋃((CoFinite nat)∪ ExcludedSet(nat,–0,1″)))(((CoFinite nat)∪ ExcludedSet(nat,–0,1″))–restricted to″A)–is hyperconnected″
with tot have A∈Pow(nat) by auto
then have sub:A∩nat=A by auto
have ((CoFinite nat)∪ ExcludedSet(nat,–0,1″))–restricted to″A=((CoFinite nat)–restricted to″A)∪ (ExcludedSet(nat,–0,1″)–restricted to″A)
unfolding RestrictedTo˙def by auto
also from sub have ...=(CoFinite A)∪ExcludedSet(A,–0,1″) using subspace˙excludedset[ofnat–0,1″A] subspace˙cocardinal[of natnatA] unfolding Cofinite˙def
by auto
finally have ((CoFinite nat)∪ ExcludedSet(nat,–0,1″))–restricted to″A=(CoFinite A)∪ExcludedSet(A,–0,1″) by auto
with AS(2) have eq:((CoFinite A)∪ExcludedSet(A,–0,1″))–is hyperconnected″ by auto
{
assume –0,1″∩A=0
then have (CoFinite A)∪ExcludedSet(A,–0,1″)=Pow(A) using empty˙excludedset[of –0,1″A] unfolding Cofinite˙def CoCardinal˙def
by auto

with eq have Pow(A)–is hyperconnected″ by auto
then have Pow(A)–is connected″ using HConn˙imp˙Conn by auto
moreover
have Pow(A)–is anti-″IsConnected using discrete˙tot˙dis unfolding IsTotDis˙def by auto
moreover
have ⋃(Pow(A))∈Pow(⋃(Pow(A))) by auto
moreover
have Pow(A)–restricted to″⋃(Pow(A))=Pow(A) unfolding RestrictedTo˙def by blast
ultimately have (⋃(Pow(A)))–is in the spectrum of″IsConnected unfolding antiProperty˙def
by auto
then have A–is in the spectrum of″IsConnected by auto
then have A≲1 using conn˙spectrum by auto
then have A–is in the spectrum of″IsHConnected using HConn˙spectrum by auto
}
moreover
{
assume AS:–0,1″∩A≠0
{
assume A=–0″∨A=–1″
then have A≈1 using singleton˙eqpoll˙1 by auto
then have A≲1 using eqpoll˙imp˙lepoll by auto
then have A–is in the spectrum of″IsHConnected using HConn˙spectrum by auto
}
moreover
{
assume AS2:¬(A=–0″∨A=–1″)
{
assume AS3:A⊆–0,1″
with AS AS2 have A˙def:A=–0,1″ by blast
then have ExcludedSet(A,–0,1″)=ExcludedSet(A,A) by auto
moreover have ExcludedSet(A,A)=–0,A″ unfolding ExcludedSet˙def by blast
ultimately have ExcludedSet(A,–0,1″)=–0,A″ by auto
moreover
have 0∈(CoFinite A) using empty˙open[of CoFinite A]
CoCar˙is˙topology[OF InfCard˙nat,of A] unfolding Cofinite˙def by auto
moreover
have ⋃(CoFinite A)=A using union˙cocardinal unfolding Cofinite˙def by auto
then have A∈(CoFinite A) using CoCar˙is˙topology[OF InfCard˙nat,of A] unfolding Cofinite˙def
IsATopology˙def by auto
ultimately have (CoFinite A)∪ExcludedSet(A,–0,1″)=(CoFinite A) by auto
with eq have(CoFinite A)–is hyperconnected″ by auto
with A˙def have hyp:(CoFinite –0,1″)–is hyperconnected″ by auto

```
        have –0″≈1–1″≈1 using singleton˙eqpoll˙1 by auto
        moreover
        have 1≺nat using n˙lesspoll˙nat by auto
        ultimately have –0″≺nat–1″≺nat using eq˙lesspoll˙trans by auto
        moreover
        have –0,1″-–1″=–0″–0,1″-–0″=–1″ by auto
          ultimately have –1″∈(CoFinite –0,1″)–0″∈(CoFinite –0,1″) –1″∩–0″=0
unfolding Cofinite˙def CoCardinal˙def
          by auto
        with hyp have False unfolding IsHConnected˙def by auto
      }
      then obtain t where t∈A t≠0 t≠1 by auto
      then have –t″∈ExcludedSet(A,–0,1″) unfolding ExcludedSet˙def by auto
      moreover
      {
        have –t″≈1 using singleton˙eqpoll˙1 by auto
        moreover
        have 1≺nat using n˙lesspoll˙nat by auto
        ultimately have –t″≺nat using eq˙lesspoll˙trans by auto
        moreover
        with ‹t∈A› have A-(A-–t″)=–t″ by auto
        ultimately have A-–t″∈(CoFinite A) unfolding Cofinite˙def CoCardinal˙def
          by auto
      }
      ultimately have –t″∈((CoFinite A)∪ExcludedSet(A,–0,1″))A-–t″∈((CoFinite
A)∪ExcludedSet(A,–0,1″))
        –t″∩(A-–t″)=0 by auto
      with eq have A-–t″=0 unfolding IsHConnected˙def by auto
      with ‹t∈A› have A=–t″ by auto
      then have A≈1 using singleton˙eqpoll˙1 by auto
      then have A≲1 using eqpoll˙imp˙lepoll by auto
      then have A–is in the spectrum of″IsHConnected using HConn˙spectrum by
auto
    }
    ultimately have A–is in the spectrum of″IsHConnected by auto
  }
  ultimately have A–is in the spectrum of″IsHConnected by auto
 }
 then have ((CoFinite nat)∪ExcludedSet(nat,–0,1″))–is anti-″IsHConnected un-
folding antiProperty˙def
  by auto
then show (joinT –CoFinite nat, ExcludedSet(nat,–0,1″)″)–is anti-″IsHConnected
using join˙top˙cofinite˙excluded˙set
  by auto
qed
```

Let's show that anti-hyperconnected is in fact T_1 and sober. The trick of the proof lies in the fact that if a subset is hyperconnected, its closure is so too (the closure of a point is then always hyperconnected because singletons

are in the spectrum); since the closure is closed, we can apply the sober property on it.

```
theorem (in topology0) T1˙sober˙imp˙anti˙HConn:
  assumes T–is T1″ and T–is sober″
  shows T–is anti-″IsHConnected
proof-
  {
    fix A assume AS:A∈Pow(⋃T)(T–restricted to″A)–is hyperconnected″
    {
      assume A=0
      then have A≲1 using empty˙lepollI by auto
       then have A–is in the spectrum of″IsHConnected using HConn˙spectrum by
auto
    }
    moreover
    {
      assume A≠0
      then obtain x where x∈A by blast
      {
        assume ¬((T–restricted to″Closure(A,T))–is hyperconnected″)
     then obtain U V where UV˙def:U∈(T–restricted to″Closure(A,T))V∈(T–restricted
to″Closure(A,T))
          U∩V=0U≠0V≠0 using IsHConnected˙def by auto
     then obtain UCA VCA where UCA∈TVCA∈TU=UCA∩Closure(A,T)V=VCA∩Closure(A,T)
          unfolding RestrictedTo˙def by auto
        from ‹A∈Pow(⋃T)› have A⊆Closure(A,T) using cl˙contains˙set by auto
          then have UCA∩A⊆UCA∩Closure(A,T)VCA∩A⊆VCA∩Closure(A,T) by
auto
     with ‹U=UCA∩Closure(A,T)›‹V=VCA∩Closure(A,T)›‹U∩V=0› have (UCA∩A)∩(VCA∩A)=0
by auto
        moreover
     from ‹UCA∈T›‹VCA∈T› have UCA∩A∈(T–restricted to″A)VCA∩A∈(T–restricted
to″A)
          unfolding RestrictedTo˙def by auto
        moreover
        note AS(2)
        ultimately have UCA∩A=0∨VCA∩A=0 using IsHConnected˙def by auto
        with ‹A⊆Closure(A,T)› have A⊆Closure(A,T)-UCA∨A⊆Closure(A,T)-VCA
by auto
        moreover
        {
      have Closure(A,T)-UCA=Closure(A,T)∩(⋃T-UCA)Closure(A,T)-VCA=Closure(A,T)∩(⋃T-VCA)
          using Top˙3˙L11(1) AS(1) by auto
          moreover
            with ‹UCA∈T›‹VCA∈T› have (⋃T-UCA)–is closed in″T(⋃T-VCA)–is
closed in″TClosure(A,T)–is closed in″T
          using Top˙3˙L9 cl˙is˙closed AS(1) by auto
         ultimately have (Closure(A,T)-UCA)–is closed in″T(Closure(A,T)-VCA)–is
closed in″T
```

using Top˙3˙L5(1) by auto
}
ultimately
have Closure(A,T)⊆Closure(A,T)-UCA∨Closure(A,T)⊆Closure(A,T)-VCA using Top˙3˙L13
by auto
then have UCA∩Closure(A,T)=0∨VCA∩Closure(A,T)=0 by auto
with ‹U=UCA∩Closure(A,T)›‹V=VCA∩Closure(A,T)› have U=0∨V=0 by auto
with ‹U≠0›‹V≠0› have False by auto
}
then have (T–restricted to″Closure(A,T))–is hyperconnected″ by auto
moreover
have Closure(A,T)–is closed in″T using cl˙is˙closed AS(1) by auto
moreover
from ‹x∈A› have Closure(A,T)≠0 using cl˙contains˙set AS(1) by auto
moreover
from AS(1) have Closure(A,T)⊆⋃T using Top˙3˙L11(1) by auto
ultimately have Closure(A,T)∈Pow(⋃T)-–0″(T –restricted to″ Closure(A, T))–is hyperconnected″ Closure(A, T) –is closed in″ T
by auto
moreover note assms(2)
ultimately have ∃x∈⋃T. (Closure(A,T)=Closure(–x″,T)∧ (∀y∈⋃T. Closure(A,T) = Closure(–y″, T) ⟶ y = x)) unfolding IsSober˙def
by auto
then obtain y where y∈⋃TClosure(A,T)=Closure(–y″,T) by auto
moreover
{
fix z assume z∈(⋃T)-–y″
with assms(1) ‹y∈⋃T› obtain U where U∈T z∈U y∉U using isT1˙def by blast
then have U∈T z∈U U⊆(⋃T)-–y″ by auto
then have ∃U∈T. z∈U ∧ U⊆(⋃T)-–y″ by auto
}
then have ∀z∈(⋃T)-–y″. ∃U∈T. z∈U ∧ U⊆(⋃T)-–y″ by auto
then have ⋃T-–y″∈T using open˙neigh˙open by auto
with ‹y∈⋃T› have –y″ –is closed in″T using IsClosed˙def by auto
with ‹y∈⋃T› have Closure(–y″,T)=–y″ using Top˙3˙L8 by auto
with ‹Closure(A,T)=Closure(–y″,T)› have Closure(A,T)=–y″ by auto
with AS(1) have A⊆–y″ using cl˙contains˙set[of A] by auto
with ‹A≠0› have A=–y″ by auto
then have A≈1 using singleton˙eqpoll˙1 by auto
then have A≲1 using eqpoll˙imp˙lepoll by auto
then have A–is in the spectrum of″IsHConnected using HConn˙spectrum by auto
}
ultimately have A–is in the spectrum of″IsHConnected by blast
}
then show thesis using antiProperty˙def by auto
qed

theorem (in topology0) anti˙HConn˙iff˙T1˙sober:
shows (T–is anti-″IsHConnected) ⟷ (T–is sober″∧T–is T_1″)
using T1˙sober˙imp˙anti˙HConn anti˙HConn˙imp˙T1 anti˙HConn˙imp˙sober by auto

A space is ultraconnected iff every two non-empty closed sets meet.

definition IsUConnected (˙–is ultraconnected″80)
where T–is ultraconnected″≡ ∀A B. A–is closed in″T∧B–is closed in″T∧A∩B=0 ⟶ A=0∨B=0

Every ultraconnected space is trivially normal.

lemma (in topology0)UConn˙imp˙normal:
assumes T–is ultraconnected″
shows T–is normal″
proof-
{
fix A B
assume AS:A–is closed in″T B–is closed in″TA∩B=0
with assms have A=0∨B=0 using IsUConnected˙def by auto
with AS(1,2) have (A⊆0∧B⊆⋃T)∨(A⊆⋃T∧B⊆0) unfolding IsClosed˙def by auto
moreover
have 0∈T using empty˙open topSpaceAssum by auto
moreover
have ⋃T∈T using topSpaceAssum unfolding IsATopology˙def by auto
ultimately have ∃U∈T. ∃V∈T. A⊆U∧B⊆V∧U∩V=0 by auto
}
then show thesis unfolding IsNormal˙def by auto
qed

Every ultraconnected space is connected.

lemma UConn˙imp˙Conn:
assumes T–is ultraconnected″
shows T–is connected″
proof-
{
fix U V
assume U∈TU–is closed in″T
then have ⋃T-(⋃T-U)=U by auto
with ‹U∈T› have (⋃T-U)–is closed in″T unfolding IsClosed˙def by auto
with ‹U–is closed in″T› assms have U=0∨⋃T-U=0 unfolding IsUConnected˙def by auto
with ‹U∈T› have U=0∨U=⋃T by auto
}
then show thesis unfolding IsConnected˙def by auto
qed

lemma UConn˙spectrum:
shows (A–is in the spectrum of″IsUConnected) ⟷ A≲1

```
proof
  assume A˙spec:(A–is in the spectrum of"IsUConnected)
  {
    assume A=0
    then have A≲1 using empty˙lepollI by auto
  }
  moreover
  {
    assume A≠0
    from A˙spec have ∀T. (T–is a topology"∧⋃T≈A) ⟶ (T–is ultraconnected")
unfolding Spec˙def by auto
    moreover
    have Pow(A)–is a topology" using Pow˙is˙top by auto
    moreover
    have ⋃Pow(A)=A by auto
    then have ⋃Pow(A)≈A by auto
    ultimately have ult:Pow(A)–is ultraconnected" by auto
    moreover
    from ‹A≠0› obtain b where b∈A by auto
    then have –b"–is closed in"Pow(A) unfolding IsClosed˙def by auto
    {
      fix c
      assume c∈Ac≠b
      then have –c"–is closed in"Pow(A)–c"∩–b"=0 unfolding IsClosed˙def by auto
      with ult ‹–b"–is closed in"Pow(A)› have False using IsUConnected˙def by auto
    }
    with ‹b∈A› have A=–b" by auto
    then have A≈1 using singleton˙eqpoll˙1 by auto
    then have A≲1 using eqpoll˙imp˙lepoll by auto
  }
  ultimately show A≲1 by auto
next
  assume A≲1
  {
    fix T
    assume T–is a topology"⋃T≈A
    {
      assume ⋃T=0
      with ‹T–is a topology"› have T=–0" using empty˙open by auto
       then have T–is ultraconnected" unfolding IsUConnected˙def IsClosed˙def by
auto
    }
    moreover
    {
      assume ⋃T≠0
      moreover
      from ‹A≲1›‹⋃T≈A› have ⋃T≲1 using eq˙lepoll˙trans by auto
      ultimately
      obtain E where eq:⋃T=–E" using lepoll˙1˙is˙sing by blast
```

```
    moreover
    have T⊆Pow(⋃T) by auto
    ultimately have T⊆Pow(–E″) by auto
    then have T⊆–0,–E″″ by blast
    with ‹T–is a topology″› have –0″⊆T T⊆–0,–E″″ using empty˙open by auto
     then have T–is ultraconnected″ unfolding IsUConnected˙def IsClosed˙def by
(simp only: eq, safe, force)
   }
   ultimately have T–is ultraconnected″ by auto
  }
  then show A–is in the spectrum of″IsUConnected unfolding Spec˙def by auto
qed
```

This time, anti-ultraconnected is an old property.

```
theorem (in topology0) anti˙UConn:
  shows (T–is anti-″IsUConnected) ⟷ T–is T1″
proof
  assume T–is T1″
  {
   fix TT
   {
    assume TT–is a topology″TT–is T1″TT–is ultraconnected″
    {
     assume ⋃TT=0
     then have ⋃TT≲1 using empty˙lepollI by auto
   then have ((⋃TT)–is in the spectrum of″IsUConnected) using UConn˙spectrum
by auto
    }
    moreover
    {
     assume ⋃TT≠0
     then obtain t where t∈⋃TT by blast
     {
      fix x
      assume p:x∈⋃TT
      {
       fix y assume y∈(⋃TT)-–x″
        with ‹TT–is T1″› p obtain U where U∈TT y∈U x∉U using isT1˙def by
blast
       then have U∈TT y∈U U⊆(⋃TT)-–x″ by auto
       then have ∃U∈TT. y∈U ∧ U⊆(⋃TT)-–x″ by auto
      }
      then have ∀y∈(⋃TT)-–x″. ∃U∈TT. y∈U ∧ U⊆(⋃TT)-–x″ by auto
   with ‹TT–is a topology″› have ⋃TT-–x″∈TT using topology0.open˙neigh˙open
unfolding topology0˙def by auto
      with p have –x″ –is closed in″TT using IsClosed˙def by auto
     }
     then have reg:∀x∈⋃TT. –x″–is closed in″TT by auto
     with ‹t∈⋃TT› have t˙cl:–t″–is closed in″TT by auto
```

{
fix y
assume y∈⋃TT
with reg have –y″–is closed in″TT by auto
with ‹TT–is ultraconnected″› t˙cl have y=t unfolding IsUConnected˙def by auto
}
with ‹t∈⋃TT› have ⋃TT=–t″ by blast
then have ⋃TT≈1 using singleton˙eqpoll˙1 by auto
then have ⋃TT≲1 using eqpoll˙imp˙lepoll by auto
then have (⋃TT)–is in the spectrum of″IsUConnected using UConn˙spectrum by auto
}
ultimately have (⋃TT)–is in the spectrum of″IsUConnected by blast
}
then have (TT–is a topology″∧TT–is T_1″∧(TT–is ultraconnected″))⟶((⋃TT)–is in the spectrum of″IsUConnected)
by auto
}
then have ∀TT. (TT–is a topology″∧TT–is T_1″∧(TT–is ultraconnected″))⟶((⋃TT)–is in the spectrum of″IsUConnected)
by auto
moreover
note here˙T1
ultimately have ∀T. T–is a topology″ ⟶ ((T–is T_1″)⟶(T–is anti-″IsUConnected)) using Q˙P˙imp˙Spec[where Q=isT1 and P=IsUConnected]
by auto
with topSpaceAssum have (T–is T_1″)⟶(T–is anti-″IsUConnected) by auto
with ‹T–is T_1″› show T–is anti-″IsUConnected by auto
next
assume ASS:T–is anti-″IsUConnected
{
fix x y
assume x∈⋃Ty∈⋃Tx≠y
then have tot:⋃(T–restricted to″–x,y″)=–x,y″ unfolding RestrictedTo˙def by auto
{
assume AS:∀U∈T. x∈U⟶y∈U
{
assume –y″–is closed in″(T–restricted to″–x,y″)
moreover
from ‹x≠y› have –x,y″-–y″=–x″ by auto
ultimately have –x″∈(T–restricted to″–x,y″) unfolding IsClosed˙def by (simp only:tot)
then obtain U where U∈T–x″=–x,y″∩U unfolding RestrictedTo˙def by auto
moreover
with ‹x≠y› have y∉–x″ y∈–x,y″ by (blast+)
with ‹–x″=–x,y″∩U› have y∉U by auto
moreover have x∈–x″ by auto

```
      with ‹–x″=–x,y″∩U› have x∈U by auto
      ultimately have x∈Uy∉UU∈T by auto
      with AS have False by auto
    }
    then have y˙no˙cl:¬(–y″–is closed in″(T–restricted to″–x,y″)) by auto
    {
      fix A B
      assume cl:A–is closed in″(T–restricted to″–x,y″)B–is closed in″(T–restricted
to″–x,y″)A∩B=0
      with tot have A⊆–x,y″B⊆–x,y″A∩B=0 unfolding IsClosed˙def by auto
      then have x∈A⟶x∉By∈A⟶y∉BA⊆–x,y″B⊆–x,y″ by auto
      {
        assume x∈A
        with ‹x∈A⟶x∉B›‹B⊆–x,y″› have B⊆–y″ by auto
        then have B=0∨B=–y″ by auto
        with y˙no˙cl cl(2) have B=0 by auto
      }
      moreover
      {
        assume x∉A
        with ‹A⊆–x,y″› have A⊆–y″ by auto
        then have A=0∨A=–y″ by auto
        with y˙no˙cl cl(1) have A=0 by auto
      }
      ultimately have A=0∨B=0 by auto
    }
      then have (T–restricted to″–x,y″)–is ultraconnected″ unfolding IsUCon-
nected˙def by auto
      with ASS ‹x∈⋃T›‹y∈⋃T› have –x,y″–is in the spectrum of″IsUConnected
unfolding antiProperty˙def
      by auto
    then have –x,y″≲1 using UConn˙spectrum by auto
    moreover have x∈–x,y″ by auto
    ultimately have –x″=–x,y″ using lepoll˙1˙is˙sing[of –x,y″x] by auto
    moreover
    have y∈–x,y″ by auto
    ultimately have y∈–x″ by auto
    then have y=x by auto
    then have False using ‹x≠y› by auto
  }
  then have ∃U∈T. x∈U∧y∉U by auto
}
then show T–is T1″ unfolding isT1˙def by auto
qed
```

Is is natural that separation axioms and connection axioms are anti-properties of each other; as the concepts of connectedness and separation are opposite.

To end this section, let's try to charaterize anti-sober spaces.

```
lemma sober˙spectrum:
  shows (A–is in the spectrum of˝IsSober) ⟷ A≲1
proof
  assume AS:A–is in the spectrum of˝IsSober
  {
    assume A=0
    then have A≲1 using empty˙lepollI by auto
  }
  moreover
  {
    assume A≠0
    note AS
    moreover
    have top:–0,A˝–is a topology˝ unfolding IsATopology˙def by auto
    moreover
    have ⋃–0,A˝=A by auto
    then have ⋃–0,A˝≈A by auto
    ultimately have –0,A˝–is sober˝ using Spec˙def by auto
    moreover
    have –0,A˝–is hyperconnected˝ using Indiscrete˙HConn by auto
    moreover
    have –0,A˝–restricted to˝A=–0,A˝ unfolding RestrictedTo˙def by auto
    moreover
    have A–is closed in˝–0,A˝ unfolding IsClosed˙def by auto
    moreover
    note ‹A≠0›
   ultimately have ∃x∈A. A=Closure(–x˝,–0,A˝)∧ (∀y∈⋃–0, A˝. A = Closure(–y˝,
–0, A˝) ⟶ y = x) unfolding IsSober˙def by auto
   then obtain x where x∈A A=Closure(–x˝,–0,A˝) and reg:∀y∈A. A = Closure(–y˝,
–0, A˝) ⟶ y = x by auto
    {
      fix y assume y∈A
     with top have Closure(–y˝,–0,A˝)–is closed in˝–0,A˝ using topology0.cl˙is˙closed
        topology0˙def by auto
      moreover
      from ‹y∈A› top have y∈Closure(–y˝,–0,A˝) using topology0.cl˙contains˙set
        topology0˙def by auto
     ultimately have A-Closure(–y˝,–0,A˝)∈–0,A˝Closure(–y˝,–0,A˝)∩A≠0 unfold-
ing IsClosed˙def
        by auto
      then have A-Closure(–y˝,–0,A˝)=A∨A-Closure(–y˝,–0,A˝)=0
        by auto
      moreover
      from ‹y∈A›‹y∈Closure(–y˝,–0,A˝)› have y∈Ay∉A-Closure(–y˝,–0,A˝) by auto
       ultimately have A-Closure(–y˝,–0,A˝)=0 by (cases A-Closure(–y˝,–0,A˝)=A,
simp, auto)
      moreover
    from ‹y∈A› top have Closure(–y˝,–0,A˝)⊆A using topology0˙def topology0.Top˙3˙L11(1)
by blast
```

```
      then have A-(A-Closure(–y″,–0,A″))=Closure(–y″,–0,A″) by auto
      ultimately have A=Closure(–y″,–0,A″) by auto
    }
    with reg have ∀y∈A. x=y by auto
    with ‹x∈A› have A=–x″ by blast
    then have A≈1 using singleton˙eqpoll˙1 by auto
    then have A≲1 using eqpoll˙imp˙lepoll by auto
  }
  ultimately show A≲1 by auto
next
  assume A≲1
  {
    fix T assume T–is a topology″⋃T≈A
    {
      assume ⋃T=0
      then have T–is sober″ unfolding IsSober˙def by auto
    }
    moreover
    {
      assume ⋃T≠0
      then obtain x where x∈⋃T by blast
      moreover
      from ‹⋃T≈A› ‹A≲1› have ⋃T≲1 using eq˙lepoll˙trans by auto
      ultimately have ⋃T=–x″ using lepoll˙1˙is˙sing by auto
      moreover
      have T⊆Pow(⋃T) by auto
      ultimately have T⊆Pow(–x″) by auto
      then have T⊆–0,–x″″ by blast
      moreover
      from ‹T–is a topology″› have 0∈T using empty˙open by auto
      moreover
      from ‹T–is a topology″› have ⋃T∈T unfolding IsATopology˙def by auto
      with ‹⋃T=–x″› have –x″∈T by auto
      ultimately have T˙def:T=–0,–x″″ by auto
      then have dd:Pow(⋃T)-–0″=—x″″ by auto
      {
        fix B assume B∈Pow(⋃T)-–0″
        with dd have B˙def:B=–x″ by auto
          from ‹T–is a topology″› have (⋃T)–is closed in″T using topology0˙def
topology0.Top˙3˙L1
         by auto
         with ‹⋃T=–x″› ‹T–is a topology″› have Closure(–x″,T)=–x″ using topol-
ogy0.Top˙3˙L8
         unfolding topology0˙def by auto
        with B˙def have B=Closure(–x″,T) by auto
        moreover
        {
         fix y assume y∈⋃T
         with ‹⋃T=–x″› have y=x by auto
```

```
        }
      then have (∀y∈⋃T. B = Closure(–y″, T) ⟶ y = x) by auto
      moreover note ‹x∈⋃T›
     ultimately have (∃x∈⋃T. B = Closure(–x″, T) ∧ (∀y∈⋃T. B = Closure(–y″,
T) ⟶ y = x))
        by auto
    }
    then have T–is sober″ unfolding IsSober˙def by auto
  }
  ultimately have T–is sober″ by blast
 }
 then show A –is in the spectrum of″ IsSober unfolding Spec˙def by auto
qed

theorem (in topology0)anti˙sober:
 shows (T–is anti-″IsSober) ⟷ T=–0,⋃T″
proof
 assume T=–0,⋃T″
 {
  fix A assume A∈Pow(⋃T)(T–restricted to″A)–is sober″
  {
   assume A=0
   then have A≲1 using empty˙lepollI by auto
   then have A–is in the spectrum of″IsSober using sober˙spectrum by auto
  }
  moreover
  {
   assume A≠0
   have ⋃T∈–0,⋃T″0∈–0,⋃T″ by auto
   with ‹T=–0,⋃T″› have (⋃T)∈T 0∈T by auto
  with ‹A∈Pow(⋃T)› have –0,A″⊆(T–restricted to″A) unfolding RestrictedTo˙def
by auto
   moreover
   have ∀B∈–0,⋃T″. B=0∨B=⋃T by auto
   with ‹T=–0,⋃T″› have ∀B∈T. B=0∨B=⋃T by auto
  with ‹A∈Pow(⋃T)› have T–restricted to″A⊆–0,A″ unfolding RestrictedTo˙def
by auto
   ultimately have top˙def:T–restricted to″A=–0,A″ by auto
   moreover
   have A–is closed in″–0,A″ unfolding IsClosed˙def by auto
   moreover
   have –0,A″–is hyperconnected″ using Indiscrete˙HConn by auto
   moreover
    from ‹A∈Pow(⋃T)› have (T–restricted to″A)–restricted to″A=T–restricted
to″A using subspace˙of˙subspace[of AAT]
     by auto
   moreover
   note ‹A≠0› ‹A∈Pow(⋃T)›
  ultimately have A∈Pow(⋃(T–restricted to″A))-–0″A–is closed in″(T–restricted
```

to″A)((T–restricted to″A)–restricted to″A)–is hyperconnected″
by auto
with ‹(T–restricted to″A)–is sober″› have ∃x∈⋃(T–restricted to″A). A=Closure(–x″,T–restricted to″A)∧(∀y∈⋃(T–restricted to″A). A=Closure(–y″,T–restricted to″A) ⟶ y=x)
unfolding IsSober˙def by auto
with top˙def have ∃x∈A. A=Closure(–x″,–0,A″)∧(∀y∈A. A=Closure(–y″,–0,A″) ⟶ y=x) by auto
then obtain x where x∈AA=Closure(–x″,–0,A″)and reg:∀y∈A. A=Closure(–y″,–0,A″) ⟶ y=x by auto
{
fix y assume y∈A
from ‹A≠0› have top:–0,A″–is a topology″ using indiscrete˙ptopology[of A] indiscrete˙partition[of A] Ptopology˙is˙a˙topology(1)[of –A″A]
by auto
with ‹y∈A› have Closure(–y″,–0,A″)–is closed in″–0,A″ using topology0.cl˙is˙closed topology0˙def by auto
moreover
from ‹y∈A› top have y∈Closure(–y″,–0,A″) using topology0.cl˙contains˙set topology0˙def by auto
ultimately have A-Closure(–y″,–0,A″)∈–0,A″Closure(–y″,–0,A″)∩A≠0 unfolding IsClosed˙def
by auto
then have A-Closure(–y″,–0,A″)=A∨A-Closure(–y″,–0,A″)=0
by auto
moreover
from ‹y∈A›‹y∈Closure(–y″,–0,A″)› have y∈Ay∉A-Closure(–y″,–0,A″) by auto
ultimately have A-Closure(–y″,–0,A″)=0 by (cases A-Closure(–y″,–0,A″)=A, simp, auto)
moreover
from ‹y∈A› top have Closure(–y″,–0,A″)⊆A using topology0˙def topology0.Top˙3˙L11(1) by blast
then have A-(A-Closure(–y″,–0,A″))=Closure(–y″,–0,A″) by auto
ultimately have A=Closure(–y″,–0,A″) by auto
}
with reg ‹x∈A› have A=–x″ by blast
then have A≈1 using singleton˙eqpoll˙1 by auto
then have A≲1 using eqpoll˙imp˙lepoll by auto
then have A–is in the spectrum of″IsSober using sober˙spectrum by auto
}
ultimately have A–is in the spectrum of″IsSober by auto
}
then show T–is anti-″IsSober using antiProperty˙def by auto
next
assume T–is anti-″IsSober
{
fix A
assume A∈TA≠0A≠⋃T
then obtain x y where x∈Ay∈⋃T-A x≠yby blast
then have –x″=–x,y″∩A by auto

with ‹A∈T› have –x″∈T–restricted to″–x,y″ unfolding RestrictedTo˙def by auto
{
assume –y″∈T–restricted to″–x,y″
from ‹y∈⋃T-A› ‹x∈A›‹A∈T› have ⋃(T–restricted to″–x,y″)=–x,y″ unfolding RestrictedTo˙def
by auto
with ‹x≠y›‹–y″∈T–restricted to″–x,y″›‹–x″∈T–restricted to″–x,y″› have (T–restricted to″–x,y″)–is T_2″
unfolding isT2˙def by auto
then have (T–restricted to″–x,y″)–is sober″ using topology0.T2˙imp˙anti˙HConn[of T–restricted to″–x,y″]
Top˙1˙L4 topology0˙def topology0.anti˙HConn˙iff˙T1˙sober[of T–restricted to″–x,y″] by auto
}
moreover
{
assume –y″∉T–restricted to″–x,y″
moreover
from ‹y∈⋃T-A› ‹x∈A›‹A∈T› have T–restricted to″–x,y″⊆Pow(–x,y″) unfolding RestrictedTo˙def by auto
then have T–restricted to″–x,y″⊆–0,–x″,–y″,–x,y″″ by blast
moreover
note ‹–x″∈T–restricted to″–x,y″› empty˙open[OF Top˙1˙L4[of –x,y″]]
moreover
from ‹y∈⋃T-A› ‹x∈A›‹A∈T› have tot:⋃(T–restricted to″–x,y″)=–x,y″ unfolding RestrictedTo˙def
by auto
from Top˙1˙L4[of –x,y″] have ⋃(T–restricted to″–x,y″)∈T–restricted to″–x,y″ unfolding IsATopology˙def
by auto
with tot have –x,y″∈T–restricted to″–x,y″ by auto
ultimately have top˙d˙def:T–restricted to″–x,y″=–0,–x″,–x,y″″ by auto
{
fix B assume B∈Pow(–x,y″)-–0″B–is closed in″(T–restricted to″–x,y″)
with top˙d˙def have (⋃(T–restricted to″–x,y″))-B∈–0,–x″,–x,y″″ unfolding IsClosed˙def by simp
moreover have B∈—x″,–y″,–x,y″″ using ‹B∈Pow(–x,y″)-–0″› by blast
moreover note tot
ultimately have –x,y″-B∈–0,–x″,–x,y″″ by auto
have xin:x∈Closure(–x″,T–restricted to″–x,y″) using topology0.cl˙contains˙set[of T–restricted to″–x,y″–x″]
Top˙1˙L4[of –x,y″] unfolding topology0˙def[of (T –restricted to″ –x, y″)] using tot by auto
{
assume –x″–is closed in″(T–restricted to″–x,y″)
then have –x,y″-–x″∈(T–restricted to″–x,y″) unfolding IsClosed˙def using tot
by auto
moreover

```
        from ‹x≠y› have –x,y″-–x″=–y″ by auto
        ultimately have –y″∈(T–restricted to″–x,y″) by auto
        then have False using ‹–y″∉(T–restricted to″–x,y″)› by auto
      }
      then have ¬(–x″–is closed in″(T–restricted to″–x,y″)) by auto
      moreover
     from tot have (Closure(–x″,T–restricted to″–x,y″))–is closed in″(T–restricted
to″–x,y″)
       using topology0.cl˙is˙closed unfolding topology0˙def using Top˙1˙L4[of –x,y″]
        tot by auto
      ultimately have ¬(Closure(–x″,T–restricted to″–x,y″)=–x″) by auto
      moreover note xin topology0.Top˙3˙L11(1)[of T–restricted to″–x,y″–x″] tot
         ultimately have cl˙x:Closure(–x″,T–restricted to″–x,y″)=–x,y″ unfolding
topology0˙def
        using Top˙1˙L4[of –x,y″] by auto
       have –y″–is closed in″(T–restricted to″–x,y″) unfolding IsClosed˙def using
tot
        top˙d˙def ‹x≠y› by auto
    then have cl˙y:Closure(–y″,T–restricted to″–x,y″)=–y″ using topology0.Top˙3˙L8[of
T–restricted to″–x,y″]
        unfolding topology0˙def using Top˙1˙L4[of –x,y″] tot by auto
      {
        assume –x,y″-B=0
        with ‹B∈Pow(–x,y″)-–0″› have B:–x,y″=B by auto
        {
          fix m
          assume dis:m∈–x,y″ and B˙def:B=Closure(–m″,T–restricted to″–x,y″)
          {
            assume m=y
            with B˙def have B=Closure(–y″,T–restricted to″–x,y″) by auto
            with cl˙y have B=–y″ by auto
            with B have –x,y″=–y″ by auto
            moreover have x∈–x,y″ by auto
            ultimately
            have x∈–y″ by auto
            with ‹x≠y› have False by auto
          }
          with dis have m=x by auto
        }
       then have (∀m∈–x,y″. B=Closure(–m″,T–restricted to″–x,y″)⟶m=x ) by
auto
        moreover
        have B=Closure(–x″,T–restricted to″–x,y″) using cl˙x B by auto
     ultimately have ∃t∈–x,y″. B=Closure(–t″,T–restricted to″–x,y″) ∧ (∀m∈–x,y″.
B=Closure(–m″,T–restricted to″–x,y″)⟶m=t )
          by auto
      }
      moreover
      {
```

```
assume –x,y″-B≠0
with ‹–x,y″-B∈–0,–x″,–x,y″″› have or:–x,y″-B=–x″∨–x,y″-B=–x,y″ by auto
{
  assume –x,y″-B=–x″
  then have x∈–x,y″-B by auto
  with ‹B∈—x″,–y″,–x,y″″› ‹x≠y› have B:B=–y″ by blast
  {
    fix m
    assume dis:m∈–x,y″ and B˙def:B=Closure(–m″,T–restricted to″–x,y″)
    {
      assume m=x
      with B˙def have B=Closure(–x″,T–restricted to″–x,y″) by auto
      with cl˙x have B=–x,y″ by auto
      with B have –x,y″=–y″ by auto
      moreover have x∈–x,y″ by auto
      ultimately
      have x∈–y″ by auto
      with ‹x≠y› have False by auto
    }
    with dis have m=y by auto
  }
  moreover
  have B=Closure(–y″,T–restricted to″–x,y″) using cl˙y B by auto
      ultimately have ∃t∈–x,y″. B=Closure(–t″,T–restricted to″–x,y″) ∧
(∀m∈–x,y″. B=Closure(–m″,T–restricted to″–x,y″)⟶m=t )
    by auto
}
moreover
{
  assume –x,y″-B≠–x″
  with or have –x,y″-B=–x,y″ by auto
  then have x∈–x,y″-By∈–x,y″-B by auto
  with ‹B∈—x″,–y″,–x,y″″› ‹x≠y› have False by auto
}
ultimately have ∃t∈–x,y″. B=Closure(–t″,T–restricted to″–x,y″) ∧ (∀m∈–x,y″.
B=Closure(–m″,T–restricted to″–x,y″)⟶m=t )
  by auto
}
ultimately have ∃t∈–x,y″. B=Closure(–t″,T–restricted to″–x,y″) ∧ (∀m∈–x,y″.
B=Closure(–m″,T–restricted to″–x,y″)⟶m=t )
  by auto
}
then have (T–restricted to″–x,y″)–is sober″ unfolding IsSober˙def using tot by
auto
}
ultimately have (T–restricted to″–x,y″)–is sober″ by auto
with ‹T–is anti-″IsSober› have –x,y″–is in the spectrum of″IsSober unfolding
antiProperty˙def
using ‹x∈A›‹A∈T›‹y∈⋃T-A› by auto
```

then have –x,y″≲1 using sober˙spectrum by auto
moreover
have x∈–x,y″ by auto
ultimately have –x,y″=–x″ using lepoll˙1˙is˙sing[of –x,y″x] by auto
moreover have y∈–x,y″ by auto
ultimately have y∈–x″ by auto
then have False using ‹x≠y› by auto
}
then have T⊆–0,⋃T″ by auto
with empty˙open[OF topSpaceAssum] topSpaceAssum show T=–0,⋃T″ unfolding IsATopology˙def
by auto
qed

end

63 Topology 8

theory Topology˙ZF˙8 imports Topology˙ZF˙6 EquivClass1
begin

This theory deals with quotient topologies.

63.1 Definition of quotient topology

Given a surjective function $f : X \to Y$ and a topology τ in X, it is posible to consider a special topology in Y. f is called quotient function.

definition(in topology0)
QuotientTop (–quotient topology in″˙–by″˙ 80)
where f∈surj(⋃T,Y) ⟹–quotient topology in″Y–by″f≡
–U∈Pow(Y). f-U∈T″

abbreviation QuotientTopTop (–quotient topology in″˙–by″˙–from″˙)
where QuotientTopTop(Y,f,T) ≡ topology0.QuotientTop(T,Y,f)

The quotient topology is indeed a topology.

theorem(in topology0) quotientTop˙is˙top:
assumes f∈surj(⋃T,Y)
shows (–quotient topology in″ Y –by″ f) –is a topology″
proof-
have (–quotient topology in″ Y –by″ f)=–U ∈ Pow(Y) . f - U ∈ T″ using QuotientTop˙def assms
by auto moreover
{
fix M x B assume M:M ⊆ –U ∈ Pow(Y) . f - U ∈ T″
then have ⋃M⊆Y by blast moreover
have A1:f - (⋃M)=(⋃y∈(⋃M). f-–y″) using vimage˙eq˙UN by blast

```
    {
      fix A assume A∈M
      with M have A∈Pow(Y) f - A∈T by auto
      have f - A=(⋃y∈A. f-–y″) using vimage˙eq˙UN by blast
    }
    then have (⋃A∈M. f- A)=(⋃A∈M. (⋃y∈A. f-–y″)) by auto
    then have (⋃A∈M. f- A)=(⋃y∈⋃M. f-–y″) by auto
    with A1 have A2:f - (⋃M)=⋃–f- A. A∈M″ by auto
    {
      fix A assume A∈M
      with M have f - A∈T by auto
    }
    then have ∀A∈M. f - A∈T by auto
    then have –f- A. A∈M″⊆T by auto
    then have (⋃–f- A. A∈M″)∈T using topSpaceAssum unfolding IsATopology˙def
by auto
    with A2 have (f - (⋃M))∈T by auto
    ultimately have ⋃M∈–U∈Pow(Y). f-U∈T″ by auto
  }
  moreover
  {
    fix U V assume U∈–U∈Pow(Y). f-U∈T″V∈–U∈Pow(Y). f-U∈T″
    then have U∈Pow(Y)V∈Pow(Y)f-U∈Tf-V∈T by auto
     then have (f-U)∩(f-V)∈T using topSpaceAssum unfolding IsATopology˙def by
auto
    then have f- (U∩V)∈T using invim˙inter˙inter˙invim assms unfolding surj˙def
      by auto
    with ‹U∈Pow(Y)›‹V∈Pow(Y)› have U∩V∈–U∈Pow(Y). f-U∈T″ by auto
  }
  ultimately show thesis using IsATopology˙def by auto
qed
```

The quotient function is continuous.

```
lemma (in topology0) quotient˙func˙cont:
  assumes f∈surj(⋃T,Y)
  shows IsContinuous(T,(–quotient topology in″ Y –by″ f),f)
    unfolding IsContinuous˙def using QuotientTop˙def assms by auto
```

One of the important properties of this topology, is that a function from the quotient space is continuous iff the composition with the quotient function is continuous.

```
theorem(in two˙top˙spaces0) cont˙quotient˙top:
  assumes h∈surj(⋃τ1,Y) g:Y→⋃τ2 IsContinuous(τ1,τ2,g O h)
  shows IsContinuous((–quotient topology in″ Y –by″ h –from″ τ1),τ2,g)
proof-
  {
    fix U assume U∈τ2
    with assms(3) have (g O h)-(U)∈τ1 unfolding IsContinuous˙def by auto
    then have h-(g-(U))∈τ1 using vimage˙comp by auto
```

then have g-(U)∈(–quotient topology in″ Y –by″ h –from″ τ_1) using topology0.QuotientTop˙def
tau1˙is˙top assms(1) using func1˙1˙L3 assms(2) unfolding topology0˙def by auto
}
then show thesis unfolding IsContinuous˙def by auto
qed

The underlying set of the quotient topology is Y.

lemma(in topology0) total˙quo˙func:
assumes f∈surj(⋃T,Y)
shows (⋃(–quotient topology in″Y–by″f))=Y
proof-
from assms have f-Y=⋃T using func1˙1˙L4 unfolding surj˙def by auto moreover
have ⋃T∈T using topSpaceAssum unfolding IsATopology˙def by auto ultimately
have Y∈(–quotient topology in″Y–by″f–from″T) using QuotientTop˙def assms by auto
then show thesis using QuotientTop˙def assms by auto
qed

63.2 Quotient topologies from equivalence relations

In this section we will show that the quotient topologies come from an equivalence relation.

First, some lemmas for relations.

lemma quotient˙proj˙fun:
shows –⟨b,r–b″⟩. b∈A″:A→A//r unfolding Pi˙def function˙def domain˙def
unfolding quotient˙def by auto

lemma quotient˙proj˙surj:
shows –⟨b,r–b″⟩. b∈A″∈surj(A,A//r)
proof-
{
fix y assume y∈A//r
then obtain yy where A:yy∈A y=r–yy″ unfolding quotient˙def by auto
then have ⟨yy,y⟩∈–⟨b,r–b″⟩. b∈A″ by auto
then have –⟨b,r–b″⟩. b∈A″yy=y using apply˙equality[OF ˙ quotient˙proj˙fun] by auto
with A(1) have ∃yy∈A. –⟨b,r–b″⟩. b∈A″yy=y by auto
}
with quotient˙proj˙fun show thesis unfolding surj˙def by auto
qed

lemma preim˙equi˙proj:
assumes U⊆A//r equiv(A,r)
shows –⟨b,r–b″⟩. b∈A″-U=⋃U
proof
{
fix y assume y∈⋃U

```
    then obtain V where V:y∈VV∈U by auto
    with ‹U⊆(A//r)› have y∈A using EquivClass˙1˙L1 assms(2) by auto moreover
    from ‹U⊆(A//r)› V have r–y″=V using EquivClass˙1˙L2 assms(2) by auto
    moreover note V(2) ultimately have y∈–x∈A. r–x″∈U″ by auto
    then have y∈–⟨b,r–b″⟩. b∈A″-U by auto
  }
  then show ⋃U⊆–⟨b,r–b″⟩. b∈A″-U by blast moreover
  {
    fix y assume y∈–⟨b,r–b″⟩. b∈A″-U
    then have yy:y∈–x∈A. r–x″∈U″ by auto
    then have r–y″∈U by auto moreover
    from yy have y∈r–y″ using assms equiv˙class˙self by auto ultimately
    have y∈⋃U by auto
  }
  then show –⟨b,r–b″⟩. b∈A″-U⊆⋃U by blast
qed
```

Now we define what a quotient topology from an equivalence relation is:

```
definition(in topology0)
  EquivQuo (–quotient by″˙ 70)
 where equiv(⋃T,r)⟹(–quotient by″r)≡–quotient topology in″(⋃T)//r–by″–⟨b,r–b″⟩.
b∈⋃T″

abbreviation
  EquivQuoTop (˙–quotient by″˙ 60)
  where EquivQuoTop(T,r)≡topology0.EquivQuo(T,r)
```

First, another description of the topology (more intuitive):

```
theorem (in topology0) quotient˙equiv˙rel:
  assumes equiv(⋃T,r)
  shows (–quotient by″r)=–U∈Pow((⋃T)//r). ⋃U∈T″
proof-
 have (–quotient topology in″(⋃T)//r–by″–⟨b,r–b″⟩. b∈⋃T″)=–U∈Pow((⋃T)//r).
–⟨b,r–b″⟩. b∈⋃T″-U∈T″
    using QuotientTop˙def quotient˙proj˙surj by auto moreover
 have –U∈Pow((⋃T)//r). –⟨b,r–b″⟩. b∈⋃T″-U∈T″=–U∈Pow((⋃T)//r). ⋃U∈T″
    proof
     {
       fix U assume U∈–U∈Pow((⋃T)//r). –⟨b,r–b″⟩. b∈⋃T″-U∈T″
         then have U∈–U∈Pow((⋃T)//r). ⋃U∈T″ using preim˙equi˙proj assms by
auto
     }
     then show –U∈Pow((⋃T)//r). –⟨b,r–b″⟩. b∈⋃T″-U∈T″⊆–U∈Pow((⋃T)//r).
⋃U∈T″ by auto
     {
       fix U assume U∈–U∈Pow((⋃T)//r). ⋃U∈T″
     then have U∈–U∈Pow((⋃T)//r). –⟨b,r–b″⟩. b∈⋃T″-U∈T″ using preim˙equi˙proj
assms by auto
     }
```

```
  then show –U∈Pow((⋃T)//r). ⋃U∈T″⊆–U∈Pow((⋃T)//r). –⟨b,r–b″⟩. b∈⋃T″-U∈T″
by auto
  qed
 ultimately show thesis using EquivQuo˙def assms by auto
qed
```

We apply previous results to this topology.

```
theorem(in topology0) total˙quo˙equi:
 assumes equiv(⋃T,r)
 shows ⋃(–quotient by″r)=(⋃T)//r
 using total˙quo˙func quotient˙proj˙surj EquivQuo˙def assms by auto

theorem(in topology0) equiv˙quo˙is˙top:
 assumes equiv(⋃T,r)
 shows (–quotient by″r)–is a topology″
 using quotientTop˙is˙top quotient˙proj˙surj EquivQuo˙def assms by auto
```

MAIN RESULT: All quotient topologies arise from an equivalence relation given by the quotient function $f : X \to Y$. This means that any quotient topology is homeomorphic to a topology given by an equivalence relation quotient.

```
theorem(in topology0) equiv˙quotient˙top:
 assumes f∈surj(⋃T,Y)
 defines r≡–⟨x,y⟩∈⋃T×⋃T. f(x)=f(y)″
 defines g≡–⟨y,f-–y″⟩. y∈Y″
 shows equiv(⋃T,r) and IsAhomeomorphism((–quotient topology in″Y–by″f),(–quotient
by″r),g)
proof-
 have ff:f:⋃T→Y using assms(1) unfolding surj˙def by auto
 show B:equiv(⋃T,r) unfolding equiv˙def refl˙def sym˙def trans˙def unfolding r˙def
by auto
 have gg:g:Y→((⋃T)//r)
  proof-
   {
    fix B assume B∈g
    then obtain y where Y:y∈Y B=⟨y,f-–y″⟩ unfolding g˙def by auto
    then have f-–y″⊆⋃T using func1˙1˙L3 ff by blast
    then have eq:f-–y″=–x∈⋃T. ⟨x,y⟩∈f″ using vimage˙iff by auto
   from Y obtain A where A1:A∈⋃TfA=y using assms(1) unfolding surj˙def by
blast
    with eq have A:A∈f-–y″ using apply˙Pair[OF ff] by auto
    {
     fix t assume t∈f-–y″
     with A have t∈⋃TA∈⋃T⟨t,y⟩∈f⟨A,y⟩∈f using eq by auto
     then have ft=fA using apply˙equality assms(1) unfolding surj˙def by auto
     with ‹t∈⋃T›‹A∈⋃T› have ⟨A,t⟩∈r using r˙def by auto
     then have t∈r–A″ using image˙iff by auto
    }
    then have f-–y″⊆r–A″ by auto moreover
```

```
      {
        fix t assume t∈r–A″
        then have ⟨A,t⟩∈r using image˙iff by auto
       then have un:t∈⋃TA∈⋃T and eq2:ft=fA unfolding r˙def by auto moreover
        from un have ⟨t,ft⟩∈f using apply˙Pair[OF ff] by auto
        with eq2 A1 have ⟨t,y⟩∈f by auto
        with un have t∈f-–y″ using eq by auto
      }
      then have r–A″⊆f-–y″ by auto ultimately
      have f-–y″=r–A″ by auto
      then have f-–y″∈ (⋃T)//r using A1(1) unfolding quotient˙def by auto
      with Y have B∈Y×(⋃T)//r by auto
    }
    then have ∀A∈g. A∈ Y×(⋃T)//r by auto
    then have g⊆(Y×(⋃T)//r) by auto moreover
    then show thesis unfolding Pi˙def function˙def domain˙def g˙def by auto
  qed
then have gg2:g:Y→(⋃(–quotient by″r)) using total˙quo˙equi B by auto
{
  fix s assume S:s∈(–quotient topology in″Y–by″f)
 then have s∈Pow(Y)and P:f-s∈T using QuotientTop˙def topSpaceAssum assms(1)
    by auto
  have f-s=(⋃y∈s. f-–y″) using vimage˙eq˙UN by blast moreover
  from ‹s∈Pow(Y)› have ∀y∈s. ⟨y,f-–y″⟩∈g unfolding g˙def by auto
  then have ∀y∈s. gy=f-–y″ using apply˙equality gg by auto ultimately
  have f-s=(⋃y∈s. gy) by auto
  with P have (⋃y∈s. gy)∈T by auto moreover
  from ‹s∈Pow(Y)› have ∀y∈s. gy∈(⋃T)//r using apply˙type gg by auto
  ultimately have –gy. y∈s″∈(–quotient by″r) using quotient˙equiv˙rel B by auto
  with ‹s∈Pow(Y)› have gs∈(–quotient by″r) using func˙imagedef gg by auto
}
 then have gopen:∀s∈(–quotient topology in″Y–by″f). gs∈(T–quotient by″r) by
auto
have pr˙fun:–⟨b,r–b″⟩. b∈⋃T″:⋃T→(⋃T)//r using quotient˙proj˙fun by auto
{
  fix b assume b:b∈⋃T
  have bY:fb∈Y using apply˙funtype ff b by auto
  with b have com:(g O f)b=g(fb) using comp˙fun˙apply ff by auto
  from bY have pg:⟨fb,f-(–fb″)⟩∈g unfolding g˙def by auto
  then have g(fb)=f-(–fb″) using apply˙equality gg by auto
  with com have comeq:(g O f)b=f-(–fb″) by auto
  from b have A:f–b″=–fb″ –b″⊆⋃T using func˙imagedef ff by auto
  from A(2) have b∈f - (f  –b″) using func1˙1˙L9 ff by blast
  then have b∈f-(–fb″) using A(1) by auto moreover
  from pg have f-(–fb″)∈(⋃T)//r using gg unfolding Pi˙def by auto
  ultimately have r–b″=f-(–fb″) using EquivClass˙1˙L2 B by auto
  then have (g O f)b=r–b″ using comeq by auto moreover
  from b have ⟨b,r–b″⟩∈–⟨b,r–b″⟩. b∈⋃T″ by auto
   with pr˙fun have –⟨b,r–b″⟩. b∈⋃T″b=r–b″ using apply˙equality by auto ulti-
```

mately
have (g O f)b=–⟨b,r–b″⟩. b∈⋃T″b by auto
}
then have reg:∀b∈⋃T. (g O f)b=–⟨b,r–b″⟩. b∈⋃T″b by auto moreover
have compp:g O f∈⋃T→(⋃T)//r using comp˙fun ff gg by auto
have feq:(g O f)=–⟨b,r–b″⟩. b∈⋃T″ using fun˙extension[OF compp pr˙fun] reg by auto
then have IsContinuous(T,–quotient by″r,(g O f)) using quotient˙func˙cont quotient˙proj˙surj
EquivQuo˙def topSpaceAssum B by auto moreover
have (g O f):⋃T→⋃(–quotient by″r) using comp˙fun ff gg2 by auto
ultimately have gcont:IsContinuous(–quotient topology in″Y–by″f,–quotient by″r,g)
using two˙top˙spaces0.cont˙quotient˙top assms(1) gg2 unfolding two˙top˙spaces0˙def
using topSpaceAssum equiv˙quo˙is˙top B by auto
{
fix x y assume T:x∈Yy∈Ygx=gy
then have f-–x″=f-–y″ using apply˙equality gg unfolding g˙def by auto
then have f(f-–x″)=f(f-–y″) by auto
with T(1,2) have –x″=–y″ using surj˙image˙vimage assms(1) by auto
then have x=y by auto
}
with gg2 have g∈inj(Y,⋃(–quotient by″r)) unfolding inj˙def by auto moreover
have g O f∈surj(⋃T, (⋃T)//r) using feq quotient˙proj˙surj by auto
then have g∈surj(Y,(⋃T)//r) using comp˙mem˙surjD1 ff gg by auto
then have g∈surj(Y,⋃(T–quotient by″r)) using total˙quo˙equi B by auto
ultimately have g∈bij(⋃(–quotient topology in″Y–by″f),⋃(–quotient by″r)) unfolding bij˙def using total˙quo˙func assms(1) by auto
with gcont gopen show IsAhomeomorphism((–quotient topology in″Y–by″f),(–quotient by″r),g)
using bij˙cont˙open˙homeo by auto
qed

lemma product˙equiv˙rel˙fun:
shows –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″:(⋃T×⋃T)→((⋃T)//r×(⋃T)//r)
proof-
have –⟨b,r–b″⟩. b∈⋃T″∈⋃T→(⋃T)//r using quotient˙proj˙fun by auto moreover
have ∀A∈⋃T. ⟨A,r–A″⟩∈–⟨b,r–b″⟩. b∈⋃T″ by auto
ultimately have ∀A∈⋃T. –⟨b,r–b″⟩. b∈⋃T″A=r–A″ using apply˙equality by auto
then have IN: –⟨⟨b, c⟩, r –b″, r –c″⟩ . ⟨b,c⟩ ∈ ⋃T × ⋃T″= –⟨⟨x, y⟩, –⟨b, r –b″⟩ . b ∈ ⋃T″ x, –⟨b, r –b″⟩ . b ∈ ⋃T″ y⟩ . ⟨x,y⟩ ∈ ⋃T × ⋃T″
by force
then show thesis using prod˙fun quotient˙proj˙fun by auto
qed

lemma(in topology0) prod˙equiv˙rel˙surj:
shows –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″:surj(⋃(ProductTopology(T,T)),((⋃T)//r×(⋃T)//r))
proof-
have fun:–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″:(⋃T×⋃T)→((⋃T)//r×(⋃T)//r)
using

product˙equiv˙rel˙fun by auto moreover
{
fix M assume M∈((⋃T)//r×(⋃T)//r)
then obtain M1 M2 where M:M=⟨M1,M2⟩ M1∈(⋃T)//rM2∈(⋃T)//r by auto
then obtain m1 m2 where m:m1∈⋃Tm2∈⋃TM1=r–m1″M2=r–m2″ unfolding quotient˙def
by auto
then have mm:⟨m1,m2⟩∈(⋃T×⋃T) by auto
then have ⟨⟨m1,m2⟩,⟨r–m1″,r–m2″⟩⟩∈–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″ by auto
then have –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″⟨m1,m2⟩=⟨r–m1″,r–m2″⟩
using apply˙equality fun by auto
then have –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″⟨m1,m2⟩=M using M(1) m(3,4) by auto
then have ∃R∈(⋃T×⋃T). –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″R=M using mm by auto
}
ultimately show thesis unfolding surj˙def using Top˙1˙4˙T1(3) topSpaceAssum by auto
qed

lemma(in topology0) product˙quo˙fun:
assumes equiv(⋃T,r)
shows IsContinuous(ProductTopology(T,T),ProductTopology(–quotient by″r,(–quotient by″r)),–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″)
proof-
have –⟨b,r–b″⟩. b∈⋃T″:⋃T→(⋃T)//r using quotient˙proj˙fun by auto moreover
have ∀A∈⋃T. ⟨A,r–A″⟩∈–⟨b,r–b″⟩. b∈⋃T″ by auto ultimately
have ∀A∈⋃T. –⟨b,r–b″⟩. b∈⋃T″A=r–A″ using apply˙equality by auto
then have IN: –⟨⟨b, c⟩, r –b″, r –c″⟩ . ⟨b,c⟩ ∈ ⋃T × ⋃T″= –⟨⟨x, y⟩, –⟨b, r –b″⟩ . b ∈ ⋃T″ x, –⟨b, r –b″⟩ . b ∈ ⋃T″ y⟩ . ⟨x,y⟩ ∈ ⋃T × ⋃T″
by force
have cont:IsContinuous(T,–quotient by″r,–⟨b,r–b″⟩. b∈⋃T″) using quotient˙func˙cont quotient˙proj˙surj
EquivQuo˙def assms by auto
have tot:⋃(T–quotient by″r) = (⋃T) // r and top:(–quotient by″r) –is a topology″ using total˙quo˙equi equiv˙quo˙is˙top assms by auto
then have fun:–⟨b,r–b″⟩. b∈⋃T″:⋃T→⋃(–quotient by″r) using quotient˙proj˙fun by auto
then have two:two˙top˙spaces0(T,–quotient by″r,–⟨b,r–b″⟩. b∈⋃T″) unfolding two˙top˙spaces0˙def using topSpaceAssum top by auto
show thesis using two˙top˙spaces0.product˙cont˙functions two fun fun cont cont top topSpaceAssum IN by auto
qed

The product of quotient topologies is a quotient topology given that the quotient map is open. This isn't true in general.

theorem(in topology0) prod˙quotient:
assumes equiv(⋃T,r) ∀A∈T. –⟨b,r–b″⟩. b∈⋃T″A∈(–quotient by″r)

shows (ProductTopology(–quotient by″r,–quotient by″r)) = (–quotient topology in″(((⋃T)//r)×((⋃T)//r))–by″(–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″)–from″(ProductTopology(T,T)))
proof
{
fix A assume A:A∈ProductTopology(–quotient by″r,–quotient by″r)
from assms have IsContinuous(ProductTopology(T,T),ProductTopology(–quotient by″r,(–quotient by″r)),–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″) using product˙quo˙fun
by auto
with A have –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″-A∈ProductTopology(T,T)
unfolding IsContinuous˙def by auto moreover
from A have A⊆⋃ProductTopology(T–quotient by″r,T–quotient by″r) by auto
then have A⊆⋃(T–quotient by″r)×⋃(T–quotient by″r) using Top˙1˙4˙T1(3) equiv˙quo˙is˙top equiv˙quo˙is˙top
using assms by auto
then have A∈Pow(((⋃T)//r)×((⋃T)//r)) using total˙quo˙equi assms by auto
ultimately have A∈(–quotient topology in″(((⋃T)//r)×((⋃T)//r))–by″–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″–from″(ProductTopology(T,T)))
using topology0.QuotientTop˙def Top˙1˙4˙T1(1) topSpaceAssum prod˙equiv˙rel˙surj assms(1) unfolding topology0˙def by auto
}
then show ProductTopology(T–quotient by″r,T–quotient by″r)⊆(–quotient topology in″(((⋃T)//r)×((⋃T)//r))–by″–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″–from″(ProductTopology(T,T)))
by auto
{
fix A assume A∈(–quotient topology in″(((⋃T)//r)×((⋃T)//r))–by″–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″–from″(ProductTopology(T,T)))
then have A:A⊆(((⋃T)//r)×((⋃T)//r) –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″-A∈ProductTopology(T,T)
using topology0.QuotientTop˙def Top˙1˙4˙T1(1) topSpaceAssum prod˙equiv˙rel˙surj assms(1) unfolding topology0˙def by auto
{
fix CC assume CC∈A
with A(1) obtain C1 C2 where CC:CC=⟨C1,C2⟩ C1∈(((⋃T)//r)C2∈(((⋃T)//r) by auto
then obtain c1 c2 where CC1:c1∈⋃Tc2∈⋃T and CC2:C1=r–c1″C2=r–c2″ unfolding quotient˙def
by auto
then have ⟨c1,c2⟩∈⋃T×⋃T by auto
then have ⟨⟨c1,c2⟩,⟨r–c1″,r–c2″⟩⟩∈–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″ by auto
with CC2 CC have ⟨⟨c1,c2⟩,CC⟩∈–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″ by auto
with ‹CC∈A› have ⟨c1,c2⟩∈–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″-A
using vimage˙iff by auto
with A(2) have ∃V W. V ∈ T ∧ W ∈ T ∧ V × W ⊆ –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″-A ∧ ⟨c1,c2⟩ ∈ V × W
using prod˙top˙point˙neighb topSpaceAssum by blast
then obtain V W where VW:V∈TW∈TV × W ⊆ –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″-Ac1∈Vc2∈W by auto
with assms(2) have –⟨b,r–b″⟩. b∈⋃T″V∈(T–quotient by″r)–⟨b,r–b″⟩. b∈⋃T″W∈(T–quotient by″r) by auto

then have P:–⟨b,r–b″⟩. b∈⋃T″V×–⟨b,r–b″⟩. b∈⋃T″W∈ProductTopology(T–quotient by″r,T–quotient by″r) using prod˙open˙open˙prod equiv˙quo˙is˙top
assms(1) by auto
{
fix S assume S∈–⟨b,r–b″⟩. b∈⋃T″V×–⟨b,r–b″⟩. b∈⋃T″W
then obtain s1 s2 where S:S=⟨s1,s2⟩s1∈–⟨b,r–b″⟩. b∈⋃T″Vs2∈–⟨b,r–b″⟩. b∈⋃T″W by blast
then obtain t1 t2 where T:⟨t1,s1⟩∈–⟨b,r–b″⟩. b∈⋃T″⟨t2,s2⟩∈–⟨b,r–b″⟩. b∈⋃T″t1∈Vt2∈W using image˙iff by auto
then have ⟨t1,t2⟩∈V×W by auto
with VW(3) have ⟨t1,t2⟩∈–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″-A by auto
then have ∃SS∈A. ⟨⟨t1,t2⟩,SS⟩∈–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″ using vimage˙iff by auto
then obtain SS where SS∈A⟨⟨t1,t2⟩,SS⟩∈–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″ by auto moreover
from T VW(1,2) have ⟨t1,t2⟩∈⋃T×⋃T⟨s1,s2⟩=⟨r–t1″,r–t2″⟩ by auto
with S(1) have ⟨⟨t1,t2⟩,S⟩∈–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″ by auto
ultimately have S∈A using product˙equiv˙rel˙fun unfolding Pi˙def function˙def by auto
}
then have sub:–⟨b,r–b″⟩. b∈⋃T″V×–⟨b,r–b″⟩. b∈⋃T″W⊆A by blast
have ⟨c1,C1⟩∈–⟨b,r–b″⟩. b∈⋃T″⟨c2,C2⟩∈–⟨b,r–b″⟩. b∈⋃T″ using CC2 CC1 by auto
with ‹c1∈V›‹c2∈W› have C1∈–⟨b,r–b″⟩. b∈⋃T″VC2∈–⟨b,r–b″⟩. b∈⋃T″W using image˙iff by auto
then have CC∈–⟨b,r–b″⟩. b∈⋃T″V×–⟨b,r–b″⟩. b∈⋃T″W using CC by auto
with sub P have ∃OO∈ProductTopology(T–quotient by″r,T–quotient by″r). CC∈OO∧ OO⊆A
using exI[where x=–⟨b,r–b″⟩. b∈⋃T″V×–⟨b,r–b″⟩. b∈⋃T″W and P=λOO. OO∈ProductTopology(T–quotient by″r,T–quotient by″r)∧ CC∈OO∧ OO⊆A]
by auto
}
then have ∀C∈A. ∃OO∈ProductTopology(T–quotient by″r,T–quotient by″r). C∈OO∧ OO⊆A by auto
then have A∈ProductTopology(T–quotient by″r,T–quotient by″r) using topology0.open˙neigh˙open
unfolding topology0˙def using Top˙1˙4˙T1 equiv˙quo˙is˙top assms by auto
}
then show (–quotient topology in″(((⋃T)//r)×((⋃T)//r))–by″–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″–from″(ProductTopology(T,T)))⊆ProductTopology(T–quotient by″r,T–quotient by″r)
by auto
qed

end

64 Topology 9

theory Topology˙ZF˙9

imports Topology˙ZF˙2 Group˙ZF˙2 Topology˙ZF˙7 Topology˙ZF˙8
begin

64.1 Group of homeomorphisms

This theory file deals with the fact the set homeomorphisms of a topological space into itself forms a group.

First, we define the set of homeomorphisms.

definition
HomeoG(T) ≡ –f:⋃T→⋃T. IsAhomeomorphism(T,T,f)″

The homeomorphisms are closed by composition.

lemma (in topology0) homeo˙composition:
assumes f∈HomeoG(T)g∈HomeoG(T)
shows Composition(⋃T)⟨f, g⟩∈HomeoG(T)
proof-
from assms have fun:f∈⋃T→⋃Tg∈⋃T→⋃T and homeo:IsAhomeomorphism(T,T,f)IsAhomeomorphism(T,T
unfolding HomeoG˙def
by auto
from fun have f O g∈⋃T→⋃T using comp˙fun by auto moreover
from homeo have bij:f∈bij(⋃T,⋃T)g∈bij(⋃T,⋃T) and cont:IsContinuous(T,T,f)IsContinuous(T,T,g)
and contconv:
IsContinuous(T,T,converse(f))IsContinuous(T,T,converse(g)) unfolding IsAhome-
omorphism˙def by auto
from bij have f O g∈bij(⋃T,⋃T) using comp˙bij by auto moreover
from cont have IsContinuous(T,T,f O g) using comp˙cont by auto moreover
have converse(f O g)=converse(g) O converse(f) using converse˙comp by auto
with contconv have IsContinuous(T,T,converse(f O g)) using comp˙cont by auto
ultimately
have f O g∈HomeoG(T) unfolding HomeoG˙def IsAhomeomorphism˙def by auto
then show thesis using func˙ZF˙5˙L2 fun by auto
qed

The identity function is a homeomorphism.

lemma (in topology0) homeo˙id:
shows id(⋃T)∈HomeoG(T)
proof-
have converse(id(⋃T)) O id(⋃T)=id(⋃T) using left˙comp˙inverse id˙bij by auto
then have converse(id(⋃T))=id(⋃T) using right˙comp˙id by auto
then show thesis unfolding HomeoG˙def IsAhomeomorphism˙def using id˙cont
id˙type id˙bij
by auto
qed

The homeomorphisms form a monoid and its neutral element is the identity.

theorem (in topology0) homeo˙submonoid:

```
 shows IsAmonoid(HomeoG(T),restrict(Composition(⋃T),HomeoG(T)×HomeoG(T)))

 TheNeutralElement(HomeoG(T),restrict(Composition(⋃T),HomeoG(T)×HomeoG(T)))=id(⋃T)
proof-
 have cl:HomeoG(T) –is closed under″ Composition(⋃T) unfolding IsOpClosed˙def
using homeo˙composition by auto
 moreover have sub:HomeoG(T)⊆⋃T→⋃T unfolding HomeoG˙def by auto more-
over
  have ne:TheNeutralElement(⋃T→⋃T, Composition(⋃T))∈HomeoG(T) using
homeo˙id Group˙ZF˙2˙5˙L2(2) by auto
 ultimately show IsAmonoid(HomeoG(T),restrict(Composition(⋃T),HomeoG(T)×HomeoG(T)))
using Group˙ZF˙2˙5˙L2(1)
   monoid0.group0˙1˙T1 unfolding monoid0˙def by force
 from cl sub ne have TheNeutralElement(HomeoG(T),restrict(Composition(⋃T),HomeoG(T)×HomeoG(T)))=
Composition(⋃T))
   using Group˙ZF˙2˙5˙L2(1) group0˙1˙L6 by blast moreover
 have id(⋃T)=TheNeutralElement(⋃T→⋃T, Composition(⋃T)) using Group˙ZF˙2˙5˙L2(2)
by auto
 ultimately show TheNeutralElement(HomeoG(T),restrict(Composition(⋃T),HomeoG(T)×HomeoG(T)))=id(
by auto
qed
```

The homeomorphisms form a group, with the composition.

```
theorem(in topology0) homeo˙group:
 shows IsAgroup(HomeoG(T),restrict(Composition(⋃T),HomeoG(T)×HomeoG(T)))
proof-
 {
   fix x assume AS:x∈HomeoG(T)
   then have surj:x∈surj(⋃T,⋃T) and bij:x∈bij(⋃T,⋃T) unfolding HomeoG˙def
IsAhomeomorphism˙def bij˙def by auto
   from bij have converse(x)∈bij(⋃T,⋃T) using bij˙converse˙bij by auto
     with bij have conx˙fun:converse(x)∈⋃T→⋃Tx∈⋃T→⋃T unfolding bij˙def
inj˙def by auto
   from surj have id:x O converse(x)=id(⋃T) using right˙comp˙inverse by auto
    from conx˙fun have Composition(⋃T)⟨x,converse(x)⟩=x O converse(x) using
func˙ZF˙5˙L2 by auto
   with id have Composition(⋃T)⟨x,converse(x)⟩=id(⋃T) by auto
  moreover have converse(x)∈HomeoG(T) unfolding HomeoG˙def using conx˙fun(1)
homeo˙inv AS unfolding HomeoG˙def
     by auto
   ultimately have ∃M∈HomeoG(T). Composition(⋃T)⟨x,M⟩=id(⋃T) by auto
 }
 then have ∀x∈HomeoG(T). ∃M∈HomeoG(T). Composition(⋃T)⟨x,M⟩=id(⋃T)
by auto
 then show thesis using homeo˙submonoid definition˙of˙group by auto
qed
```

64.2 Examples computed

As a first example, we show that the group of homeomorphisms of the cocardinal topology is the group of bijective functions.

```
theorem homeo˙cocardinal:
  assumes InfCard(Q)
  shows HomeoG(CoCardinal(X,Q))=bij(X,X)
proof
  from assms have n:Q≠0 unfolding InfCard˙def by auto
  then show HomeoG(CoCardinal(X,Q)) ⊆ bij(X, X) unfolding HomeoG˙def IsA-
homeomorphism˙def
    using union˙cocardinal by auto
  {
    fix f assume a:f∈bij(X,X)
    then have converse(f)∈bij(X,X) using bij˙converse˙bij by auto
    then have cinj:converse(f)∈inj(X,X) unfolding bij˙def by auto
    from a have fun:f∈X→X unfolding bij˙def inj˙def by auto
   then have two:two˙top˙spaces0((CoCardinal(X,Q)),(CoCardinal(X,Q)),f) unfold-
ing two˙top˙spaces0˙def
      using union˙cocardinal assms n CoCar˙is˙topology by auto
    {
      fix N assume N–is closed in″(CoCardinal(X,Q))
     then have N˙def:N=X ∨ (N∈Pow(X) ∧ N≺Q) using closed˙sets˙cocardinal n by
auto
     then have restrict(converse(f),N)∈bij(N,converse(f)N) using cinj restrict˙bij by
auto
      then have N≈f-N unfolding vimage˙def eqpoll˙def by auto
      then have f-N≈N using eqpoll˙sym by auto
      with N˙def have N=X ∨ (f-N≺Q ∧ N∈Pow(X)) using eq˙lesspoll˙trans by auto
      with fun have f-N=X ∨ (f-N≺Q ∧ (f-N)∈Pow(X)) using func1˙1˙L3 func1˙1˙L4
by auto
      then have f-N –is closed in″(CoCardinal(X,Q)) using closed˙sets˙cocardinal n
by auto
    }
   then have ∀N. N–is closed in″(CoCardinal(X,Q)) ⟶ f-N –is closed in″(CoCardinal(X,Q))
by auto
   then have IsContinuous((CoCardinal(X,Q)),(CoCardinal(X,Q)),f) using two˙top˙spaces0.Top˙ZF˙2˙1˙L4

      two˙top˙spaces0.Top˙ZF˙2˙1˙L3 two˙top˙spaces0.Top˙ZF˙2˙1˙L2 two by auto
  }
   then have ∀f∈bij(X,X). IsContinuous((CoCardinal(X,Q)),(CoCardinal(X,Q)),f)
by auto
   then have ∀f∈bij(X,X). IsContinuous((CoCardinal(X,Q)),(CoCardinal(X,Q)),f)
∧ IsContinuous((CoCardinal(X,Q)),(CoCardinal(X,Q)),converse(f))
    using bij˙converse˙bij by auto
 then have ∀f∈bij(X,X). IsAhomeomorphism((CoCardinal(X,Q)),(CoCardinal(X,Q)),f)
unfolding IsAhomeomorphism˙def
    using n union˙cocardinal by auto
  then show bij(X,X)⊆HomeoG((CoCardinal(X,Q))) unfolding HomeoG˙def bij˙def
```

```
inj˙def using n union˙cocardinal
    by auto
qed
```

The group of homeomorphism of the excluded set is a direct product of the bijections on $X \setminus T$ and the bijections on $X \cap T$.

```
theorem homeo˙excluded:
  shows HomeoG(ExcludedSet(X,T))=–f∈bij(X,X). f(X-T)=(X-T)″
proof
  have sub1:X-T⊆X by auto
  {
    fix g assume g∈HomeoG(ExcludedSet(X,T))
   then have fun:g:X→X and bij:g∈bij(X,X) and hom:IsAhomeomorphism((ExcludedSet(X,T)),(ExcludedSet(X.
unfolding HomeoG˙def
      using union˙excludedset unfolding IsAhomeomorphism˙def by auto
    {
      assume A:g(X-T)=X and B:X∩T≠0
     have rfun:restrict(g,X-T):X-T→X using fun restrict˙fun sub1 by auto moreover
      from A fun have –gaa. aa∈X-T″=X using func˙imagedef sub1 by auto
      then have ∀x∈X. x∈–gaa. aa∈X-T″ by auto
      then have ∀x∈X. ∃aa∈X-T. x=gaa by auto
      then have ∀x∈X. ∃aa∈X-T. x=restrict(g,X-T)aa by auto
      with A have surj:restrict(g,X-T)∈surj(X-T,X) using rfun unfolding surj˙def by
auto
      from B obtain d where d∈Xd∈T by auto
      with bij have gd∈X using apply˙funtype unfolding bij˙def inj˙def by auto
       then obtain s where restrict(g,X-T)s=gds∈X-T using surj unfolding surj˙def
by blast
      then have gs=gd by auto
      with ‹d∈X›‹s∈X-T› have s=d using bij unfolding bij˙def inj˙def by auto
      then have False using ‹s∈X-T› ‹d∈T› by auto
    }
    then have g(X-T)=X ⟶ X∩T=0 by auto
    then have reg:g(X-T)=X ⟶ X-T=X by auto
    then have g(X-T)=X ⟶ g(X-T)=X-T by auto
     then have g(X-T)=X ⟶ g∈–f∈bij(X,X). f(X-T)=(X-T)″ using bij by auto
moreover
    {
      fix gg
    assume A:gg(X-T)≠X and hom2:IsAhomeomorphism((ExcludedSet(X,T)),(ExcludedSet(X,T)),gg)
       from hom2 have fun:gg∈X→X and bij:gg∈bij(X,X) unfolding IsAhomeomor-
phism˙def bij˙def inj˙def using union˙excludedset by auto
      have sub:X-T⊆⋃(ExcludedSet(X,T)) using union˙excludedset by auto
    with hom2 have gg(Interior(X-T,(ExcludedSet(X,T))))=Interior(gg(X-T),(ExcludedSet(X,T)))
        using int˙top˙invariant by auto moreover
    from sub1 have Interior(X-T,(ExcludedSet(X,T)))=X-T using interior˙set˙excludedset
by auto
     ultimately have gg(X-T)=Interior(gg(X-T),(ExcludedSet(X,T))) by auto more-
over
```

have ss:gg(X-T)⊆X using fun func1˙1˙L6(2) by auto
then have Interior(gg(X-T),(ExcludedSet(X,T))) = (gg(X-T))-T using interior˙set˙excludedset A
by auto
ultimately have eq:gg(X-T)=(gg(X-T))-T by auto
{
assume (gg(X-T))∩T≠0
then obtain t where t∈T and im:t∈gg(X-T) by blast
then have t∉(gg(X-T))-T by auto
then have False using eq im by auto
}
then have (gg(X-T))∩T=0 by auto
then have gg(X-T)⊆X-T using ss by blast
}
then have ∀gg. gg(X-T)≠X ∧ IsAhomeomorphism(ExcludedSet(X,T),ExcludedSet(X,T),gg)⟶ gg(X-T)⊆X-T by auto moreover
from bij have conbij:converse(g)∈bij(X,X) using bij˙converse˙bij by auto
then have confun:converse(g)∈X→X unfolding bij˙def inj˙def by auto
{
assume A:converse(g)(X-T)=X and B:X∩T≠0
have rfun:restrict(converse(g),X-T):X-T→X using confun restrict˙fun sub1 by auto moreover
from A confun have –converse(g)aa. aa∈X-T″=X using func˙imagedef sub1 by auto
then have ∀x∈X. x∈–converse(g)aa. aa∈X-T″ by auto
then have ∀x∈X. ∃aa∈X-T. x=converse(g)aa by auto
then have ∀x∈X. ∃aa∈X-T. x=restrict(converse(g),X-T)aa by auto
with A have surj:restrict(converse(g),X-T)∈surj(X-T,X) using rfun unfolding surj˙def by auto
from B obtain d where d∈Xd∈T by auto
with conbij have converse(g)d∈X using apply˙funtype unfolding bij˙def inj˙def by auto
then obtain s where restrict(converse(g),X-T)s=converse(g)ds∈X-T using surj unfolding surj˙def by blast
then have converse(g)s=converse(g)d by auto
with ‹d∈X›‹s∈X-T› have s=d using conbij unfolding bij˙def inj˙def by auto
then have False using ‹s∈X-T› ‹d∈T› by auto
}
then have converse(g)(X-T)=X ⟶ X∩T=0 by auto
then have converse(g)(X-T)=X ⟶ X-T=X by auto
then have converse(g)(X-T)=X ⟶ g-(X-T)=(X-T) unfolding vimage˙def by auto
then have G:converse(g)(X-T)=X ⟶ g(g-(X-T))=g(X-T) by auto
have GG:g(g-(X-T))=(X-T) using sub1 surj˙image˙vimage bij unfolding bij˙def by auto
with G have converse(g)(X-T)=X ⟶ g(X-T)=X-T by auto
then have converse(g)(X-T)=X ⟶ g∈–f∈bij(X,X). f(X-T)=(X-T)″ using bij by auto moreover
from hom have IsAhomeomorphism(ExcludedSet(X,T), ExcludedSet(X,T), con-

```
verse(g)) using homeo˙inv by auto
  moreover note hom ultimately have g∈–f∈bij(X,X). f(X-T)=(X-T)″ ∨ (g(X-T)⊆X-T
∧ converse(g)(X-T)⊆X-T)
     by force
    then have g∈–f∈bij(X,X). f(X-T)=(X-T)″ ∨ (g(X-T)⊆X-T ∧ g-(X-T)⊆X-T)
unfolding vimage˙def by auto moreover
   have g-(X-T)⊆X-T ⟶ g(g-(X-T))⊆g(X-T) using func1˙1˙L8 by auto
   with GG have g-(X-T)⊆X-T ⟶ (X-T)⊆g(X-T) by force
  ultimately have g∈–f∈bij(X,X). f(X-T)=(X-T)″ ∨ (g(X-T)⊆X-T ∧ (X-T)⊆g(X-T))
by auto
   then have g∈–f∈bij(X,X). f(X-T)=(X-T)″ using bij by auto
  }
  then show HomeoG(ExcludedSet(X,T))⊆–f∈bij(X,X). f(X-T)=(X-T)″ by auto
  {
   fix g assume as:g∈bij(X,X)g(X-T)=X-T
   then have inj:g∈inj(X,X) and im:g-(g(X-T))=g-(X-T) unfolding bij˙def by auto
   from inj have g-(g(X-T))=X-T using inj˙vimage˙image sub1 by force
   with im have as˙3:g-(X-T)=X-T by auto
   {
    fix A
    assume A∈(ExcludedSet(X,T))
    then have A=X∨A∩T=0 A⊆X unfolding ExcludedSet˙def by auto
    then have A⊆X-T∨A=X by auto moreover
    {
     assume A=X
      with as(1) have gA=X using surj˙range˙image˙domain unfolding bij˙def by
auto
    }
    moreover
    {
     assume A⊆X-T
     then have gA⊆g(X-T) using func1˙1˙L8 by auto
     then have gA⊆(X-T) using as(2)  by auto
    }
    ultimately have gA⊆(X-T) ∨ gA=X by auto
    then have gA∈(ExcludedSet(X,T)) unfolding ExcludedSet˙def by auto
   }
   then have ∀A∈(ExcludedSet(X,T)). gA∈(ExcludedSet(X,T)) by auto moreover
   {
    fix A assume A∈(ExcludedSet(X,T))
    then have A=X∨A∩T=0 A⊆X unfolding ExcludedSet˙def by auto
    then have A⊆X-T∨A=X by auto moreover
    {
     assume A=X
     with as(1) have g-A=X using func1˙1˙L4 unfolding bij˙def inj˙def by auto
    }
    moreover
    {
     assume A⊆X-T
```

```
      then have g-A⊆g-(X-T) using func1˙1˙L8 by auto
      then have g-A⊆(X-T) using as˙3 by auto
    }
    ultimately have g-A⊆(X-T) ∨ g-A=X by auto
    then have g-A∈(ExcludedSet(X,T)) unfolding ExcludedSet˙def by auto
  }
   then have IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),g) unfolding Is-
Continuous˙def by auto moreover
  note as(1) ultimately have IsAhomeomorphism(ExcludedSet(X,T),ExcludedSet(X,T),g)

    using union˙excludedset bij˙cont˙open˙homeo by auto
  with as(1) have g∈HomeoG(ExcludedSet(X,T)) unfolding bij˙def inj˙def HomeoG˙def
using union˙excludedset by auto
  }
  then show –f ∈ bij(X, X) . f  (X - T) = X - T″ ⊆ HomeoG(ExcludedSet(X,T))
by auto
qed
```

We now give some lemmas that will help us compute HomeoG(IncludedSet(X,T)).

```
lemma cont˙in˙cont˙ex:
  assumes IsContinuous(IncludedSet(X,T),IncludedSet(X,T),f) f:X→X T⊆X
  shows IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),f)
proof-
  from assms(2,3) have two:two˙top˙spaces0(IncludedSet(X,T),IncludedSet(X,T),f)
using union˙includedset includedset˙is˙topology
    unfolding two˙top˙spaces0˙def by auto
  {
    fix A assume A∈(ExcludedSet(X,T))
    then have A∩T=0 ∨ A=XA⊆X unfolding ExcludedSet˙def by auto
    then have A–is closed in″(IncludedSet(X,T)) using closed˙sets˙includedset assms
by auto
   then have f-A–is closed in″(IncludedSet(X,T)) using two˙top˙spaces0.TopZF˙2˙1˙L1
assms(1)
      two assms includedset˙is˙topology by auto
    then have (f-A)∩T=0 ∨ f-A=Xf-A⊆X using closed˙sets˙includedset assms(1,3)
by auto
    then have f-A∈(ExcludedSet(X,T)) unfolding ExcludedSet˙def by auto
  }
 then show IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),f) unfolding IsCon-
tinuous˙def by auto
qed

lemma cont˙ex˙cont˙in:
  assumes IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),f) f:X→X T⊆X
  shows IsContinuous(IncludedSet(X,T),IncludedSet(X,T),f)
proof-
  from assms(2) have two:two˙top˙spaces0(ExcludedSet(X,T),ExcludedSet(X,T),f)
using union˙excludedset excludedset˙is˙topology
    unfolding two˙top˙spaces0˙def by auto
```

```
{
  fix A assume A∈(IncludedSet(X,T))
  then have T⊆A ∨ A=0A⊆X unfolding IncludedSet˙def by auto
 then have A–is closed in″(ExcludedSet(X,T)) using closed˙sets˙excludedset assms
by auto
 then have f-A–is closed in″(ExcludedSet(X,T)) using two˙top˙spaces0.TopZF˙2˙1˙L1
assms(1)
    two assms excludedset˙is˙topology by auto
   then have T⊆(f-A) ∨ f-A=0f-A⊆X using closed˙sets˙excludedset assms(1,3) by
auto
  then have f-A∈(IncludedSet(X,T)) unfolding IncludedSet˙def by auto
}
then show IsContinuous(IncludedSet(X,T),IncludedSet(X,T),f) unfolding IsCon-
tinuous˙def by auto
qed
```

The previous lemmas imply that the group of homeomorphisms of the included set topology is the same as the one of the excluded set topology.

```
lemma homeo˙included:
 assumes T⊆X
 shows HomeoG(IncludedSet(X,T))=–f ∈ bij(X, X) . f  (X - T) = X - T″
proof-
 {
  fix f assume f∈HomeoG(IncludedSet(X,T))
   then have hom:IsAhomeomorphism(IncludedSet(X,T),IncludedSet(X,T),f) and
fun:f∈X→X and
  bij:f∈bij(X,X) unfolding HomeoG˙def IsAhomeomorphism˙def using union˙includedset
assms by auto
   then have cont:IsContinuous(IncludedSet(X,T),IncludedSet(X,T),f) unfolding
IsAhomeomorphism˙def by auto
 then have IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),f) using cont˙in˙cont˙ex
fun assms by auto moreover
  {
  from hom have cont1:IsContinuous(IncludedSet(X,T),IncludedSet(X,T),converse(f))
unfolding IsAhomeomorphism˙def by auto moreover
    have converse(f):X→X using bij˙converse˙bij bij unfolding bij˙def inj˙def by
auto moreover
   note assms ultimately
     have IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),converse(f)) using
cont˙in˙cont˙ex assms by auto
  }
   then have IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),converse(f)) by
auto
  moreover note bij ultimately
 have IsAhomeomorphism(ExcludedSet(X,T),ExcludedSet(X,T),f) unfolding IsA-
homeomorphism˙def
   using union˙excludedset by auto
   with fun have f∈HomeoG(ExcludedSet(X,T)) unfolding HomeoG˙def using
union˙excludedset by auto
```

```
  }
 then have HomeoG(IncludedSet(X,T))⊆HomeoG(ExcludedSet(X,T)) by auto more-
over
  {
    fix f assume f∈HomeoG(ExcludedSet(X,T))
   then have hom:IsAhomeomorphism(ExcludedSet(X,T),ExcludedSet(X,T),f) and
fun:f∈X→X and
    bij:f∈bij(X,X) unfolding HomeoG˙def IsAhomeomorphism˙def using union˙excludedset
assms by auto
    then have cont:IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),f) unfolding
IsAhomeomorphism˙def by auto
   then have IsContinuous(IncludedSet(X,T),IncludedSet(X,T),f) using cont˙ex˙cont˙in
fun assms by auto moreover
    {
    from hom have cont1:IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),converse(f))
unfolding IsAhomeomorphism˙def by auto moreover
       have converse(f):X→X using bij˙converse˙bij bij unfolding bij˙def inj˙def by
auto moreover
      note assms ultimately
    have IsContinuous(IncludedSet(X,T),IncludedSet(X,T),converse(f)) using cont˙ex˙cont˙in
assms by auto
    }
   then have IsContinuous(IncludedSet(X,T),IncludedSet(X,T),converse(f)) by auto
    moreover note bij ultimately
    have IsAhomeomorphism(IncludedSet(X,T),IncludedSet(X,T),f) unfolding IsA-
homeomorphism˙def
      using union˙includedset assms by auto
   with fun have f∈HomeoG(IncludedSet(X,T)) unfolding HomeoG˙def using union˙includedset
assms by auto
  }
  then have HomeoG(ExcludedSet(X,T))⊆HomeoG(IncludedSet(X,T)) by auto ul-
timately
  show thesis using homeo˙excluded by auto
qed
```

Finally, let's compute part of the group of homeomorphisms of an order topology.

```
lemma homeo˙order:
  assumes IsLinOrder(X,r)∃x y. x≠y∧x∈X∧y∈X
  shows ord˙iso(X,r,X,r)⊆HomeoG(OrdTopology X r)
proof
  fix f assume f∈ord˙iso(X,r,X,r)
  then have bij:f∈bij(X,X) and ord:∀x∈X. ∀y∈X. ⟨x, y⟩ ∈ r ⟷ ⟨f  x, f  y⟩ ∈ r
    unfolding ord˙iso˙def by auto
   have twoSpac:two˙top˙spaces0(OrdTopology X r,OrdTopology X r,f) unfolding
two˙top˙spaces0˙def
    using bij unfolding bij˙def inj˙def using union˙ordtopology[OF assms] Ordtopol-
ogy˙is˙a˙topology(1)[OF assms(1)]
    by auto
```

```
{
  fix c d assume A:c∈Xd∈X
  {
    fix x assume AA:x∈Xx≠cx≠d⟨c,x⟩∈r⟨x,d⟩∈r
    then have ⟨fc,fx⟩∈r⟨fx,fd⟩∈r using A(2,1) ord by auto moreover
    {
      assume fx=fc ∨ fx=fd
     then have x=c∨x=d using bij unfolding bij˙def inj˙def using A(2,1) AA(1) by
auto
      then have False using AA(2,3) by auto
    }
    then have fx≠fcfx≠fd by auto moreover
    have fx∈X using bij unfolding bij˙def inj˙def using apply˙type AA(1) by auto
    ultimately have fx∈IntervalX(X,r,fc,fd) unfolding IntervalX˙def Interval˙def by
auto
  }
 then have –fx. x∈IntervalX(X,r,c,d)″⊆IntervalX(X,r,fc,fd) unfolding IntervalX˙def
Interval˙def by auto
  moreover
  {
    fix y assume y∈IntervalX(X,r,fc,fd)
    then have y:y∈Xy≠fcy≠fd⟨fc,y⟩∈r⟨y,fd⟩∈r unfolding IntervalX˙def Interval˙def
by auto
    then obtain s where s:s∈Xy=fs using bij unfolding bij˙def surj˙def by auto
    {
      assume s=c∨s=d
      then have fs=fc∨fs=fd by auto
      then have False using s(2) y(2,3) by auto
    }
    then have s≠cs≠d by auto moreover
    have ⟨c,s⟩∈r⟨s,d⟩∈r using y(4,5) s ord A(2,1) by auto moreover
     note s(1) ultimately have s∈IntervalX(X,r,c,d) unfolding IntervalX˙def Inter-
val˙def by auto
    then have y∈–fx. x∈IntervalX(X,r,c,d)″ using s(2) by auto
  }
   ultimately have –fx. x∈IntervalX(X,r,c,d)″=IntervalX(X,r,fc,fd) by auto more-
over
  have IntervalX(X,r,c,d)⊆X unfolding IntervalX˙def by auto moreover
  have f:X→X using bij unfolding bij˙def surj˙def by auto ultimately
  have fIntervalX(X,r,c,d)=IntervalX(X,r,fc,fd) using func˙imagedef by auto
}
 then have inter:∀c∈X. ∀d∈X. fIntervalX(X,r,c,d)=IntervalX(X,r,fc,fd) ∧ fc∈X ∧
fd∈X using bij
  unfolding bij˙def inj˙def by auto
{
  fix c assume A:c∈X
  {
    fix x assume AA:x∈Xx≠c⟨c,x⟩∈r
    then have ⟨fc,fx⟩∈r using A ord by auto moreover
```

```
      {
        assume fx=fc
        then have x=c using bij unfolding bij˙def inj˙def using A AA(1) by auto
        then have False using AA(2) by auto
      }
      then have fx≠fc by auto moreover
      have fx∈X using bij unfolding bij˙def inj˙def using apply˙type AA(1) by auto
      ultimately have fx∈RightRayX(X,r,fc) unfolding RightRayX˙def by auto
    }
  then have –fx. x∈RightRayX(X,r,c)″⊆RightRayX(X,r,fc) unfolding RightRayX˙def
by auto
    moreover
    {
      fix y assume y∈RightRayX(X,r,fc)
      then have y:y∈Xy≠fc⟨fc,y⟩∈r unfolding RightRayX˙def by auto
      then obtain s where s:s∈Xy=fs using bij unfolding bij˙def surj˙def by auto
      {
        assume s=c
        then have fs=fc by auto
        then have False using s(2) y(2) by auto
      }
      then have s≠c by auto moreover
      have ⟨c,s⟩∈r using y(3) s ord A by auto moreover
       note s(1) ultimately have s∈RightRayX(X,r,c) unfolding RightRayX˙def by
auto
      then have y∈–fx. x∈RightRayX(X,r,c)″ using s(2) by auto
    }
  ultimately have –fx. x∈RightRayX(X,r,c)″=RightRayX(X,r,fc) by auto moreover
    have RightRayX(X,r,c)⊆X unfolding RightRayX˙def by auto moreover
    have f:X→X using bij unfolding bij˙def surj˙def by auto ultimately
    have fRightRayX(X,r,c)=RightRayX(X,r,fc) using func˙imagedef by auto
  }
  then have rray:∀c∈X. fRightRayX(X,r,c)=RightRayX(X,r,fc) ∧ fc∈X using bij
    unfolding bij˙def inj˙def by auto
  {
    fix c assume A:c∈X
    {
      fix x assume AA:x∈Xx≠c⟨x,c⟩∈r
      then have ⟨fx,fc⟩∈r using A ord by auto moreover
      {
        assume fx=fc
        then have x=c using bij unfolding bij˙def inj˙def using A AA(1) by auto
        then have False using AA(2) by auto
      }
      then have fx≠fc by auto moreover
      have fx∈X using bij unfolding bij˙def inj˙def using apply˙type AA(1) by auto
      ultimately have fx∈LeftRayX(X,r,fc) unfolding LeftRayX˙def by auto
    }
    then have –fx. x∈LeftRayX(X,r,c)″⊆LeftRayX(X,r,fc) unfolding LeftRayX˙def
```

by auto
moreover
{
fix y assume y∈LeftRayX(X,r,fc)
then have y:y∈Xy≠fc⟨y,fc⟩∈r unfolding LeftRayX˙def by auto
then obtain s where s:s∈Xy=fs using bij unfolding bij˙def surj˙def by auto
{
assume s=c
then have fs=fc by auto
then have False using s(2) y(2) by auto
}
then have s≠c by auto moreover
have ⟨s,c⟩∈r using y(3) s ord A by auto moreover
note s(1) ultimately have s∈LeftRayX(X,r,c) unfolding LeftRayX˙def by auto
then have y∈–fx. x∈LeftRayX(X,r,c)″ using s(2) by auto
}
ultimately have –fx. x∈LeftRayX(X,r,c)″=LeftRayX(X,r,fc) by auto moreover
have LeftRayX(X,r,c)⊆X unfolding LeftRayX˙def by auto moreover
have f:X→X using bij unfolding bij˙def surj˙def by auto ultimately
have fLeftRayX(X,r,c)=LeftRayX(X,r,fc) using func˙imagedef by auto
}
then have lray:∀ c∈X. fLeftRayX(X,r,c)=LeftRayX(X,r,fc)∧fc∈X using bij
unfolding bij˙def inj˙def by auto
have r1:∀ U∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪ –LeftRayX(X, r, b) . b ∈ X″ ∪
–RightRayX(X, r, b) . b ∈ X″. fU∈(–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪ –LeftRayX(X, r, b) . b ∈ X″ ∪
–RightRayX(X, r, b) . b ∈ X″) apply safe prefer 3 using rray apply blast prefer 2 using lray apply blast
using inter apply auto
proof-
fix xa y assume xa∈Xy∈X
then have fxa∈Xfy∈X using bij unfolding bij˙def inj˙def by auto
then show ∃ x∈X. ∃ ya∈X. IntervalX(X, r, f xa, f y) = IntervalX(X, r, x, ya) by auto
qed
have r2:–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪ –LeftRayX(X, r, b) . b ∈ X″ ∪ –RightRayX(X, r, b) . b ∈ X″⊆(OrdTopology X r)
using base˙sets˙open[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]] by blast
{
fix U assume U∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪ –LeftRayX(X, r, b) . b ∈ X″ ∪ –RightRayX(X, r, b) . b ∈ X″
with r1 have fU∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪ –LeftRayX(X, r, b) . b ∈ X″ ∪ –RightRayX(X, r, b) . b ∈ X″
by auto
with r2 have fU∈(OrdTopology X r) by blast
}
then have ∀ U∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪ –LeftRayX(X, r, b) . b ∈ X″ ∪

–RightRayX(X, r, b) . b ∈ X″. fU∈(OrdTopology X r) by blast
then have f˙open:∀ U∈(OrdTopology X r). fU∈(OrdTopology X r) using two˙top˙spaces0.base˙image˙open[OF twoSpac Ordtopology˙is˙a˙topology(2)[OF assms(1)]]
by auto
{
fix c d assume A:c∈Xd∈X
then obtain cc dd where pre:fcc=cfdd=dcc∈Xdd∈X using bij unfolding bij˙def surj˙def by blast
with inter have f IntervalX(X, r, cc, dd) = IntervalX(X, r, c, d) by auto
then have f-(fIntervalX(X, r, cc, dd)) = f-(IntervalX(X, r, c, d)) by auto
moreover
have IntervalX(X, r, cc, dd)⊆X unfolding IntervalX˙def by auto moreover
have f∈inj(X,X) using bij unfolding bij˙def by auto ultimately
have IntervalX(X, r, cc, dd)=f-IntervalX(X, r, c, d) using inj˙vimage˙image by auto
moreover
from pre(3,4) have IntervalX(X, r, cc, dd)∈–IntervalX(X,r,e1,e2). ⟨e1,e2⟩∈X×X″ by auto
ultimately have f-IntervalX(X, r, c, d)∈(OrdTopology X r) using
base˙sets˙open[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]] by auto
}
then have inter:∀ c∈X. ∀ d∈X. f-IntervalX(X, r, c, d)∈(OrdTopology X r) by auto
{
fix c assume A:c∈X
then obtain cc where pre:fcc=ccc∈X using bij unfolding bij˙def surj˙def by blast
with rray have f RightRayX(X, r, cc) = RightRayX(X, r, c) by auto
then have f-(fRightRayX(X, r, cc)) = f-(RightRayX(X, r, c)) by auto
moreover
have RightRayX(X, r, cc)⊆X unfolding RightRayX˙def by auto moreover
have f∈inj(X,X) using bij unfolding bij˙def by auto ultimately
have RightRayX(X, r, cc)=f-RightRayX(X, r, c) using inj˙vimage˙image by auto
moreover
from pre(2) have RightRayX(X, r, cc)∈–RightRayX(X,r,e2). e2∈X″ by auto
ultimately have f-RightRayX(X, r, c)∈(OrdTopology X r) using
base˙sets˙open[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]] by auto
}
then have rray:∀ c∈X. f-RightRayX(X, r, c)∈(OrdTopology X r) by auto
{
fix c assume A:c∈X
then obtain cc where pre:fcc=ccc∈X using bij unfolding bij˙def surj˙def by blast
with lray have f LeftRayX(X, r, cc) = LeftRayX(X, r, c) by auto
then have f-(fLeftRayX(X, r, cc)) = f-(LeftRayX(X, r, c)) by auto
moreover
have LeftRayX(X, r, cc)⊆X unfolding LeftRayX˙def by auto moreover
have f∈inj(X,X) using bij unfolding bij˙def by auto ultimately
have LeftRayX(X, r, cc)=f-LeftRayX(X, r, c) using inj˙vimage˙image by auto
moreover
from pre(2) have LeftRayX(X, r, cc)∈–LeftRayX(X,r,e2). e2∈X″ by auto
ultimately have f-LeftRayX(X, r, c)∈(OrdTopology X r) using

base˙sets˙open[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]] by auto
}
then have lray:∀ c∈X. f-LeftRayX(X, r, c)∈(OrdTopology X r) by auto
{
fix U assume U∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪ –LeftRayX(X, r, b) . b ∈ X″ ∪ –RightRayX(X, r, b) . b ∈ X″
with lray inter rray have f-U∈(OrdTopology X r) by auto
}
then have ∀ U∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪ –LeftRayX(X, r, b) . b ∈ X″ ∪ –RightRayX(X, r, b) . b ∈ X″.
f-U∈(OrdTopology X r) by blast
then have fcont:IsContinuous(OrdTopology X r,OrdTopology X r,f) using two˙top˙spaces0.Top˙ZF˙2˙1˙L5[OF twoSpac
Ordtopology˙is˙a˙topology(2)[OF assms(1)]] by auto
from fcont f˙open bij have IsAhomeomorphism(OrdTopology X r,OrdTopology X r,f) using bij˙cont˙open˙homeo
union˙ordtopology[OF assms] by auto
then show f∈HomeoG(OrdTopology X r) unfolding HomeoG˙def using bij union˙ordtopology[OF assms]
unfolding bij˙def inj˙def by auto
qed

This last example shows that order isomorphic sets give homeomorphic topological spaces.

64.3 Properties preserved by functions

The continuous image of a connected space is connected.

theorem (in two˙top˙spaces0) cont˙image˙conn:
assumes IsContinuous(τ_1,τ_2,f) f∈surj(X_1,X_2) τ_1–is connected″
shows τ_2–is connected″
proof-
{
fix U
assume Uop:U∈τ_2 and Ucl:U–is closed in″τ_2
from Uop assms(1) have f-U∈τ_1 unfolding IsContinuous˙def by auto moreover
from Ucl assms(1) have f-U–is closed in″τ_1 using TopZF˙2˙1˙L1 by auto ultimately
have disj:f-U=0 ∨ f-U=$\bigcup\tau_1$ using assms(3) unfolding IsConnected˙def by auto moreover
{
assume as:f-U≠0
then have U≠0 using func1˙1˙L13 by auto
from as disj have f-U=$\bigcup\tau_1$ by auto
then have f(f-U)=f($\bigcup\tau_1$) by auto moreover
have U⊆$\bigcup\tau_2$ using Uop by blast ultimately
have U=f($\bigcup\tau_1$) using surj˙image˙vimage assms(2) Uop by force
then have $\bigcup\tau_2$=U using surj˙range˙image˙domain assms(2) by auto

```
    }
    moreover
    {
      assume as:U≠0
      from Uop have s:U⊆⋃τ2 by auto
      with as obtain u where uU:u∈U by auto
      with s have u∈⋃τ2 by auto
       with assms(2) obtain w where fw=uw∈⋃τ1 unfolding surj˙def X1˙def X2˙def
by blast
      with uU have w∈f-U using func1˙1˙L15 assms(2) unfolding surj˙def by auto
      then have f-U≠0 by auto
    }
    ultimately have U=0∨U=⋃τ2 by auto
  }
  then show thesis unfolding IsConnected˙def by auto
qed
```

Every continuous function from a space which has some property P and a space which has the property anti(P), given that this property is preserved by continuous functions, if follows that the range of the function is in the spectrum. Applied to connectedness, it follows that continuous functions from a connected space to a totally-disconnected one are constant.

```
corollary(in two˙top˙spaces0) cont˙conn˙tot˙disc:
  assumes IsContinuous(τ1,τ2,f) τ1–is connected″ τ2–is totally-disconnected″ f:X1→X2
X1≠0
  shows ∃q∈X2. ∀w∈X1. f(w)=q
proof-
  from assms(4) have surj:f∈surj(X1,range(f)) using fun˙is˙surj by auto
  have sub:range(f)⊆X2 using func1˙1˙L5B assms(4) by auto
   from assms(1) have cont:IsContinuous(τ1,τ2–restricted to″range(f),f) using re-
str˙image˙cont range˙image˙domain
    assms(4) by auto
  have union:⋃(τ2–restricted to″range(f))=range(f) unfolding RestrictedTo˙def us-
ing sub by auto
  then have two˙top˙spaces0(τ1,τ2–restricted to″range(f),f) unfolding two˙top˙spaces0˙def
      using surj unfolding surj˙def using tau1˙is˙top topology0.Top˙1˙L4 unfolding
topology0˙def using tau2˙is˙top
    by auto
  then have conn:(τ2–restricted to″range(f))–is connected″ using two˙top˙spaces0.cont˙image˙conn
surj assms(2) cont
    union by auto
  then have range(f)–is in the spectrum of″IsConnected using assms(3) sub unfolding
IsTotDis˙def antiProperty˙def
    using union by auto
  then have range(f)≲1 using conn˙spectrum by auto moreover
  from assms(5) have fX1≠0 using func1˙1˙L15A assms(4) by auto
  then have range(f)≠0 using range˙image˙domain assms(4) by auto
  ultimately obtain q where uniq:range(f)=–q″ using lepoll˙1˙is˙sing by blast
  {
```

fix w assume w∈X_1
then have fw∈range(f) using func1˙1˙L5A(2) assms(4) by auto
with uniq have fw=q by auto
}
then have ∀w∈X_1. fw=q by auto
then show thesis using uniq sub by auto
qed

The continuous image of a compact space is compact.

theorem (in two˙top˙spaces0) cont˙image˙com:
assumes IsContinuous(τ_1,τ_2,f) f∈surj(X_1,X_2) X_1–is compact of cardinal″K–in″τ_1
shows X_2–is compact of cardinal″K–in″τ_2
proof-
have X_2⊆⋃τ_2 by auto moreover
{
fix U assume as:X_2⊆⋃U U⊆τ_2
then have P:–f-V. V∈U″⊆τ_1 using assms(1) unfolding IsContinuous˙def by auto
from as(1) have f-X_2 ⊆ f-(⋃U) by blast
then have f-X_2 ⊆ converse(f)(⋃U) unfolding vimage˙def by auto moreover
have converse(f)(⋃U)=(⋃V∈U. converse(f)V) using image˙UN by force ultimately
have f-X_2 ⊆ (⋃V∈U. converse(f)V) by auto
then have f-X_2 ⊆ (⋃V∈U. f-V) unfolding vimage˙def by auto
then have X_1 ⊆ (⋃V∈U. f-V) using func1˙1˙L4 assms(2) unfolding surj˙def by force
then have X_1 ⊆ ⋃–f-V. V∈U″ by auto
with P assms(3) have ∃N∈Pow(–f-V. V∈U″). X_1 ⊆ ⋃N ∧ N≺K unfolding IsCompactOfCard˙def by auto
then obtain N where N∈Pow(–f-V. V∈U″) X_1 ⊆ ⋃N N≺K by auto
then have fin:N≺K and sub:N⊆–f-V. V∈U″ and cov:X_1 ⊆ ⋃N unfolding FinPow˙def by auto
from sub have –fR. R∈N″⊆–f(f-V). V∈U″ by auto moreover
have ∀V∈U. V⊆⋃τ_2 using as(2) by auto ultimately
have –fR. R∈N″⊆U using surj˙image˙vimage assms(2) by auto moreover
let FN=–⟨R,fR⟩. R∈N″
have FN:FN:N→–fR. R∈N″ unfolding Pi˙def function˙def domain˙def by auto
{
fix S assume S∈–fR. R∈N″
then obtain R where R˙def:R∈NfR=S by auto
then have ⟨R,fR⟩∈FN by auto
then have FNR=fR using FN apply˙equality by auto
then have ∃R∈N. FNR=S using R˙def by auto
}
then have surj:FN∈surj(N,–fR. R∈N″) unfolding surj˙def using FN by force
from fin have N:N≲K Ord(K) using assms(3) lesspoll˙imp˙lepoll unfolding IsCompactOfCard˙def
using Card˙is˙Ord by auto
then have –fR. R∈N″≲N using surj˙fun˙inv˙2 surj by auto
then have –fR. R∈N″≺K using fin lesspoll˙trans1 by blast

moreover
have $\bigcup$–fR. R∈N″=f($\bigcup$N) using image˙UN by auto
then have fX$_1$ ⊆ $\bigcup$–fR. R∈N″ using cov by blast
then have X$_2$ ⊆ $\bigcup$–fR. R∈N″ using assms(2) surj˙range˙image˙domain by auto
ultimately have ∃NN∈Pow(U). X$_2$ ⊆ $\bigcup$NN ∧ NN≺K by auto
}
then have ∀U∈Pow(τ_2). X$_2$ ⊆ $\bigcup$U ⟶ (∃NN∈Pow(U). X$_2$ ⊆ $\bigcup$NN ∧ NN≺K) by auto
ultimately show thesis using assms(3) unfolding IsCompactOfCard˙def by auto
qed

As it happends to connected spaces, a continuous function from a compact space to an anti-compact space has finite range.

corollary (in two˙top˙spaces0) cont˙comp˙anti˙comp:
assumes IsContinuous(τ_1,τ_2,f) X$_1$–is compact in″τ_1 τ_2–is anti-compact″ f:X$_1$→X$_2$ X$_1$≠0
shows Finite(range(f)) and range(f)≠0
proof-
from assms(4) have surj:f∈surj(X$_1$,range(f)) using fun˙is˙surj by auto
have sub:range(f)⊆X$_2$ using func1˙1˙L5B assms(4) by auto
from assms(1) have cont:IsContinuous(τ_1,τ_2–restricted to″range(f),f) using restr˙image˙cont range˙image˙domain
assms(4) by auto
have union:$\bigcup$(τ_2–restricted to″range(f))=range(f) unfolding RestrictedTo˙def using sub by auto
then have two˙top˙spaces0(τ_1,τ_2–restricted to″range(f),f) unfolding two˙top˙spaces0˙def
using surj unfolding surj˙def using tau1˙is˙top topology0.Top˙1˙L4 unfolding topology0˙def using tau2˙is˙top
by auto
then have range(f)–is compact in″(τ_2–restricted to″range(f)) using surj two˙top˙spaces0.cont˙image˙com cont union
assms(2) Compact˙is˙card˙nat by force
then have range(f)–is in the spectrum of″(λT. ($\bigcup$T) –is compact in″T) using assms(3) sub unfolding IsAntiComp˙def antiProperty˙def
using union by auto
then show Finite(range(f)) using compact˙spectrum by auto moreover
from assms(5) have fX$_1$≠0 using func1˙1˙L15A assms(4) by auto
then show range(f)≠0 using range˙image˙domain assms(4) by auto
qed

As a consequence, it follows that quotient topological spaces of compact (connected) spaces are compact (connected).

corollary(in topology0) compQuot:
assumes ($\bigcup$T)–is compact in″T equiv($\bigcup$T,r)
shows ($\bigcup$T)//r–is compact in″(–quotient by″r)
proof-
have surj:–⟨b,r–b″⟩. b∈$\bigcup$T″∈surj($\bigcup$T,($\bigcup$T)//r) using quotient˙proj˙surj by auto
moreover have tot:$\bigcup$(–quotient by″r)=($\bigcup$T)//r using total˙quo˙equi assms(2) by auto

ultimately have cont:IsContinuous(T,–quotient by″r,–⟨b,r–b″⟩. b∈⋃T″) using quotient˙func˙cont
EquivQuo˙def assms(2) by auto
from surj tot have two˙top˙spaces0(T,–quotient by″r,–⟨b,r–b″⟩. b∈⋃T″) unfolding two˙top˙spaces0˙def
using topSpaceAssum equiv˙quo˙is˙top assms(2) unfolding surj˙def by auto
with surj cont tot assms(1) show thesis using two˙top˙spaces0.cont˙image˙com Compact˙is˙card˙nat by force
qed

corollary(in topology0) ConnQuot:
assumes T–is connected″ equiv(⋃T,r)
shows (–quotient by″r)–is connected″
proof-
have surj:–⟨b,r–b″⟩. b∈⋃T″∈surj(⋃T,(⋃T)//r) using quotient˙proj˙surj by auto
moreover have tot:⋃(–quotient by″r)=(⋃T)//r using total˙quo˙equi assms(2) by auto
ultimately have cont:IsContinuous(T,–quotient by″r,–⟨b,r–b″⟩. b∈⋃T″) using quotient˙func˙cont
EquivQuo˙def assms(2) by auto
from surj tot have two˙top˙spaces0(T,–quotient by″r,–⟨b,r–b″⟩. b∈⋃T″) unfolding two˙top˙spaces0˙def
using topSpaceAssum equiv˙quo˙is˙top assms(2) unfolding surj˙def by auto
with surj cont tot assms(1) show thesis using two˙top˙spaces0.cont˙image˙conn by force
qed

end

65 Topology 10

theory Topology˙ZF˙10
imports Topology˙ZF˙7
begin

This file deals with properties of product spaces. We only consider product of two spaces, and most of this proofs, can be used to prove the results in product of a finite number of spaces.

65.1 Closure and closed sets in product space

The closure of a product, is the product of the closures.

lemma cl˙product:
assumes T–is a topology″ S–is a topology″ A⊆⋃T B⊆⋃S
shows Closure(A×B,ProductTopology(T,S))=Closure(A,T)×Closure(B,S)
proof
have A×B⊆⋃T×⋃S using assms(3,4) by auto

then have sub:A×B⊆⋃ProductTopology(T,S) using Top˙1˙4˙T1(3) assms(1,2) by auto
have top:ProductTopology(T,S)–is a topology″ using Top˙1˙4˙T1(1) assms(1,2) by auto
{
fix x assume asx:x∈Closure(A×B,ProductTopology(T,S))
then have reg:∀U∈ProductTopology(T,S). x∈U ⟶ U∩(A×B)≠0 using topology0.cl˙inter˙neigh
sub top unfolding topology0˙def by blast
from asx have x∈⋃ProductTopology(T,S) using topology0.Top˙3˙L11(1) top unfolding topology0˙def
using sub by blast
then have xSigma:x∈⋃T×⋃S using Top˙1˙4˙T1(3) assms(1,2) by auto
then have ⟨fst(x),snd(x)⟩∈⋃T×⋃S using Pair˙fst˙snd˙eq by auto
then have xT:fst(x)∈⋃T and xS:snd(x)∈⋃S by auto
{
fix U V assume as:U∈T fst(x)∈U
have ⋃S∈S using assms(2) unfolding IsATopology˙def by auto
with as have U×(⋃S)∈ProductCollection(T,S) unfolding ProductCollection˙def by auto
then have P:U×(⋃S)∈ProductTopology(T,S) using Top˙1˙4˙T1(2) assms(1,2) base˙sets˙open by blast
with xS as(2) have ⟨fst(x),snd(x)⟩∈U×(⋃S) by auto
then have x∈U×(⋃S) using Pair˙fst˙snd˙eq xSigma by auto
with P reg have U×(⋃S)∩A×B≠0 by auto
then have noEm:U∩A≠0 by auto
}
then have ∀U∈T. fst(x)∈U ⟶ U∩A≠0 by auto moreover
{
fix U V assume as:U∈S snd(x)∈U
have ⋃T∈T using assms(1) unfolding IsATopology˙def by auto
with as have (⋃T)×U∈ProductCollection(T,S) unfolding ProductCollection˙def by auto
then have P:(⋃T)×U∈ProductTopology(T,S) using Top˙1˙4˙T1(2) assms(1,2) base˙sets˙open by blast
with xT as(2) have ⟨fst(x),snd(x)⟩∈(⋃T)×U by auto
then have x∈(⋃T)×U using Pair˙fst˙snd˙eq xSigma by auto
with P reg have (⋃T)×U∩A×B≠0 by auto
then have noEm:U∩B≠0 by auto
}
then have ∀U∈S. snd(x)∈U ⟶ U∩B≠0 by auto
ultimately have fst(x)∈Closure(A,T) snd(x)∈Closure(B,S) using
topology0.inter˙neigh˙cl assms(3,4) unfolding topology0˙def
using assms(1,2) xT xS by auto
then have ⟨fst(x),snd(x)⟩∈Closure(A,T)×Closure(B,S) by auto
with xSigma have x∈Closure(A,T)×Closure(B,S) by auto
}
then show Closure(A×B,ProductTopology(T,S))⊆Closure(A,T)×Closure(B,S) by auto

```
  {
    fix x assume x:x∈Closure(A,T)×Closure(B,S)
    then have xcl:fst(x)∈Closure(A,T) snd(x)∈Closure(B,S) by auto
  from xcl(1) have regT:∀U∈T. fst(x)∈U ⟶ U∩A≠0 using topology0.cl˙inter˙neigh
      unfolding topology0˙def using assms(1,3) by blast
  from xcl(2) have regS:∀U∈S. snd(x)∈U ⟶ U∩B≠0 using topology0.cl˙inter˙neigh
      unfolding topology0˙def using assms(2,4) by blast
     from x assms(3,4) have x∈⋃T×⋃S using topology0.Top˙3˙L11(1) unfolding
topology0˙def
      using assms(1,2) by blast
    then have xtot:x∈⋃ProductTopology(T,S) using Top˙1˙4˙T1(3) assms(1,2) by
auto
    {
      fix PO assume as:PO∈ProductTopology(T,S) x∈PO
      then obtain POB where base:POB∈ProductCollection(T,S) x∈POBPOB⊆PO
using point˙open˙base˙neigh
        Top˙1˙4˙T1(2) assms(1,2) base˙sets˙open by blast
     then obtain VT VS where V:VT∈T VS∈S x∈VT×VS POB=VT×VS unfolding
ProductCollection˙def
        by auto
      from V(3) have x:fst(x)∈VT snd(x)∈VS by auto
      from V(1) regT x(1) have VT∩A≠0 by auto moreover
      from V(2) regS x(2) have VS∩B≠0 by auto ultimately
      have VT×VS∩A×B≠0 by auto
      with V(4) base(3) have PO∩A×B≠0 by blast
    }
    then have ∀P∈ProductTopology(T,S). x∈P ⟶ P∩A×B≠0 by auto
  then have x∈Closure(A×B,ProductTopology(T,S)) using topology0.inter˙neigh˙cl
      unfolding topology0˙def using top sub xtot by auto
  }
 then show Closure(A,T)×Closure(B,S)⊆Closure(A×B,ProductTopology(T,S)) by
auto
qed
```

The product of closed sets, is closed in the product topology.

```
corollary closed˙product:
  assumes T–is a topology″ S–is a topology″ A–is closed in″TB–is closed in″S
  shows (A×B) –is closed in″ProductTopology(T,S)
proof-
  from assms(3,4) have sub:A⊆⋃TB⊆⋃S unfolding IsClosed˙def by auto
  then have A×B⊆⋃T×⋃S by auto
  then have sub1:A×B⊆⋃ProductTopology(T,S) using Top˙1˙4˙T1(3) assms(1,2)
by auto
  from sub assms have Closure(A,T)=AClosure(B,S)=B using topology0.Top˙3˙L8
    unfolding topology0˙def by auto
  then have Closure(A×B,ProductTopology(T,S))=A×B using cl˙product
    assms(1,2) sub by auto
  then show thesis using topology0.Top˙3˙L8 unfolding topology0˙def
    using sub1 Top˙1˙4˙T1(1) assms(1,2) by auto
```

qed

65.2 Separation properties in product space

The product of T_0 spaces is T_0.

theorem T0˙product:
assumes T–is a topology″S–is a topology″T–is T_0″S–is T_0″
shows ProductTopology(T,S)–is T_0″
proof-
{
fix x y assume x∈⋃ProductTopology(T,S)y∈⋃ProductTopology(T,S)x≠y
then have tot:x∈⋃T×⋃Sy∈⋃T×⋃Sx≠y using Top˙1˙4˙T1(3) assms(1,2) by auto
then have ⟨fst(x),snd(x)⟩∈⋃T×⋃S⟨fst(y),snd(y)⟩∈⋃T×⋃S and disj:fst(x)≠fst(y)∨snd(x)≠snd(y)

using Pair˙fst˙snd˙eq by auto
then have T:fst(x)∈⋃Tfst(y)∈⋃T and S:snd(y)∈⋃Ssnd(x)∈⋃S and p:fst(x)≠fst(y)∨snd(x)≠snd(y)
by auto
{
assume fst(x)≠fst(y)
with T assms(3) have (∃U∈T. (fst(x)∈U∧fst(y)∉U)∨(fst(y)∈U∧fst(x)∉U)) unfolding
isT0˙def by auto
then obtain U where U∈T (fst(x)∈U∧fst(y)∉U)∨(fst(y)∈U∧fst(x)∉U) by auto
with S have (⟨fst(x),snd(x)⟩∈U×(⋃S) ∧ ⟨fst(y),snd(y)⟩∉U×(⋃S))∨(⟨fst(y),snd(y)⟩∈U×(⋃S) ∧ ⟨fst(x),snd(x)⟩∉U×(⋃S))
by auto
then have (x∈U×(⋃S) ∧ y∉U×(⋃S))∨(y∈U×(⋃S) ∧ x∉U×(⋃S)) using Pair˙fst˙snd˙eq tot(1,2) by auto
moreover have (⋃S)∈S using assms(2) unfolding IsATopology˙def by auto
with ‹U∈T› have U×(⋃S)∈ProductTopology(T,S) using prod˙open˙open˙prod assms(1,2) by auto
ultimately
have ∃V∈ProductTopology(T,S). (x∈V ∧ y∉V)∨(y∈V ∧ x∉V) proof qed
} moreover
{
assume snd(x)≠snd(y)
with S assms(4) have (∃U∈S. (snd(x)∈U∧snd(y)∉U)∨(snd(y)∈U∧snd(x)∉U)) unfolding
isT0˙def by auto
then obtain U where U∈S (snd(x)∈U∧snd(y)∉U)∨(snd(y)∈U∧snd(x)∉U) by auto
with T have (⟨fst(x),snd(x)⟩∈(⋃T)×U ∧ ⟨fst(y),snd(y)⟩∉(⋃T)×U)∨(⟨fst(y),snd(y)⟩∈(⋃T)×U ∧ ⟨fst(x),snd(x)⟩∉(⋃T)×U)
by auto
then have (x∈(⋃T)×U ∧ y∉(⋃T)×U)∨(y∈(⋃T)×U ∧ x∉(⋃T)×U) using Pair˙fst˙snd˙eq tot(1,2) by auto
moreover have (⋃T)∈T using assms(1) unfolding IsATopology˙def by auto
with ‹U∈S› have (⋃T)×U∈ProductTopology(T,S) using prod˙open˙open˙prod

```
assms(1,2) by auto
      ultimately
      have ∃V∈ProductTopology(T,S). (x∈V ∧ y∉V)∨(y∈V ∧ x∉V) proof qed
    }moreover
    note disj
     ultimately have ∃V∈ProductTopology(T,S). (x∈V ∧ y∉V)∨(y∈V ∧ x∉V) by
auto
  }
  then show thesis unfolding isT0˙def by auto
qed
```

The product of T_1 spaces is T_1.

```
theorem T1˙product:
  assumes T–is a topology″S–is a topology″T–is T1″S–is T1″
  shows ProductTopology(T,S)–is T1″
proof-
  {
    fix x y assume x∈⋃ProductTopology(T,S)y∈⋃ProductTopology(T,S)x≠y
     then have tot:x∈⋃T×⋃Sy∈⋃T×⋃Sx≠y using Top˙1˙4˙T1(3) assms(1,2) by
auto
   then have ⟨fst(x),snd(x)⟩∈⋃T×⋃S⟨fst(y),snd(y)⟩∈⋃T×⋃S and disj:fst(x)≠fst(y)∨snd(x)≠snd(y)

      using Pair˙fst˙snd˙eq by auto
   then have T:fst(x)∈⋃Tfst(y)∈⋃T and S:snd(y)∈⋃Ssnd(x)∈⋃S and p:fst(x)≠fst(y)∨snd(x)≠snd(y)
      by auto
    {
      assume fst(x)≠fst(y)
      with T assms(3) have (∃U∈T. (fst(x)∈U∧fst(y)∉U)) unfolding
        isT1˙def by auto
      then obtain U where U∈T (fst(x)∈U∧fst(y)∉U) by auto
      with S have (⟨fst(x),snd(x)⟩∈U×(⋃S) ∧ ⟨fst(y),snd(y)⟩∉U×(⋃S)) by auto
      then have (x∈U×(⋃S) ∧ y∉U×(⋃S)) using Pair˙fst˙snd˙eq tot(1,2) by auto
      moreover have (⋃S)∈S using assms(2) unfolding IsATopology˙def by auto
      with ‹U∈T› have U×(⋃S)∈ProductTopology(T,S) using prod˙open˙open˙prod
assms(1,2) by auto
      ultimately
      have ∃V∈ProductTopology(T,S). (x∈V ∧ y∉V) proof qed
    } moreover
    {
      assume snd(x)≠snd(y)
      with S assms(4) have (∃U∈S. (snd(x)∈U∧snd(y)∉U)) unfolding
        isT1˙def by auto
      then obtain U where U∈S (snd(x)∈U∧snd(y)∉U) by auto
      with T have (⟨fst(x),snd(x)⟩∈(⋃T)×U ∧ ⟨fst(y),snd(y)⟩∉(⋃T)×U) by auto
      then have (x∈(⋃T)×U ∧ y∉(⋃T)×U) using Pair˙fst˙snd˙eq tot(1,2) by auto
      moreover have (⋃T)∈T using assms(1) unfolding IsATopology˙def by auto
      with ‹U∈S› have (⋃T)×U∈ProductTopology(T,S) using prod˙open˙open˙prod
assms(1,2) by auto
      ultimately
```

```
    have ∃V∈ProductTopology(T,S). (x∈V ∧ y∉V) proof qed
   }moreover
   note disj
   ultimately have ∃V∈ProductTopology(T,S). (x∈V ∧ y∉V) by auto
  }
  then show thesis unfolding isT1˙def by auto
qed
```

The product of T_2 spaces is T_2.

```
theorem T2˙product:
  assumes T–is a topology″S–is a topology″T–is T₂″S–is T₂″
  shows ProductTopology(T,S)–is T₂″
proof-
  {
   fix x y assume x∈⋃ProductTopology(T,S)y∈⋃ProductTopology(T,S)x≠y
    then have tot:x∈⋃T×⋃Sy∈⋃T×⋃Sx≠y using Top˙1˙4˙T1(3) assms(1,2) by
auto
   then have ⟨fst(x),snd(x)⟩∈⋃T×⋃S⟨fst(y),snd(y)⟩∈⋃T×⋃S and disj:fst(x)≠fst(y)∨snd(x)≠snd(y)

     using Pair˙fst˙snd˙eq by auto
   then have T:fst(x)∈⋃Tfst(y)∈⋃T and S:snd(y)∈⋃Ssnd(x)∈⋃S and p:fst(x)≠fst(y)∨snd(x)≠snd(y)
     by auto
    {
     assume fst(x)≠fst(y)
      with T assms(3) have (∃U∈T. ∃V∈T. (fst(x)∈U∧fst(y)∈V) ∧ U∩V=0) un-
folding
       isT2˙def by auto
     then obtain U V where U∈T V∈T fst(x)∈U fst(y)∈V U∩V=0 by auto
    with S have ⟨fst(x),snd(x)⟩∈U×(⋃S) ⟨fst(y),snd(y)⟩∈V×(⋃S) and disjoint:(U×⋃S)∩(V×⋃S)=0
by auto
     then have x∈U×(⋃S)y∈V×(⋃S) using Pair˙fst˙snd˙eq tot(1,2) by auto
     moreover have (⋃S)∈S using assms(2) unfolding IsATopology˙def by auto
    with ‹U∈T›‹V∈T› have P:U×(⋃S)∈ProductTopology(T,S) V×(⋃S)∈ProductTopology(T,S)

       using prod˙open˙open˙prod assms(1,2) by auto
     note disjoint ultimately
     have x∈U×(⋃S) ∧ y∈V×(⋃S) ∧ (U×(⋃S))∩(V×(⋃S))=0 by auto
    with P(2) have ∃UU∈ProductTopology(T,S). (x∈U×(⋃S) ∧ y∈UU ∧ (U×(⋃S))∩UU=0)
     using exI[where x=V×(⋃S) and P=λt. t∈ProductTopology(T,S) ∧ (x∈U×(⋃S)
∧ y∈t ∧ (U×(⋃S))∩t=0)] by auto
      with P(1) have ∃VV∈ProductTopology(T,S). ∃UU∈ProductTopology(T,S).
(x∈VV ∧ y∈UU ∧ VV∩UU=0)
     using exI[where x=U×(⋃S) and P=λt. t∈ProductTopology(T,S) ∧ (∃UU∈ProductTopology(T,S).
(x∈t ∧ y∈UU ∧ (t)∩UU=0))] by auto
    } moreover
    {
     assume snd(x)≠snd(y)
       with S assms(4) have (∃U∈S. ∃V∈S. (snd(x)∈U∧snd(y)∈V) ∧ U∩V=0)
unfolding
```

isT2˙def by auto
then obtain U V where U∈S V∈S snd(x)∈U snd(y)∈V U∩V=0 by auto
with T have ⟨fst(x),snd(x)⟩∈(⋃T)×U ⟨fst(y),snd(y)⟩∈(⋃T)×V and disjoint:((⋃T)×U)∩((⋃T)×V)=0 by auto
then have x∈(⋃T)×Uy∈(⋃T)×V using Pair˙fst˙snd˙eq tot(1,2) by auto
moreover have (⋃T)∈T using assms(1) unfolding IsATopology˙def by auto
with ‹U∈S›‹V∈S› have P:(⋃T)×U∈ProductTopology(T,S) (⋃T)×V∈ProductTopology(T,S)

using prod˙open˙open˙prod assms(1,2) by auto
note disjoint ultimately
have x∈(⋃T)×U ∧ y∈(⋃T)×V ∧ ((⋃T)×U)∩((⋃T)×V)=0 by auto
with P(2) have ∃UU∈ProductTopology(T,S). (x∈(⋃T)×U ∧ y∈UU ∧ ((⋃T)×U)∩UU=0)
using exI[where x=(⋃T)×V and P=λt. t∈ProductTopology(T,S) ∧ (x∈(⋃T)×U ∧ y∈t ∧ ((⋃T)×U)∩t=0)] by auto
with P(1) have ∃VV∈ProductTopology(T,S). ∃UU∈ProductTopology(T,S). (x∈VV ∧ y∈UU ∧ VV∩UU=0)
using exI[where x=(⋃T)×U and P=λt. t∈ProductTopology(T,S) ∧ (∃UU∈ProductTopology(T,S). (x∈t ∧ y∈UU ∧ (t)∩UU=0))] by auto
} moreover
note disj
ultimately have ∃VV∈ProductTopology(T, S). ∃UU∈ProductTopology(T, S). x ∈ VV ∧ y ∈ UU ∧ VV ∩ UU = 0 by auto
}
then show thesis unfolding isT2˙def by auto
qed

The product of regular spaces is regular.

theorem regular˙product:
assumes T–is a topology″ S–is a topology″ T–is regular″ S–is regular″
shows ProductTopology(T,S)–is regular″
proof-
{
fix x U assume x∈⋃ProductTopology(T,S) U∈ProductTopology(T,S) x∈U
then obtain V W where VW:V∈TW∈S V×W⊆U and x:x∈V×W using prod˙top˙point˙neighb

assms(1,2) by blast
then have p:fst(x)∈Vsnd(x)∈W by auto
from p(1) ‹V∈T› obtain VV where VV:fst(x)∈VV Closure(VV,T)⊆V VV∈T using
assms(1,3) topology0.regular˙imp˙exist˙clos˙neig unfolding topology0˙def
by force moreover
from p(2) ‹W∈S› obtain WW where WW:snd(x)∈WW Closure(WW,S)⊆W WW∈S using
assms(2,4) topology0.regular˙imp˙exist˙clos˙neig unfolding topology0˙def
by force ultimately
have x∈VV×WW using x by auto
moreover from ‹Closure(VV,T)⊆V› ‹Closure(WW,S)⊆W› have Closure(VV,T)×Closure(WW,S) ⊆ V×W
by auto

moreover from VV(3) WW(3) have VV⊆⋃TWW⊆⋃S by auto
ultimately have x∈VV×WW Closure(VV×WW,ProductTopology(T,S)) ⊆ V×W using cl˙product assms(1,2)
by auto
moreover have VV×WW∈ProductTopology(T,S) using prod˙open˙open˙prod assms(1,2)
VV(3) WW(3) by auto
ultimately have ∃Z∈ProductTopology(T,S). x∈Z ∧ Closure(Z,ProductTopology(T,S))⊆V×W by auto
with VW(3) have ∃Z∈ProductTopology(T,S). x∈Z ∧ Closure(Z,ProductTopology(T,S))⊆U by auto
}
then have ∀x∈⋃ProductTopology(T,S). ∀U∈ProductTopology(T,S).x∈U ⟶ (∃Z∈ProductTopology(T,S). x∈Z ∧ Closure(Z,ProductTopology(T,S))⊆U)
by auto
then show thesis using topology0.exist˙clos˙neig˙imp˙regular unfolding topology0˙def
using assms(1,2) Top˙1˙4˙T1(1) by auto
qed

65.3 Connection properties in product space

First, we prove that the projection functions are open.

lemma projection˙open:
assumes T–is a topology″S–is a topology″B∈ProductTopology(T,S)
shows –y∈⋃T. ∃x∈⋃S. ⟨y,x⟩∈B″∈T
proof-
{
fix z assume z∈–y∈⋃T. ∃x∈⋃S. ⟨y,x⟩∈B″
then obtain x where x:x∈⋃S and z:z∈⋃T and p:⟨z,x⟩∈B by auto
then have z∈–y∈⋃T. ⟨y,x⟩∈B″ –y∈⋃T. ⟨y,x⟩∈B″⊆–y∈⋃T. ∃x∈⋃S. ⟨y,x⟩∈B″ by auto moreover
from x have –y∈⋃T. ⟨y,x⟩∈B″∈T using prod˙sec˙open2 assms by auto
ultimately have ∃V∈T. z∈V ∧ V⊆–y∈⋃T. ∃x∈⋃S. ⟨y,x⟩∈B″ unfolding Bex˙def by auto
}
then show –y∈⋃T. ∃x∈⋃S. ⟨y,x⟩∈B″∈T using topology0.open˙neigh˙open unfolding topology0˙def
using assms(1) by blast
qed

lemma projection˙open2:
assumes T–is a topology″S–is a topology″B∈ProductTopology(T,S)
shows –y∈⋃S. ∃x∈⋃T. ⟨x,y⟩∈B″∈S
proof-
{
fix z assume z∈–y∈⋃S. ∃x∈⋃T. ⟨x,y⟩∈B″
then obtain x where x:x∈⋃T and z:z∈⋃S and p:⟨x,z⟩∈B by auto
then have z∈–y∈⋃S. ⟨x,y⟩∈B″ –y∈⋃S. ⟨x,y⟩∈B″⊆–y∈⋃S. ∃x∈⋃T. ⟨x,y⟩∈B″ by auto moreover

from x have –y∈⋃S. ⟨x,y⟩∈B″∈S using prod˙sec˙open1 assms by auto
ultimately have ∃V∈S. z∈V ∧ V⊆–y∈⋃S. ∃x∈⋃T. ⟨x,y⟩∈B″ unfolding Bex˙def by auto
}
then show –y∈⋃S. ∃x∈⋃T. ⟨x,y⟩∈B″∈S using topology0.open˙neigh˙open unfolding topology0˙def
using assms(2) by blast
qed

The product of connected spaces is connected.

theorem compact˙product:
assumes T–is a topology″S–is a topology″T–is connected″S–is connected″
shows ProductTopology(T,S)–is connected″
proof-
{
fix U assume U:U∈ProductTopology(T,S) U–is closed in″ProductTopology(T,S)
then have P:U∈ProductTopology(T,S) ⋃ProductTopology(T,S)-U∈ProductTopology(T,S)
unfolding IsClosed˙def by auto
{
fix s assume s:s∈⋃S
with P(1) have p:–x∈⋃T. ⟨x,s⟩∈U″∈T using prod˙sec˙open2 assms(1,2) by auto
from s P(2) have oop:–y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)″∈T using prod˙sec˙open2
assms(1,2) by blast
then have ⋃T-(⋃T-–y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)″)=–y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)″ by auto
with oop have cl:(⋃T-–y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)″) –is closed in″T unfolding IsClosed˙def by auto
{
fix t assume t∈⋃T-–y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)″
then have tt:t∈⋃T t∉–y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)″ by auto
then have ⟨t,s⟩∉(⋃ProductTopology(T,S)-U) by auto
then have ⟨t,s⟩∈U ∨ ⟨t,s⟩∉⋃ProductTopology(T,S) by auto
then have ⟨t,s⟩∈U ∨ ⟨t,s⟩∉⋃T×⋃S using Top˙1˙4˙T1(3) assms(1,2) by auto
with tt(1) s have ⟨t,s⟩∈U by auto
with tt(1) have t∈–x∈⋃T. ⟨x,s⟩∈U″ by auto
} moreover
{
fix t assume t∈–x∈⋃T. ⟨x,s⟩∈U″
then have tt:t∈⋃T ⟨t,s⟩∈U by auto
then have ⟨t,s⟩∉⋃ProductTopology(T,S)-U by auto
then have t∉–y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)″ by auto
with tt(1) have t∈⋃T-–y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)″ by auto
}
ultimately have –x∈⋃T. ⟨x,s⟩∈U″=⋃T-–y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)″ by blast
with cl have –x∈⋃T. ⟨x,s⟩∈U″–is closed in″T by auto
with p assms(3) have –x∈⋃T. ⟨x,s⟩∈U″=0 ∨ –x∈⋃T. ⟨x,s⟩∈U″=⋃T unfolding

IsConnected˙def
by auto moreover
{
assume –x∈⋃T. ⟨x,s⟩∈U″=0
then have ∀x∈⋃T. ⟨x,s⟩∉U by auto
}
moreover
{
assume AA:–x∈⋃T. ⟨x,s⟩∈U″=⋃T
{
fix x assume x∈⋃T
with AA have x∈–x∈⋃T. ⟨x,s⟩∈U″ by auto
then have ⟨x,s⟩∈U by auto
}
then have ∀x∈⋃T. ⟨x,s⟩∈U by auto
}
ultimately have (∀x∈⋃T. ⟨x,s⟩∉U) ∨ (∀x∈⋃T. ⟨x,s⟩∈U) by blast
}
then have reg:∀s∈⋃S. (∀x∈⋃T. ⟨x,s⟩∉U) ∨ (∀x∈⋃T. ⟨x,s⟩∈U) by auto
{
fix q assume qU:q∈⋃T×–snd(qq). qq∈U″
then obtain t u where t:t∈⋃T u∈U q=⟨t,snd(u)⟩ by auto
with U(1) have u∈⋃ProductTopology(T,S) by auto
then have u∈⋃T×⋃S using Top˙1˙4˙T1(3) assms(1,2) by auto moreover
then have uu:u=⟨fst(u),snd(u)⟩ using Pair˙fst˙snd˙eq by auto ultimately
have fu:fst(u)∈⋃Tsnd(u)∈⋃S by (safe,auto)
with reg have (∀tt∈⋃T. ⟨tt,snd(u)⟩∉U)∨(∀tt∈⋃T. ⟨tt,snd(u)⟩∈U) by auto
with ‹u∈U› uu fu(1) have ∀tt∈⋃T. ⟨tt,snd(u)⟩∈U by force
with t(1,3) have q∈U by auto
}
then have U1:⋃T×–snd(qq). qq∈U″⊆U by auto
{
fix t assume t:t∈⋃T
with P(1) have p:–x∈⋃S. ⟨t,x⟩∈U″∈S using prod˙sec˙open1 assms(1,2) by auto
from t P(2) have oop:–x∈⋃S. ⟨t,x⟩∈(⋃ProductTopology(T,S)-U)″∈S using prod˙sec˙open1
assms(1,2) by blast
then have ⋃S-(⋃S-–x∈⋃S. ⟨t,x⟩∈(⋃ProductTopology(T,S)-U)″)=–y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)″ by auto
with oop have cl:(⋃S-–y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)″) –is closed in″S unfolding IsClosed˙def by auto
{
fix s assume s∈⋃S-–y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)″
then have tt:s∈⋃S s∉–y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)″ by auto
then have ⟨t,s⟩∉(⋃ProductTopology(T,S)-U) by auto
then have ⟨t,s⟩∈U ∨ ⟨t,s⟩∉⋃ProductTopology(T,S) by auto
then have ⟨t,s⟩∈U ∨ ⟨t,s⟩∉⋃T×⋃S using Top˙1˙4˙T1(3) assms(1,2) by auto
with tt(1) t have ⟨t,s⟩∈U by auto
with tt(1) have s∈–x∈⋃S. ⟨t,x⟩∈U″ by auto

} moreover
{
fix s assume s∈–x∈⋃S. ⟨t,x⟩∈U″
then have tt:s∈⋃S ⟨t,s⟩∈U by auto
then have ⟨t,s⟩∉⋃ProductTopology(T,S)-U by auto
then have s∉–y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)″ by auto
with tt(1) have s∈⋃S-–y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)″ by auto
}
ultimately have –x∈⋃S. ⟨t,x⟩∈U″=⋃S-–y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)″ by blast
with cl have –x∈⋃S. ⟨t,x⟩∈U″–is closed in″S by auto
with p assms(4) have –x∈⋃S. ⟨t,x⟩∈U″=0 ∨ –x∈⋃S. ⟨t,x⟩∈U″=⋃S unfolding IsConnected˙def
by auto moreover
{
assume –x∈⋃S. ⟨t,x⟩∈U″=0
then have ∀x∈⋃S. ⟨t,x⟩∉U by auto
}
moreover
{
assume AA:–x∈⋃S. ⟨t,x⟩∈U″=⋃S
{
fix x assume x∈⋃S
with AA have x∈–x∈⋃S. ⟨t,x⟩∈U″ by auto
then have ⟨t,x⟩∈U by auto
}
then have ∀x∈⋃S. ⟨t,x⟩∈U by auto
}
ultimately have (∀x∈⋃S. ⟨t,x⟩∉U) ∨ (∀x∈⋃S. ⟨t,x⟩∈U) by blast
}
then have reg:∀s∈⋃T. (∀x∈⋃S. ⟨s,x⟩∉U) ∨ (∀x∈⋃S. ⟨s,x⟩∈U) by auto
{
fix q assume qU:q∈–fst(qq). qq∈U″×⋃S
then obtain qq s where t:q=⟨fst(qq),s⟩ qq∈U s∈⋃S by auto
with U(1) have qq∈⋃ProductTopology(T,S) by auto
then have qq∈⋃T×⋃S using Top˙1˙4˙T1(3) assms(1,2) by auto moreover
then have qq:qq=⟨fst(qq),snd(qq)⟩ using Pair˙fst˙snd˙eq by auto ultimately
have fq:fst(qq)∈⋃Tsnd(qq)∈⋃S by (safe,auto)
from fq(1) reg have (∀tt∈⋃S. ⟨fst(qq),tt⟩∉U)∨(∀tt∈⋃S. ⟨fst(qq),tt⟩∈U) by auto moreover
with ‹qq∈U› qq fq(2) have ∀tt∈⋃S. ⟨fst(qq),tt⟩∈U by force
with t(1,3) have q∈U by auto
}
then have U2:–fst(qq). qq∈U″×⋃S⊆U by blast
{
assume U≠0
then obtain u where u:u∈U by auto
{
fix aa assume aa∈⋃T×⋃S

then obtain t s where t∈⋃Ts∈⋃Saa=⟨t,s⟩ by auto
with u have ⟨t,snd(u)⟩∈⋃T×–snd(qq). qq∈U″ by auto
with U1 have ⟨t,snd(u)⟩∈U by auto
moreover have t=fst(⟨t,snd(u)⟩) by auto moreover note ‹s∈⋃S› ultimately
have ⟨t,s⟩∈–fst(qq). qq∈U″×⋃S by blast
with U2 have ⟨t,s⟩∈U by auto
with ‹aa=⟨t,s⟩› have aa∈U by auto
}
then have ⋃T×⋃S⊆U by auto moreover
with U(1) have U⊆⋃ProductTopology(T,S) by auto ultimately
have ⋃T×⋃S=U using Top˙1˙4˙T1(3) assms(1,2) by auto
}
then have (U=0)∨(U=⋃T×⋃S) by auto
}
then show thesis unfolding IsConnected˙def using Top˙1˙4˙T1(3) assms(1,2) by auto
qed

end

66 Topology 11

theory Topology˙ZF˙11 imports Topology˙ZF˙7 Finite˙ZF˙1

begin

This file deals with order topologies. The order topology is already defined in Topology˙ZF˙examples˙1.thy.

66.1 Order topologies

We will assume most of the time that the ordered set has more than one point. It is natural to think that the topological properties can be translated to properties of the order; since every order rises one and only one topology in a set.

66.2 Separation properties

Order topologies have a lot of separation properties.

Every order topology is Hausdorff.

theorem order˙top˙T2:
assumes IsLinOrder(X,r) ∃x y. x≠y∧x∈X∧y∈X
shows (OrdTopology X r)–is T_2″
proof-
{
fix x y assume A1:x∈⋃(OrdTopology X r)y∈⋃(OrdTopology X r)x≠y

then have AS:x∈Xy∈Xx≠y using union˙ordtopology[OF assms(1) assms(2)] by auto
{
assume A2:∃z∈X-–x,y″. (⟨x,y⟩∈r⟶⟨x,z⟩∈r∧⟨z,y⟩∈r)∧(⟨y,x⟩∈r⟶⟨y,z⟩∈r∧⟨z,x⟩∈r)
from AS(1,2) assms(1) have ⟨x,y⟩∈r∨⟨y,x⟩∈r unfolding IsLinOrder˙def IsTotal˙def by auto moreover
{
assume ⟨x,y⟩∈r
with AS A2 obtain z where z:⟨x,z⟩∈r⟨z,y⟩∈rz∈Xz≠xz≠y by auto
with AS(1,2) have x∈LeftRayX(X,r,z)y∈RightRayX(X,r,z) unfolding LeftRayX˙def RightRayX˙def
by auto moreover
have LeftRayX(X,r,z)∩RightRayX(X,r,z)=0 using inter˙lray˙rray[OF z(3) z(3) assms(1)]
unfolding IntervalX˙def using Order˙ZF˙2˙L4[OF total˙is˙refl ˙ z(3)] assms(1) unfolding IsLinOrder˙def
by auto moreover
have LeftRayX(X,r,z)∈(OrdTopology X r)RightRayX(X,r,z)∈(OrdTopology X r)
using z(3) base˙sets˙open[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]] by auto
ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U ∧ y∈V ∧ U∩V=0 by auto
}
moreover
{
assume ⟨y,x⟩∈r
with AS A2 obtain z where z:⟨y,z⟩∈r⟨z,x⟩∈rz∈Xz≠xz≠y by auto
with AS(1,2) have y∈LeftRayX(X,r,z)x∈RightRayX(X,r,z) unfolding LeftRayX˙def RightRayX˙def
by auto moreover
have LeftRayX(X,r,z)∩RightRayX(X,r,z)=0 using inter˙lray˙rray[OF z(3) z(3) assms(1)]
unfolding IntervalX˙def using Order˙ZF˙2˙L4[OF total˙is˙refl ˙ z(3)] assms(1) unfolding IsLinOrder˙def
by auto moreover
have LeftRayX(X,r,z)∈(OrdTopology X r)RightRayX(X,r,z)∈(OrdTopology X r)
using z(3) base˙sets˙open[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]] by auto
ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U ∧ y∈V ∧ U∩V=0 by auto
}
ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U ∧ y∈V ∧ U∩V=0 by auto
}
moreover
{
assume A2:∀z∈X - –x, y″. (⟨x, y⟩ ∈ r ∧ (⟨x, z⟩ ∉ r ∨ ⟨z, y⟩ ∉ r)) ∨ (⟨y, x⟩ ∈ r

∧ (⟨y, z⟩ ∉ r ∨ ⟨z, x⟩ ∉ r))
from AS(1,2) assms(1) have disj:⟨x,y⟩∈r∨⟨y,x⟩∈r unfolding IsLinOrder˙def IsTotal˙def by auto moreover
{
assume TT:⟨x,y⟩∈r
with AS assms(1) have T:⟨y,x⟩∉r unfolding IsLinOrder˙def antisym˙def by auto
from TT AS(1-3) have x∈LeftRayX(X,r,y)y∈RightRayX(X,r,x) unfolding LeftRayX˙def RightRayX˙def
by auto moreover
{
fix z assume z∈LeftRayX(X,r,y)∩RightRayX(X,r,x)
then have ⟨z,y⟩∈r⟨x,z⟩∈rz∈X-–x,y″ unfolding RightRayX˙def LeftRayX˙def by auto
with A2 T have False by auto
}
then have LeftRayX(X,r,y)∩RightRayX(X,r,x)=0 by auto moreover
have LeftRayX(X,r,y)∈(OrdTopology X r)RightRayX(X,r,x)∈(OrdTopology X r)
using base˙sets˙open[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]] AS by auto
ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U ∧ y∈V ∧ U∩V=0 by auto
}
moreover
{
assume TT:⟨y,x⟩∈r
with AS assms(1) have T:⟨x,y⟩∉r unfolding IsLinOrder˙def antisym˙def by auto
from TT AS(1-3) have y∈LeftRayX(X,r,x)x∈RightRayX(X,r,y) unfolding LeftRayX˙def RightRayX˙def
by auto moreover
{
fix z assume z∈LeftRayX(X,r,x)∩RightRayX(X,r,y)
then have ⟨z,x⟩∈r⟨y,z⟩∈rz∈X-–x,y″ unfolding RightRayX˙def LeftRayX˙def by auto
with A2 T have False by auto
}
then have LeftRayX(X,r,x)∩RightRayX(X,r,y)=0 by auto moreover
have LeftRayX(X,r,x)∈(OrdTopology X r)RightRayX(X,r,y)∈(OrdTopology X r)
using base˙sets˙open[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]] AS by auto
ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U ∧ y∈V ∧ U∩V=0 by auto
}
ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U ∧ y∈V ∧ U∩V=0 by auto
}

ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U ∧ y∈V ∧ U∩V=0 by auto
}
then show thesis unfolding isT2˙def by auto
qed

Every order topology is T_4, but the proof needs lots of machinery. At the end of the file, we will prove that every order topology is normal; sooner or later.

66.3 Connectedness properties

Connectedness is related to two properties of orders: completeness and density

Some order-dense properties:

definition
IsDenseSub (˙ –is dense in˝˙–with respect to˝˙) where
A –is dense in˝X–with respect to˝r ≡
∀x∈X. ∀y∈X. ⟨x,y⟩∈r ∧ x≠y ⟶ (∃z∈A-–x,y˝. ⟨x,z⟩∈r∧⟨z,y⟩∈r)

definition
IsDenseUnp (˙ –is not-properly dense in˝˙–with respect to˝˙) where
A –is not-properly dense in˝X–with respect to˝r ≡
∀x∈X. ∀y∈X. ⟨x,y⟩∈r ∧ x≠y ⟶ (∃z∈A. ⟨x,z⟩∈r∧⟨z,y⟩∈r)

definition
IsWeaklyDenseSub (˙ –is weakly dense in˝˙–with respect to˝˙) where
A –is weakly dense in˝X–with respect to˝r ≡
∀x∈X. ∀y∈X. ⟨x,y⟩∈r ∧ x≠y ⟶ ((∃z∈A-–x,y˝. ⟨x,z⟩∈r∧⟨z,y⟩∈r)∨ IntervalX(X,r,x,y)=0)

definition
IsDense (˙ –is dense with respect to˝˙) where
X –is dense with respect to˝r ≡
∀x∈X. ∀y∈X. ⟨x,y⟩∈r ∧ x≠y ⟶ (∃z∈X-–x,y˝. ⟨x,z⟩∈r∧⟨z,y⟩∈r)

lemma dense˙sub:
shows (X –is dense with respect to˝r) ⟷ (X –is dense in˝X–with respect to˝r)
unfolding IsDenseSub˙def IsDense˙def by auto

lemma not˙prop˙dense˙sub:
shows (A –is dense in˝X–with respect to˝r) ⟶ (A –is not-properly dense in˝X–with respect to˝r)
unfolding IsDenseSub˙def IsDenseUnp˙def by auto

In densely ordered sets, intervals are infinite.

theorem dense˙order˙inf˙intervals:
assumes IsLinOrder(X,r) IntervalX(X, r, b, c)≠0b∈Xc∈X X–is dense with respect to˝r

```
  shows ¬Finite(IntervalX(X, r, b, c))
proof
  assume fin:Finite(IntervalX(X, r, b, c))
  have sub:IntervalX(X, r, b, c)⊆X unfolding IntervalX˙def by auto
  have p:Minimum(r,IntervalX(X, r, b, c))∈IntervalX(X, r, b, c) using Finite˙ZF˙1˙T2(2)[OF
assms(1) Finite˙Fin[OF fin sub] assms(2)]
    by auto
  then have ⟨b,Minimum(r,IntervalX(X, r, b, c))⟩∈rb≠Minimum(r,IntervalX(X, r,
b, c))
    unfolding IntervalX˙def using Order˙ZF˙2˙L1 by auto
  with assms(3,5) sub p obtain z1 where z1:z1∈Xz1≠bz1≠Minimum(r,IntervalX(X,
r, b, c))⟨b,z1⟩∈r⟨z1,Minimum(r,IntervalX(X, r, b, c))⟩∈r
    unfolding IsDense˙def by blast
  from p have B:⟨Minimum(r,IntervalX(X, r, b, c)),c⟩∈r unfolding IntervalX˙def
using Order˙ZF˙2˙L1 by auto moreover
  have trans(r) using assms(1) unfolding IsLinOrder˙def by auto moreover
  note z1(5) ultimately have z1a:⟨z1,c⟩∈r unfolding trans˙def by fast
  {
    assume z1=c
    with B have ⟨Minimum(r,IntervalX(X, r, b, c)),z1⟩∈r by auto
    with z1(5) have z1=Minimum(r,IntervalX(X, r, b, c)) using assms(1) unfolding
IsLinOrder˙def antisym˙def by auto
    then have False using z1(3) by auto
  }
  then have z1≠c by auto
  with z1(1,2,4) z1a have z1∈IntervalX(X, r, b, c) unfolding IntervalX˙def using
Order˙ZF˙2˙L1 by auto
  then have ⟨Minimum(r,IntervalX(X, r, b, c)),z1⟩∈r using Finite˙ZF˙1˙T2(4)[OF
assms(1) Finite˙Fin[OF fin sub] assms(2)] by auto
  with z1(5) have z1=Minimum(r,IntervalX(X, r, b, c)) using assms(1) unfolding
IsLinOrder˙def antisym˙def by auto
  with z1(3) show False by auto
qed
```

Left rays are infinite.

```
theorem dense˙order˙inf˙lrays:
  assumes IsLinOrder(X,r) LeftRayX(X,r,c)≠0c∈X  X–is dense with respect to″r
  shows ¬Finite(LeftRayX(X,r,c))
proof-
  from assms(2) obtain b where b∈X⟨b,c⟩∈rb≠c unfolding LeftRayX˙def by auto
  with assms(3) obtain z where z∈X-–b,c″⟨b,z⟩∈r⟨z,c⟩∈r using assms(4) unfolding
IsDense˙def by auto
  then have IntervalX(X, r, b, c)≠0 unfolding IntervalX˙def using Order˙ZF˙2˙L1 by
auto
  then have nFIN:¬Finite(IntervalX(X, r, b, c)) using dense˙order˙inf˙intervals[OF
assms(1) ˙ ˙ assms(3,4)]
    ‹b∈X› by auto
  {
    fix d assume d∈IntervalX(X, r, b, c)
```

then have ⟨b,d⟩∈r⟨d,c⟩∈rd∈Xd≠bd≠c unfolding IntervalX˙def using Order˙ZF˙2˙L1 by auto
then have d∈LeftRayX(X,r,c) unfolding LeftRayX˙def by auto
}
then have IntervalX(X, r, b, c)⊆LeftRayX(X,r,c) by auto
with nFIN show thesis using subset˙Finite by auto
qed

Right rays are infinite.

theorem dense˙order˙inf˙rrays:
assumes IsLinOrder(X,r) RightRayX(X,r,b)≠0b∈X X–is dense with respect to″r
shows ¬Finite(RightRayX(X,r,b))
proof-
from assms(2) obtain c where c∈X⟨b,c⟩∈rb≠c unfolding RightRayX˙def by auto
with assms(3) obtain z where z∈X-–b,c″⟨b,z⟩∈r⟨z,c⟩∈r using assms(4) unfolding IsDense˙def by auto
then have IntervalX(X, r, b, c)≠0 unfolding IntervalX˙def using Order˙ZF˙2˙L1 by auto
then have nFIN:¬Finite(IntervalX(X, r, b, c)) using dense˙order˙inf˙intervals[OF assms(1) ˙ assms(3) ˙ assms(4)]
‹c∈X› by auto
{
fix d assume d∈IntervalX(X, r, b, c)
then have ⟨b,d⟩∈r⟨d,c⟩∈rd∈Xd≠bd≠c unfolding IntervalX˙def using Order˙ZF˙2˙L1 by auto
then have d∈RightRayX(X,r,b) unfolding RightRayX˙def by auto
}
then have IntervalX(X, r, b, c)⊆RightRayX(X,r,b) by auto
with nFIN show thesis using subset˙Finite by auto
qed

The whole space in a densely ordered set is infinite.

corollary dense˙order˙infinite:
assumes IsLinOrder(X,r) X–is dense with respect to″r
∃x y. x≠y∧x∈X∧y∈X
shows ¬(X≺nat)
proof-
from assms(3) obtain b c where B:b∈Xc∈Xb≠c by auto
{
assume ⟨b,c⟩∉r
with assms(1) have ⟨c,b⟩∈r unfolding IsLinOrder˙def IsTotal˙def using ‹b∈X›‹c∈X› by auto
with assms(2) B obtain z where z∈X-–b,c″⟨c,z⟩∈r⟨z,b⟩∈r unfolding IsDense˙def by auto
then have IntervalX(X,r,c,b)≠0 unfolding IntervalX˙def using Order˙ZF˙2˙L1 by auto
then have ¬(Finite(IntervalX(X,r,c,b))) using dense˙order˙inf˙intervals[OF assms(1) ˙ ‹c∈X›‹b∈X› assms(2)]
by auto moreover

```
    have IntervalX(X,r,c,b)⊆X unfolding IntervalX˙def by auto
    ultimately have ¬(Finite(X)) using subset˙Finite by auto
    then have ¬(X≺nat) using lesspoll˙nat˙is˙Finite by auto
  }
  moreover
  {
    assume ⟨b,c⟩∈r
    with assms(2) B obtain z where z∈X-–b,c″⟨b,z⟩∈r⟨z,c⟩∈r unfolding IsDense˙def
by auto
    then have IntervalX(X,r,b,c)≠0 unfolding IntervalX˙def using Order˙ZF˙2˙L1 by
auto
    then have ¬(Finite(IntervalX(X,r,b,c))) using dense˙order˙inf˙intervals[OF assms(1)
˙ ‹b∈X›‹c∈X› assms(2)]
      by auto moreover
    have IntervalX(X,r,b,c)⊆X unfolding IntervalX˙def by auto
    ultimately have ¬(Finite(X)) using subset˙Finite by auto
    then have ¬(X≺nat) using lesspoll˙nat˙is˙Finite by auto
  }
  ultimately show thesis by auto
qed
```

If an order topology is connected, then the order is complete. It is equivalent to assume that $r \subseteq X \times X$ or prove that $r \cap X \times X$ is complete.

```
theorem conn˙imp˙complete:
  assumes IsLinOrder(X,r) ∃x y. x≠y∧x∈X∧y∈X r⊆X×X
     (OrdTopology X r)–is connected″
  shows r–is complete″
proof-
  {
    assume ¬(r–is complete″)
    then obtain A where A:A≠0IsBoundedAbove(A,r)¬(HasAminimum(r, ⋂b∈A. r
–b″)) unfolding
      IsComplete˙def by auto
     from A(3) have r1:∀m∈⋂b∈A. r  –b″. ∃x∈⋂b∈A. r  –b″. ⟨m,x⟩∉r unfolding
HasAminimum˙def
      by force
    from A(1,2) obtain b where r2:∀x∈A. ⟨x, b⟩ ∈ r unfolding IsBoundedAbove˙def
by auto
    with assms(3) A(1) have A⊆Xb∈X by auto
    with assms(3) have r3:∀c∈A. r  –c″⊆X using image˙iff by auto
    from r2 have ∀x∈A. b∈r–x″ using image˙iff by auto
    then have noE:b∈(⋂b∈A. r  –b″) using A(1) by auto
    {
      fix x assume x∈(⋂b∈A. r  –b″)
      then have ∀c∈A. x∈r–c″ by auto
      with A(1) obtain c where c∈A x∈r–c″ by auto
      with r3 have x∈X by auto
    }
    then have sub:(⋂b∈A. r  –b″)⊆X by auto
```

```
{
  fix x assume x:x∈(⋂b∈A. r `-b")
  with r1 have ∃z∈⋂b∈A. r `-b". ⟨x,z⟩∉r by auto
  then obtain z where z:z∈(⋂b∈A. r `-b")⟨x,z⟩∉r by auto
  from x z(1) sub have x∈Xz∈X by auto
  with z(2) have ⟨z,x⟩∈r using assms(1) unfolding IsLinOrder˙def IsTotal˙def by
auto
then have xx:x∈RightRayX(X,r,z) unfolding RightRayX˙def using ‹x∈X›‹⟨x,z⟩∉r›
   assms(1) unfolding IsLinOrder˙def using total˙is˙refl unfolding refl˙def by auto
  {
    fix m assume m∈RightRayX(X,r,z)
    then have m:m∈X-`-z"⟨z,m⟩∈r unfolding RightRayX˙def by auto
    {
      fix c assume c∈A
      with z(1) have ⟨c,z⟩∈r using image˙iff by auto
      with m(2) have ⟨c,m⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def
by fast
      then have m∈r`-c" using image˙iff by auto
    }
    with A(1) have m∈(⋂b∈A. r `-b") by auto
  }
  then have sub1:RightRayX(X,r,z)⊆(⋂b∈A. r `-b") by auto
  have RightRayX(X,r,z)∈(OrdTopology X r) using
    base˙sets˙open[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]] ‹z∈X› by auto
 with sub1 xx have ∃U∈(OrdTopology X r). x∈U ∧ U⊆(⋂b∈A. r `-b") by auto
}
then have (⋂b∈A. r `-b")∈(OrdTopology X r) using topology0.open˙neigh˙open[OF
topology0˙ordtopology[OF assms(1)]]
  by auto moreover
{
  fix x assume x∈X-(⋂b∈A. r `-b")
  then have x∈Xx∉(⋂b∈A. r `-b") by auto
  with A(1) obtain b where x∉r`-b"b∈A by auto
  then have ⟨b,x⟩∉r using image˙iff by auto
 with ‹A⊆X› ‹b∈A›‹x∈X› have ⟨x,b⟩∈r using assms(1) unfolding IsLinOrder˙def
    IsTotal˙def by auto
 then have xx:x∈LeftRayX(X,r,b) unfolding LeftRayX˙def using ‹x∈X› ‹⟨b,x⟩∉r›
   assms(1) unfolding IsLinOrder˙def using total˙is˙refl unfolding refl˙def by auto
  {
    fix y assume y∈LeftRayX(X,r,b)∩(⋂b∈A. r `-b")
    then have y∈X-`-b"⟨y,b⟩∈r∀c∈A. y∈r`-c" unfolding LeftRayX˙def by auto
    then have y∈X⟨y,b⟩∈r∀c∈A. ⟨c,y⟩∈r using image˙iff by auto
     with ‹b∈A› have y=b using assms(1) unfolding IsLinOrder˙def antisym˙def
by auto
    then have False using ‹y∈X-`-b"› by auto
  }
  then have sub1:LeftRayX(X,r,b)⊆X-(⋂b∈A. r `-b") unfolding LeftRayX˙def
by auto
  have LeftRayX(X,r,b)∈(OrdTopology X r) using
```

```
        base˙sets˙open[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]] ‹b∈A›‹A⊆X›
by blast
      with sub1 xx have ∃U∈(OrdTopology X r). x∈U∧U⊆X-(⋂b∈A. r `b″) by
auto
    }
  then have X - (⋂b∈A. r `b″)∈(OrdTopology X r) using topology0.open˙neigh˙open[OF
topology0˙ordtopology[OF assms(1)]]
      by auto
     then have ⋃(OrdTopology X r)-(⋂b∈A. r `b″)∈(OrdTopology X r) using
union˙ordtopology[OF assms(1,2)] by auto
   then have (⋂b∈A. r `b″)`is closed in″(OrdTopology X r) unfolding IsClosed˙def
using union˙ordtopology[OF assms(1,2)]
      sub by auto
    moreover note assms(4) ultimately
      have (⋂b∈A. r `b″)=0∨(⋂b∈A. r `b″)=X using union˙ordtopology[OF
assms(1,2)] unfolding IsConnected˙def
      by auto
    then have e1:(⋂b∈A. r `b″)=X using noE by auto
    then have ∀x∈X. ∀b∈A. x∈r`b″ by auto
    then have r4:∀x∈X. ∀b∈A. ⟨b,x⟩∈r using image˙iff by auto
    {
      fix a1 a2 assume aA:a1∈Aa2∈Aa1≠a2
      with ‹A⊆X› have aX:a1∈Xa2∈X by auto
      with r4 aA(1,2) have ⟨a1,a2⟩∈r⟨a2,a1⟩∈r by auto
     then have a1=a2 using assms(1) unfolding IsLinOrder˙def antisym˙def by auto
      with aA(3) have False by auto
    }
    moreover
    from A(1) obtain t where t∈A by auto
    ultimately have A=`t″ by auto
    with r4 have ∀x∈X. ⟨t,x⟩∈rt∈X using ‹A⊆X› by auto
    then have HasAminimum(r,X) unfolding HasAminimum˙def by auto
    with e1 have HasAminimum(r,⋂b∈A. r `b″) by auto
    with A(3) have False by auto
  }
  then show thesis by auto
qed
```

If an order topology is connected, then the order is dense.

```
theorem conn˙imp˙dense:
  assumes IsLinOrder(X,r) ∃x y. x≠y∧x∈X∧y∈X
     (OrdTopology X r)`is connected″
  shows X `is dense with respect to″r
proof-
  {
    assume ¬(X `is dense with respect to″r)
   then have ∃x1∈X. ∃x2∈X. ⟨x1,x2⟩∈r∧x1≠x2∧(∀z∈X-`x1,x2″. ⟨x1,z⟩∉r∨⟨z,x2⟩∉r)
      unfolding IsDense˙def by auto
   then obtain x1 x2 where x:x1∈Xx2∈X⟨x1,x2⟩∈rx1≠x2(∀z∈X-`x1,x2″. ⟨x1,z⟩∉r∨⟨z,x2⟩∉r)
```

```
by auto
  from x(1,2) have P:LeftRayX(X,r,x2)∈(OrdTopology X r)RightRayX(X,r,x1)∈(OrdTopology
X r)
     using  base˙sets˙open[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]] by auto
   {
     fix x assume x∈X-LeftRayX(X,r,x2)
     then have x∈X x∉LeftRayX(X,r,x2) by auto
     then have ⟨x,x2⟩∉r∨x=x2 unfolding LeftRayX˙def by auto
   then have ⟨x2,x⟩∈r∨x=x2 using assms(1) ‹x∈X› ‹x2∈X› unfolding IsLinOrder˙def
       IsTotal˙def by auto
    then have s:⟨x2,x⟩∈r using assms(1) unfolding IsLinOrder˙def using total˙is˙refl
‹x2∈X›
       unfolding refl˙def by auto
     with x(3) have ⟨x1,x⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def by
fast
     then have x=x1∨x∈RightRayX(X,r,x1) unfolding RightRayX˙def using ‹x∈X›
by auto
     with s have ⟨x2,x1⟩∈r∨x∈RightRayX(X,r,x1) by auto
       with x(3) have x1=x2 ∨ x∈RightRayX(X,r,x1) using assms(1) unfolding
IsLinOrder˙def
       antisym˙def by auto
     with x(4) have x∈RightRayX(X,r,x1) by auto
   }
   then have X-LeftRayX(X,r,x2)⊆RightRayX(X,r,x1) by auto moreover
   {
     fix x assume x∈RightRayX(X,r,x1)
     then have xr:x∈X-–x1″⟨x1,x⟩∈r unfolding RightRayX˙def by auto
     {
       assume x∈LeftRayX(X,r,x2)
       then have xl:x∈X-–x2″⟨x,x2⟩∈r unfolding LeftRayX˙def by auto
       from xl xr x(5) have False by auto
     }
     with xr(1) have x∈X-LeftRayX(X,r,x2) by auto
   }
   ultimately have RightRayX(X,r,x1)=X-LeftRayX(X,r,x2) by auto
  then have LeftRayX(X,r,x2)–is closed in″(OrdTopology X r) using P(2) union˙ordtopology[
     OF assms(1,2)] unfolding IsClosed˙def LeftRayX˙def by auto
  with P(1) have LeftRayX(X,r,x2)=0∨LeftRayX(X,r,x2)=X using union˙ordtopology[
     OF assms(1,2)] assms(3) unfolding IsConnected˙def by auto
   with x(1,3,4) have LeftRayX(X,r,x2)=X unfolding LeftRayX˙def by auto
   then have x2∈LeftRayX(X,r,x2) using x(2) by auto
   then have False unfolding LeftRayX˙def by auto
  }
  then show thesis by auto
qed
```

Actually a connected order topology is one that comes from a dense and complete order.

First a lemma. In a complete ordered set, every non-empty set bounded

from below has a maximum lower bound.

```
lemma complete˙order˙bounded˙below:
  assumes r–is complete″ IsBoundedBelow(A,r) A≠0 r⊆X×X
  shows HasAmaximum(r,⋂c∈A. r-–c″)
proof-
  let M=⋂c∈A. r-–c″
  from assms(3) obtain t where A:t∈A by auto
  {
    fix m assume m∈M
    with A have m∈r-–t″ by auto
    then have ⟨m,t⟩∈r by auto
  }
  then have (∀x∈⋂c∈A. r - –c″. ⟨x, t⟩ ∈ r) by auto
  then have IsBoundedAbove(M,r) unfolding IsBoundedAbove˙def by auto
  moreover
  from assms(2,3) obtain l where  ∀x∈A. ⟨l, x⟩ ∈ r unfolding IsBoundedBelow˙def
by auto
  then have ∀x∈A. l ∈ r-–x″ using vimage˙iff by auto
  with assms(3) have l∈M by auto
  then have M≠0 by auto moreover note assms(1)
 ultimately have HasAminimum(r,⋂c∈M. r  –c″) unfolding IsComplete˙def by auto
  then obtain rr where rr:rr∈(⋂c∈M. r  –c″) ∀s∈(⋂c∈M. r  –c″). ⟨rr,s⟩∈r unfolding
HasAminimum˙def
    by auto
  {
    fix aa assume A:aa∈A
    {
      fix c assume M:c∈M
      with A have ⟨c,aa⟩∈r by auto
      then have aa∈r–c″ by auto
    }
    then have aa∈(⋂c∈M. r  –c″) using rr(1) by auto
  }
  then have A⊆(⋂c∈M. r  –c″) by auto
  with rr(2) have ∀s∈A. ⟨rr,s⟩∈r by auto
  then have rr∈M using assms(3) by auto
  moreover
  {
    fix m assume m∈M
    then have rr∈r–m″ using rr(1) by auto
    then have ⟨m,rr⟩∈r by auto
  }
  then have ∀m∈M. ⟨m,rr⟩∈r by auto
  ultimately show thesis unfolding HasAmaximum˙def by auto
qed

theorem comp˙dense˙imp˙conn:
  assumes IsLinOrder(X,r) ∃x y. x≠y∧x∈X∧y∈X r⊆X×X
     X –is dense with respect to″r r–is complete″
```

shows (OrdTopology X r)–is connected″
proof-
{
assume ¬((OrdTopology X r)–is connected″)
then obtain U where U:U≠0U≠XU∈(OrdTopology X r)U–is closed in″(OrdTopology X r)
unfolding IsConnected˙def using union˙ordtopology[OF assms(1,2)] by auto
from U(4) have A:X-U∈(OrdTopology X r)U⊆X unfolding IsClosed˙def using union˙ordtopology[OF assms(1,2)] by auto
from U(1) obtain u where u∈U by auto
from A(2) U(1,2) have X-U≠0 by auto
then obtain v where v∈X-U by auto
with ‹u∈U› ‹U⊆X› have ⟨u,v⟩∈r∨⟨v,u⟩∈r using assms(1) unfolding IsLinOrder˙def IsTotal˙def
by auto
{
assume ⟨u,v⟩∈r
have LeftRayX(X,r,v)∈(OrdTopology X r) using base˙sets˙open[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]]
‹v∈X-U› by auto
then have U∩LeftRayX(X,r,v)∈(OrdTopology X r) using U(3) using Ordtopology˙is˙a˙topology(1)
[OF assms(1)] unfolding IsATopology˙def by auto
{
fix b assume b∈(U)∩LeftRayX(X,r,v)
then have ⟨b,v⟩∈r unfolding LeftRayX˙def by auto
}
then have bound:IsBoundedAbove(U∩LeftRayX(X,r,v),r) unfolding IsBoundedAbove˙def by auto moreover
with ‹⟨u,v⟩∈r›‹u∈U›‹U⊆X›‹v∈X-U› have nE:U∩LeftRayX(X,r,v)≠0 unfolding LeftRayX˙def by auto
ultimately have Hmin:HasAminimum(r,⋂c∈U∩LeftRayX(X,r,v). r–c″) using assms(5) unfolding IsComplete˙def
by auto
let min=Supremum(r,U∩LeftRayX(X,r,v))
{
fix c assume c∈U∩LeftRayX(X,r,v)
then have ⟨c,v⟩∈r unfolding LeftRayX˙def by auto
}
then have a1:⟨min,v⟩∈r using Order˙ZF˙5˙L3[OF ˙ nE Hmin] assms(1) unfolding IsLinOrder˙def
by auto
{
assume ass:min∈U
then obtain V where V:min∈VV⊆U
V∈–IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″∪–LeftRayX(X,r,b). b∈X″∪–RightRayX(X,r,b). b∈X″ using point˙open˙base˙neigh
[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)] ‹U∈(OrdTopology X r)› ass]
by blast

```
{
  assume V∈–RightRayX(X,r,b). b∈X″
  then obtain b where b:b∈X V=RightRayX(X,r,b) by auto
  note a1 moreover
  from V(1) b(2) have a2:⟨b,min⟩∈rmin≠b unfolding RightRayX˙def by auto
   ultimately have ⟨b,v⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def
by blast moreover
  {
    assume b=v
        with a1 a2(1) have b=min using assms(1) unfolding IsLinOrder˙def
antisym˙def by auto
    with a2(2) have False by auto
  }
     ultimately have False using V(2) b(2) unfolding RightRayX˙def using
‹v∈X-U› by auto
}
moreover
{
  assume V∈–LeftRayX(X,r,b). b∈X″
  then obtain b where b:V=LeftRayX(X,r,b) b∈X by auto
  {
    assume ⟨v,b⟩∈r
  then have b=v∨v∈LeftRayX(X,r,b) unfolding LeftRayX˙def using ‹v∈X-U›
by auto
    then have b=v using b(1) V(2) ‹v∈X-U› by auto
  }
   then have bv:⟨b,v⟩∈r using assms(1) unfolding IsLinOrder˙def IsTotal˙def
using b(2)
    ‹v∈X-U› by auto
  from b(1) V(1) have ⟨min,b⟩∈rmin≠b unfolding LeftRayX˙def by auto
     with assms(4) obtain z where z:⟨min,z⟩∈r⟨z,b⟩∈rz∈X-–b,min″ unfolding
IsDense˙def
    using b(2) V(1,2) ‹U⊆X› by blast
  then have rayb:z∈LeftRayX(X,r,b) unfolding LeftRayX˙def by auto
 from z(2) bv have ⟨z,v⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def
by fast
  moreover
  {
    assume z=v
    with bv have ⟨b,z⟩∈r by auto
     with z(2) have b=z using assms(1) unfolding IsLinOrder˙def antisym˙def
by auto
    then have False using z(3) by auto
  }
   ultimately have z∈LeftRayX(X,r,v) unfolding LeftRayX˙def using z(3) by
auto
  with rayb have z∈U∩LeftRayX(X,r,v) using V(2) b(1) by auto
 then have min∈r–z″ using Order˙ZF˙4˙L4(1)[OF ˙ Hmin] assms(1) unfolding
Supremum˙def IsLinOrder˙def
```

by auto
then have ⟨z,min⟩∈r by auto
with z(1,3) have False using assms(1) unfolding IsLinOrder˙def antisym˙def by auto
}
moreover
{
assume V∈–IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″
then obtain b c where b:V=IntervalX(X,r,b,c) b∈Xc∈X by auto
from b V(1) have m:⟨min,c⟩∈r⟨b,min⟩∈rmin≠b min≠c unfolding IntervalX˙def Interval˙def by auto
{
assume A:⟨c,v⟩∈r
from m obtain z where z:⟨z,c⟩∈r ⟨min,z⟩∈rz∈X-–c,min″ using assms(4) unfolding IsDense˙def
using b(3) V(1,2) ‹U⊆X› by blast
from z(2) have ⟨b,z⟩∈r using m(2) assms(1) unfolding IsLinOrder˙def trans˙def
by fast
with z(1) have z∈IntervalX(X,r,b,c)∨z=b using z(3) unfolding IntervalX˙def
Interval˙def by auto
then have z∈IntervalX(X,r,b,c) using m(2) z(2,3) using assms(1) unfolding IsLinOrder˙def
antisym˙def by auto
with b(1) V(2) have z∈U by auto moreover
from A z(1) have ⟨z,v⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def by fast
moreover have z≠v using A z(1,3) assms(1) unfolding IsLinOrder˙def antisym˙def by auto
ultimately have z∈U∩LeftRayX(X,r,v) unfolding LeftRayX˙def using z(3) by auto
then have min∈r–z″ using Order˙ZF˙4˙L4(1)[OF ˙ Hmin] assms(1) unfolding Supremum˙def IsLinOrder˙def
by auto
then have ⟨z,min⟩∈r by auto
with z(2,3) have False using assms(1) unfolding IsLinOrder˙def antisym˙def by auto
}
then have vc:⟨v,c⟩∈rv≠c using assms(1) unfolding IsLinOrder˙def IsTotal˙def using ‹v∈X-U›
b(3) by auto
{
assume min=v
with V(2,1) ‹v∈X-U› have False by auto
}
then have min≠v by auto
with a1 obtain z where z:⟨min,z⟩∈r⟨z,v⟩∈rz∈X-–min,v″ using assms(4) unfolding IsDense˙def
using V(1,2) ‹U⊆X›‹v∈X-U› by blast

```
          from z(2) vc(1) have zc:⟨z,c⟩∈r using assms(1) unfolding IsLinOrder˙def
trans˙def
          by fast moreover
      from m(2) z(1) have ⟨b,z⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def
          by fast ultimately
        have z∈Interval(r,b,c) using Order˙ZF˙2˙L1B by auto moreover
        {
          assume z=c
            then have False using z(2) vc using assms(1) unfolding IsLinOrder˙def
antisym˙def
            by fast
        }
        then have z≠c by auto moreover
        {
          assume z=b
         then have z=min using m(2) z(1) using assms(1) unfolding IsLinOrder˙def
            antisym˙def by auto
          with z(3) have False by auto
        }
        then have z≠b by auto moreover
        have z∈X using z(3) by auto ultimately
        have z∈IntervalX(X,r,b,c) unfolding IntervalX˙def by auto
        then have z∈V using b(1) by auto
        then have z∈U using V(2) by auto moreover
            from z(2,3) have z∈LeftRayX(X,r,v) unfolding LeftRayX˙def by auto
ultimately
        have z∈U∩LeftRayX(X,r,v) by auto
       then have min∈r–z″ using Order˙ZF˙4˙L4(1)[OF ˙ Hmin] assms(1) unfolding
Supremum˙def IsLinOrder˙def
          by auto
        then have ⟨z,min⟩∈r by auto
        with z(1,3) have False using assms(1) unfolding IsLinOrder˙def antisym˙def
by auto
      }
      ultimately have False using V(3) by auto
    }
    then have ass:min∈X-U using a1 assms(3) by auto
    then obtain V where V:min∈VV⊆X-U
   V∈–IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″∪–LeftRayX(X,r,b). b∈X″∪–RightRayX(X,r,b).
b∈X″ using point˙open˙base˙neigh
       [OF Ordtopology˙is˙a˙topology(2)[OF assms(1)] ‹X-U∈(OrdTopology X r)›
ass] by blast
    {
      assume V∈–IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″
      then obtain b c where b:V=IntervalX(X,r,b,c)b∈Xc∈X by auto
    from b V(1) have m:⟨min,c⟩∈r⟨b,min⟩∈rmin≠b min≠c unfolding IntervalX˙def
Interval˙def by auto
      {
        fix x assume A:x∈U∩LeftRayX(X,r,v)
```

then have ⟨x,v⟩∈rx∈U unfolding LeftRayX˙def by auto
then have x∉V using V(2) by auto
then have x∉Interval(r, b, c) ∩ X∨x=b∨x=c using b(1) unfolding IntervalX˙def by auto
then have (⟨b,x⟩∉r∨⟨x,c⟩∉r)∨x=b∨x=cx∈X using Order˙ZF˙2˙L1B ‹x∈U›‹U⊆X› by auto
then have (⟨x,b⟩∈r∨⟨c,x⟩∈r)∨x=b∨x=c using assms(1) unfolding IsLinOrder˙def IsTotal˙def
using b(2,3) by auto
then have (⟨x,b⟩∈r∨⟨c,x⟩∈r) using assms(1) unfolding IsLinOrder˙def using total˙is˙refl
unfolding refl˙def using b(2,3) by auto moreover
from A have ⟨x,min⟩∈r using Order˙ZF˙4˙L4(1)[OF ˙ Hmin] assms(1) unfolding Supremum˙def IsLinOrder˙def
by auto
ultimately have (⟨x,b⟩∈r∨⟨c,min⟩∈r) using assms(1) unfolding IsLinOrder˙def trans˙def
by fast
with m(1) have (⟨x,b⟩∈r∨c=min) using assms(1) unfolding IsLinOrder˙def antisym˙def by auto
with m(4) have ⟨x,b⟩∈r by auto
}
then have ⟨min,b⟩∈r using Order˙ZF˙5˙L3[OF ˙ nE Hmin] assms(1) unfolding IsLinOrder˙def by auto
with m(2,3) have False using assms(1) unfolding IsLinOrder˙def antisym˙def by auto
}
moreover
{
assume V∈–RightRayX(X,r,b). b∈X″
then obtain b where b:V=RightRayX(X,r,b) b∈X by auto
from b V(1) have m:⟨b,min⟩∈rmin≠b unfolding RightRayX˙def by auto
{
fix x assume A:x∈U∩LeftRayX(X,r,v)
then have ⟨x,v⟩∈rx∈U unfolding LeftRayX˙def by auto
then have x∉V using V(2) by auto
then have x∉RightRayX(X,r, b) using b(1) by auto
then have (⟨b,x⟩∉r∨x=b)x∈X unfolding RightRayX˙def using ‹x∈U›‹U⊆X› by auto
then have ⟨x,b⟩∈r using assms(1) unfolding IsLinOrder˙def using total˙is˙refl unfolding
refl˙def unfolding IsTotal˙def using b(2) by auto
}
then have ⟨min,b⟩∈r using Order˙ZF˙5˙L3[OF ˙ nE Hmin] assms(1) unfolding IsLinOrder˙def by auto
with m(2,1) have False using assms(1) unfolding IsLinOrder˙def antisym˙def by auto
} moreover
{

```
assume V∈–LeftRayX(X,r,b). b∈X″
then obtain b where b:V=LeftRayX(X,r,b) b∈X by auto
from b V(1) have m:⟨min,b⟩∈rmin≠b unfolding LeftRayX˙def by auto
{
  fix x assume A:x∈U∩LeftRayX(X,r,v)
  then have ⟨x,v⟩∈rx∈U unfolding LeftRayX˙def by auto
  then have x∉V using V(2) by auto
  then have x∉LeftRayX(X,r, b) using b(1) by auto
   then have (⟨x,b⟩∉r∨x=b)x∈X unfolding LeftRayX˙def using ‹x∈U›‹U⊆X›
by auto
 then have ⟨b,x⟩∈r using assms(1) unfolding IsLinOrder˙def using total˙is˙refl
unfolding
    refl˙def unfolding IsTotal˙def using b(2) by auto
 with m(1) have ⟨min,x⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def
by fast
  moreover
  from bound A have ∃g. ∀y∈U∩LeftRayX(X,r,v). ⟨y,g⟩∈r using nE
    unfolding IsBoundedAbove˙def by auto
  then obtain g where g:∀y∈U∩LeftRayX(X,r,v). ⟨y,g⟩∈r by auto
  with nE obtain t where t∈U∩LeftRayX(X,r,v) by auto
  with g have ⟨t,g⟩∈r by auto
  with assms(3) have g∈X by auto
  with g have boundX:∃g∈X. ∀y∈U∩LeftRayX(X,r,v). ⟨y,g⟩∈r by auto
  have ⟨x,min⟩∈r using Order˙ZF˙5˙L7(2)[OF assms(3) ˙ assms(5) ˙  boundX]
    assms(1) ‹U⊆X› A unfolding LeftRayX˙def IsLinOrder˙def by auto
 ultimately have x=min using assms(1) unfolding IsLinOrder˙def antisym˙def
by auto
}
then have U∩LeftRayX(X,r,v)⊆–min″ by auto moreover
{
  assume min∈U∩LeftRayX(X,r,v)
  then have min∈U by auto
  then have False using V(1,2) by auto
}
ultimately have False using nE by auto
}
moreover note V(3)
ultimately have False by auto
}
with assms(1) have ⟨v,u⟩∈r unfolding IsLinOrder˙def IsTotal˙def using ‹u∈U›‹U⊆X›
  ‹v∈X-U› by auto
have RightRayX(X,r,v)∈(OrdTopology X r) using base˙sets˙open[OF Ordtopol-
ogy˙is˙a˙topology(2)[OF assms(1)]]
  ‹v∈X-U› by auto
then have U∩RightRayX(X,r,v)∈(OrdTopology X r) using U(3) using Ordtopol-
ogy˙is˙a˙topology(1)
  [OF assms(1)] unfolding IsATopology˙def by auto
{
  fix b assume b∈(U)∩RightRayX(X,r,v)
```

then have ⟨v,b⟩∈r unfolding RightRayX˙def by auto
}
then have bound:IsBoundedBelow(U∩RightRayX(X,r,v),r) unfolding IsBoundedBelow˙def by auto
with ‹⟨v,u⟩∈r›‹u∈U›‹U⊆X›‹v∈X-U› have nE:U∩RightRayX(X,r,v)≠0 unfolding RightRayX˙def by auto
have Hmax:HasAmaximum(r,⋂c∈U∩RightRayX(X,r,v). r-–c″) using complete˙order˙bounded˙below[OF assms(5) bound nE assms(3)].
let max=Infimum(r,U∩RightRayX(X,r,v))
{
fix c assume c∈U∩RightRayX(X,r,v)
then have ⟨v,c⟩∈r unfolding RightRayX˙def by auto
}
then have a1:⟨v,max⟩∈r using Order˙ZF˙5˙L4[OF ˙ nE Hmax] assms(1) unfolding IsLinOrder˙def
by auto
{
assume ass:max∈U
then obtain V where V:max∈VV⊆U
V∈–IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″∪–LeftRayX(X,r,b). b∈X″∪–RightRayX(X,r,b). b∈X″ using point˙open˙base˙neigh
[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)] ‹U∈(OrdTopology X r)› ass] by blast
{
assume V∈–RightRayX(X,r,b). b∈X″
then obtain b where b:b∈X V=RightRayX(X,r,b) by auto
from V(1) b(2) have a2:⟨b,max⟩∈rmax≠b unfolding RightRayX˙def by auto
{
assume ⟨b,v⟩∈r
then have b=v∨v∈RightRayX(X,r,b) unfolding RightRayX˙def using ‹v∈X-U› by auto
then have b=v using b(2) V(2) ‹v∈X-U› by auto
}
then have bv:⟨v,b⟩∈r using assms(1) unfolding IsLinOrder˙def IsTotal˙def using b(1)
‹v∈X-U› by auto
from a2 assms(4) obtain z where z:⟨b,z⟩∈r⟨z,max⟩∈rz∈X-–b,max″ unfolding IsDense˙def
using b(1) V(1,2) ‹U⊆X› by blast
then have rayb:z∈RightRayX(X,r,b) unfolding RightRayX˙def by auto
from z(1) bv have ⟨v,z⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def by fast moreover
{
assume z=v
with bv have ⟨z,b⟩∈r by auto
with z(1) have b=z using assms(1) unfolding IsLinOrder˙def antisym˙def by auto
then have False using z(3) by auto
}

ultimately have z∈RightRayX(X,r,v) unfolding RightRayX˙def using z(3) by auto
with rayb have z∈U∩RightRayX(X,r,v) using V(2) b(2) by auto
then have max∈r-–z″ using Order˙ZF˙4˙L3(1)[OF ˙ Hmax] assms(1) unfolding Infimum˙def IsLinOrder˙def
by auto
then have ⟨max,z⟩∈r by auto
with z(2,3) have False using assms(1) unfolding IsLinOrder˙def antisym˙def by auto
}
moreover
{
assume V∈–LeftRayX(X,r,b). b∈X″
then obtain b where b:V=LeftRayX(X,r,b) b∈X by auto
note a1 moreover
from V(1) b(1) have a2:⟨max,b⟩∈rmax≠b unfolding LeftRayX˙def by auto
ultimately have ⟨v,b⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def by blast moreover
{
assume b=v
with a1 a2(1) have b=max using assms(1) unfolding IsLinOrder˙def antisym˙def by auto
with a2(2) have False by auto
}
ultimately have False using V(2) b(1) unfolding LeftRayX˙def using ‹v∈X-U› by auto
}
moreover
{
assume V∈–IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″
then obtain b c where b:V=IntervalX(X,r,b,c) b∈Xc∈X by auto
from b V(1) have m:⟨max,c⟩∈r⟨b,max⟩∈rmax≠b max≠c unfolding IntervalX˙def Interval˙def by auto
{
assume A:⟨v,b⟩∈r
from m obtain z where z:⟨z,max⟩∈r ⟨b,z⟩∈rz∈X-–b,max″ using assms(4) unfolding IsDense˙def
using b(2) V(1,2) ‹U⊆X› by blast
from z(1) have ⟨z,c⟩∈r using m(1) assms(1) unfolding IsLinOrder˙def trans˙def
by fast
with z(2) have z∈IntervalX(X,r,b,c)∨z=c using z(3) unfolding IntervalX˙def
Interval˙def by auto
then have z∈IntervalX(X,r,b,c) using m(1) z(1,3) using assms(1) unfolding IsLinOrder˙def
antisym˙def by auto
with b(1) V(2) have z∈U by auto moreover
from A z(2) have ⟨v,z⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def by fast
moreover have z≠v using A z(2,3) assms(1) unfolding IsLinOrder˙def

```
antisym˙def by auto
          ultimately have z∈U∩RightRayX(X,r,v) unfolding RightRayX˙def using
z(3) by auto
      then have max∈r-−z″ using Order˙ZF˙4˙L3(1)[OF ˙ Hmax] assms(1) unfolding
Infimum˙def IsLinOrder˙def
          by auto
        then have ⟨max,z⟩∈r by auto
        with z(1,3) have False using assms(1) unfolding IsLinOrder˙def antisym˙def
by auto
      }
      then have vc:⟨b,v⟩∈rv≠b using assms(1) unfolding IsLinOrder˙def IsTotal˙def
using ‹v∈X-U›
        b(2) by auto
      {
        assume max=v
        with V(2,1) ‹v∈X-U› have False by auto
      }
      then have v≠max by auto moreover
      note a1 moreover
      have max∈X using V(1,2) ‹U⊆X› by auto
      moreover have v∈X using ‹v∈X-U› by auto
        ultimately obtain z where z:⟨v,z⟩∈r⟨z,max⟩∈rz∈X-−v,max″ using assms(4)
unfolding IsDense˙def
        by auto
         from z(1) vc(1) have zc:⟨b,z⟩∈r using assms(1) unfolding IsLinOrder˙def
trans˙def
        by fast moreover
      from m(1) z(2) have ⟨z,c⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def
        by fast ultimately
      have z∈Interval(r,b,c) using Order˙ZF˙2˙L1B by auto moreover
      {
        assume z=b
           then have False using z(1) vc using assms(1) unfolding IsLinOrder˙def
antisym˙def
          by fast
      }
      then have z≠b by auto moreover
      {
        assume z=c
        then have z=max using m(1) z(2) using assms(1) unfolding IsLinOrder˙def
          antisym˙def by auto
        with z(3) have False by auto
      }
      then have z≠c by auto moreover
      have z∈X using z(3) by auto ultimately
      have z∈IntervalX(X,r,b,c) unfolding IntervalX˙def by auto
      then have z∈V using b(1) by auto
      then have z∈U using V(2) by auto moreover
         from z(1,3) have z∈RightRayX(X,r,v) unfolding RightRayX˙def by auto
```

ultimately
have z∈U∩RightRayX(X,r,v) by auto
then have max∈r-–z″ using Order˙ZF˙4˙L3(1)[OF ˙ Hmax] assms(1) unfolding Infimum˙def IsLinOrder˙def
by auto
then have ⟨max,z⟩∈r by auto
with z(2,3) have False using assms(1) unfolding IsLinOrder˙def antisym˙def by auto
}
ultimately have False using V(3) by auto
}
then have ass:max∈X-U using a1 assms(3) by auto
then obtain V where V:max∈VV⊆X-U
V∈–IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″∪–LeftRayX(X,r,b). b∈X″∪–RightRayX(X,r,b). b∈X″ using point˙open˙base˙neigh
[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)] ‹X-U∈(OrdTopology X r)› ass] by blast
{
assume V∈–IntervalX(X,r,b,c). ⟨b,c⟩∈X×X″
then obtain b c where b:V=IntervalX(X,r,b,c)b∈Xc∈X by auto
from b V(1) have m:⟨max,c⟩∈r⟨b,max⟩∈rmax≠b max≠c unfolding IntervalX˙def Interval˙def by auto
{
fix x assume A:x∈U∩RightRayX(X,r,v)
then have ⟨v,x⟩∈rx∈U unfolding RightRayX˙def by auto
then have x∉V using V(2) by auto
then have x∉Interval(r, b, c) ∩ X∨x=b∨x=c using b(1) unfolding IntervalX˙def by auto
then have (⟨b,x⟩∉r∨⟨x,c⟩∉r)∨x=b∨x=cx∈X using Order˙ZF˙2˙L1B ‹x∈U›‹U⊆X› by auto
then have (⟨x,b⟩∈r∨⟨c,x⟩∈r)∨x=b∨x=c using assms(1) unfolding IsLinOrder˙def IsTotal˙def
using b(2,3) by auto
then have (⟨x,b⟩∈r∨⟨c,x⟩∈r) using assms(1) unfolding IsLinOrder˙def using total˙is˙refl
unfolding refl˙def using b(2,3) by auto moreover
from A have ⟨max,x⟩∈r using Order˙ZF˙4˙L3(1)[OF ˙ Hmax] assms(1) unfolding Infimum˙def IsLinOrder˙def
by auto
ultimately have (⟨max,b⟩∈r∨⟨c,x⟩∈r) using assms(1) unfolding IsLinOrder˙def trans˙def
by fast
with m(2) have (max=b∨⟨c,x⟩∈r) using assms(1) unfolding IsLinOrder˙def antisym˙def by auto
with m(3) have ⟨c,x⟩∈r by auto
}
then have ⟨c,max⟩∈r using Order˙ZF˙5˙L4[OF ˙ nE Hmax] assms(1) unfolding IsLinOrder˙def by auto
with m(1,4) have False using assms(1) unfolding IsLinOrder˙def antisym˙def

```
by auto
  }
  moreover
  {
    assume V∈–RightRayX(X,r,b). b∈X″
    then obtain b where b:V=RightRayX(X,r,b) b∈X by auto
    from b V(1) have m:⟨b,max⟩∈rmax≠b unfolding RightRayX˙def by auto
    {
      fix x assume A:x∈U∩RightRayX(X,r,v)
      then have ⟨v,x⟩∈rx∈U unfolding RightRayX˙def by auto
      then have x∉V using V(2) by auto
      then have x∉RightRayX(X,r, b) using b(1) by auto
       then have (⟨b,x⟩∉r∨x=b)x∈X unfolding RightRayX˙def using ‹x∈U›‹U⊆X›
by auto
      then have ⟨x,b⟩∈r using assms(1) unfolding IsLinOrder˙def using total˙is˙refl
unfolding
        refl˙def unfolding IsTotal˙def using b(2) by auto moreover
    from A have ⟨max,x⟩∈r using Order˙ZF˙4˙L3(1)[OF ˙ Hmax] assms(1) unfolding
Infimum˙def IsLinOrder˙def
        by auto ultimately
      have ⟨max,b⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def by fast
       with m have False using assms(1) unfolding IsLinOrder˙def antisym˙def by
auto
    }
    then have False using nE by auto
  } moreover
  {
     assume V∈–LeftRayX(X,r,b). b∈X″
    then obtain b where b:V=LeftRayX(X,r,b) b∈X by auto
    from b V(1) have m:⟨max,b⟩∈rmax≠b unfolding LeftRayX˙def by auto
    {
      fix x assume A:x∈U∩RightRayX(X,r,v)
      then have ⟨v,x⟩∈rx∈U unfolding RightRayX˙def by auto
      then have x∉V using V(2) by auto
      then have x∉LeftRayX(X,r, b) using b(1) by auto
      then have (⟨x,b⟩∉r∨x=b)x∈X unfolding LeftRayX˙def using ‹x∈U›‹U⊆X› by
auto
      then have ⟨b,x⟩∈r using assms(1) unfolding IsLinOrder˙def using total˙is˙refl
unfolding
        refl˙def unfolding IsTotal˙def using b(2) by auto
      then have b∈r-–x″ by auto
    }
    with nE have b∈(⋂c∈U∩RightRayX(X,r,v). r-–c″) by auto
  then have ⟨b,max⟩∈r unfolding Infimum˙def using Order˙ZF˙4˙L3(2)[OF ˙ Hmax]
assms(1)
      unfolding IsLinOrder˙def by auto
   with m have False using assms(1) unfolding IsLinOrder˙def antisym˙def by auto
  }
  moreover note V(3)
```

```
    ultimately have False by auto
  }
  then show thesis by auto
qed
```

66.4 Numerability axioms

A κ-separable order topology is in relation with order density.

If an order topology has a subset A which is topologically dense, then that subset is weakly order-dense in X.

```
lemma dense˙top˙imp˙Wdense˙ord:
  assumes IsLinOrder(X,r) Closure(A,OrdTopology X r)=X A⊆X ∃x y. x ≠ y ∧ x
∈ X ∧ y ∈ X
  shows A–is weakly dense in″X–with respect to″r
proof-
  {
    fix r1 r2 assume r1∈Xr2∈Xr1≠r2 ⟨r1,r2⟩∈r
      then have IntervalX(X,r,r1,r2)∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪
–LeftRayX(X, r, b) . b ∈ X″ ∪
      –RightRayX(X, r, b) . b ∈ X″ by auto
     then have P:IntervalX(X,r,r1,r2)∈(OrdTopology X r) using base˙sets˙open[OF
Ordtopology˙is˙a˙topology(2)[OF assms(1)]]
      by auto
    have IntervalX(X,r,r1,r2)⊆X unfolding IntervalX˙def by auto
   then have int:Closure(A,OrdTopology X r)∩IntervalX(X,r,r1,r2)=IntervalX(X,r,r1,r2)
using assms(2) by auto
    {
      assume IntervalX(X,r,r1,r2)≠0
     then have A∩(IntervalX(X,r,r1,r2))≠0 using topology0.cl˙inter˙neigh[OF topol-
ogy0˙ordtopology[OF assms(1)] ˙ P , of A]
        using assms(3) union˙ordtopology[OF assms(1,4)] int by auto
    }
    then have (∃z∈A-–r1,r2″. ⟨r1,z⟩∈r∧⟨z,r2⟩∈r)∨IntervalX(X,r,r1,r2)=0 unfolding
IntervalX˙def
      Interval˙def by auto
  }
  then show thesis unfolding IsWeaklyDenseSub˙def by auto
qed
```

Conversely, a weakly order-dense set is topologically dense if it is also considered that: if there is a maximum or a minimum elements whose singletons are open, this points have to be in A. In conclusion, weakly order-density is a property closed to topological density.

Another way to see this: Consider a weakly order-dense set A:

- If X has a maximum and a minimum and $\{min, max\}$ is open: A is topologically dense in $X \setminus \{min, max\}$, where min is the minimum in

X and max is the maximum in X.

- If X has a maximum, $\{max\}$ is open and X has no minimum or $\{min\}$ isn't open: A is topologically dense in $X \setminus \{max\}$, where max is the maximum in X.

- If X has a minimum, $\{min\}$ is open and X has no maximum or $\{max\}$ isn't open A is topologically dense in $X \setminus \{min\}$, where min is the minimum in X.

- If X has no minimum or maximum, or $\{min, max\}$ has no proper open sets: A is topologically dense in X.

```
lemma Wdense˙ord˙imp˙dense˙top:
  assumes IsLinOrder(X,r) A–is weakly dense in″X–with respect to″r A⊆X ∃x y. x
≠ y ∧ x ∈ X ∧ y ∈ X
   HasAminimum(r,X)⟶–Minimum(r,X)″∈(OrdTopology X r)⟶Minimum(r,X)∈A
   HasAmaximum(r,X)⟶–Maximum(r,X)″∈(OrdTopology X r)⟶Maximum(r,X)∈A
  shows Closure(A,OrdTopology X r)=X
proof-
  {
    fix x assume x∈X
  {
    fix U assume ass:x∈UU∈(OrdTopology X r)
    then have ∃V∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪ –LeftRayX(X, r, b) .
b ∈ X″ ∪ –RightRayX(X, r, b) . b ∈ X″ . V⊆U∧x∈V
     using point˙open˙base˙neigh[OF Ordtopology˙is˙a˙topology(2)[OF assms(1)]] by
auto
   then obtain V where V:V∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪ –LeftRayX(X,
r, b) . b ∈ X″ ∪ –RightRayX(X, r, b) . b ∈ X″ V⊆U x∈V
      by blast
    note V(1) moreover
    {
      assume V∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″
      then obtain b c where b:b∈Xc∈XV=IntervalX(X, r, b, c) by auto
      with V(3) have x:⟨b,x⟩∈r ⟨x,c⟩∈r x≠b x≠c unfolding IntervalX˙def Interval˙def
by auto
      then have ⟨b,c⟩∈r using assms(1) unfolding IsLinOrder˙def trans˙def by fast
      moreover from x(1-3) have b≠c using assms(1) unfolding IsLinOrder˙def anti-
sym˙def by fast
      moreover note assms(2) b V(3)
      ultimately have ∃z∈A-–b,c″. ⟨b,z⟩∈r∧⟨z,c⟩∈r unfolding IsWeaklyDenseSub˙def
by auto
      then obtain z where z∈Az≠bz≠c⟨b,z⟩∈r⟨z,c⟩∈r by auto
      with assms(3) have z∈Az∈IntervalX(X, r, b, c) unfolding IntervalX˙def Inter-
val˙def by auto
      then have A∩U≠0 using V(2) b(3) by auto
    }
    moreover
```

```
{
  assume V∈–RightRayX(X, r, b) . b ∈ X″
  then obtain b where b:b∈XV=RightRayX(X, r, b) by auto
  with V(3) have x:⟨b,x⟩∈r b≠x unfolding RightRayX˙def by auto moreover
  note b(1) moreover
  have U⊆⋃(OrdTopology X r) using ass(2) by auto
  then have U⊆X using union˙ordtopology[OF assms(1,4)] by auto
  then have x∈X using ass(1) by auto moreover
  note assms(2) ultimately
  have disj:(∃z∈A-–b,x″. ⟨b,z⟩∈r∧⟨z,x⟩∈r)∨ IntervalX(X, r, b, x) = 0 unfolding
IsWeaklyDenseSub˙def by auto
  {
    assume B:IntervalX(X, r, b, x) = 0
    {
      assume ∃y∈X. ⟨x,y⟩∈r ∧ x≠y
      then obtain y where y:y∈X⟨x,y⟩∈r x≠y by auto
      with x have x∈IntervalX(X,r,b,y) unfolding IntervalX˙def Interval˙def
        using ‹x∈X› by auto moreover
       have ⟨b,y⟩∈r using y(2) x(1) assms(1) unfolding IsLinOrder˙def trans˙def
by fast
       moreover have b≠y using y(2,3) x(1) assms(1) unfolding IsLinOrder˙def
antisym˙def by fast
      ultimately
       have (∃z∈A-–b,y″. ⟨b,z⟩∈r∧⟨z,y⟩∈r) using assms(2) unfolding IsWeakly-
DenseSub˙def
       using y(1) b(1) by auto
      then obtain z where z∈A⟨b,z⟩∈rb≠z by auto
      then have z∈A∩V using b(2) unfolding RightRayX˙def using assms(3) by
auto
      then have z∈A∩U using V(2) by auto
      then have A∩U≠0 by auto
    }
    moreover
    {
      assume R:∀y∈X. ⟨x,y⟩∈r⟶x=y
      {
        fix y assume y∈RightRayX(X,r,b)
        then have y:⟨b,y⟩∈r y∈X-–b″ unfolding RightRayX˙def by auto
        {
          assume A:y≠x
          then have ⟨x,y⟩∉r using R y(2) by auto
          then have ⟨y,x⟩∈r using assms(1) unfolding IsLinOrder˙def IsTotal˙def
            using ‹x∈X› y(2) by auto
          with A y have y∈IntervalX(X,r,b,x) unfolding IntervalX˙def Interval˙def
            by auto
          then have False using B by auto
        }
        then have y=x by auto
      }
```

```
        then have RightRayX(X,r,b)=–x″ using V(3) b(2) by blast
        moreover
        {
          fix t assume T:t∈X
          {
            assume t=x
            then have ⟨t,x⟩∈r using assms(1) unfolding IsLinOrder˙def
              using Order˙ZF˙1˙L1 T by auto
          }
          moreover
          {
            assume t≠x
            then have ⟨x,t⟩∉r using R T by auto
            then have ⟨t,x⟩∈r using assms(1) unfolding IsLinOrder˙def IsTotal˙def
              using T ‹x∈X› by auto
          }
          ultimately have ⟨t,x⟩∈r by auto
        }
        with ‹x∈X› have HM:HasAmaximum(r,X) unfolding HasAmaximum˙def by
auto
      then have Maximum(r,X)∈X∀ t∈X. ⟨t,Maximum(r,X)⟩∈r using Order˙ZF˙4˙L3
assms(1) unfolding IsLinOrder˙def
          by auto
        with R ‹x∈X› have xm:x=Maximum(r,X) by auto
        moreover note b(2)
        ultimately have V=–Maximum(r,X)″ by auto
         then have –Maximum(r,X)″∈(OrdTopology X r) using base˙sets˙open[OF
Ordtopology˙is˙a˙topology(2)[OF assms(1)]]
          V(1) by auto
        with HM have Maximum(r,X)∈A using assms(6) by auto
        with xm have x∈A by auto
        with V(2,3) have A∩U≠0 by auto
      }
      ultimately have A∩U≠0 by auto
    }
    moreover
    {
      assume IntervalX(X, r, b, x) ≠ 0
      with disj have ∃ z∈A-–b,x″. ⟨b,z⟩∈r∧⟨z,x⟩∈r by auto
      then obtain z where z∈Az≠b⟨b,z⟩∈r by auto
      then have z∈Az∈RightRayX(X,r,b) unfolding RightRayX˙def using assms(3)
by auto
      then have z∈A∩U using V(2) b(2) by auto
      then have A∩U≠0 by auto
    }
    ultimately have A∩U≠0 by auto
  }
  moreover
  {
```

```
assume V∈–LeftRayX(X, r, b) . b ∈ X″
then obtain b where b:b∈XV=LeftRayX(X, r, b) by auto
with V(3) have x:⟨x,b⟩∈r b≠x unfolding LeftRayX˙def by auto moreover
note b(1) moreover
have U⊆⋃(OrdTopology X r) using ass(2) by auto
then have U⊆X using union˙ordtopology[OF assms(1,4)] by auto
then have x∈X using ass(1) by auto moreover
note assms(2) ultimately
have disj:(∃z∈A-–b,x″. ⟨x,z⟩∈r∧⟨z,b⟩∈r)∨ IntervalX(X, r, x, b) = 0 unfolding
IsWeaklyDenseSub˙def by auto
{
  assume B:IntervalX(X, r, x, b) = 0
  {
    assume ∃y∈X. ⟨y,x⟩∈r ∧ x≠y
    then obtain y where y:y∈X⟨y,x⟩∈r x≠y by auto
    with x have x∈IntervalX(X,r,y,b) unfolding IntervalX˙def Interval˙def
      using ‹x∈X› by auto moreover
   have ⟨y,b⟩∈r using y(2) x(1) assms(1) unfolding IsLinOrder˙def trans˙def by
fast
      moreover have b≠y using y(2,3) x(1) assms(1) unfolding IsLinOrder˙def
antisym˙def by fast
    ultimately
      have (∃z∈A-–b,y″. ⟨y,z⟩∈r∧⟨z,b⟩∈r) using assms(2) unfolding IsWeakly-
DenseSub˙def
      using y(1) b(1) by auto
    then obtain z where z∈A⟨z,b⟩∈rb≠z by auto
      then have z∈A∩V using b(2) unfolding LeftRayX˙def using assms(3) by
auto
    then have z∈A∩U using V(2) by auto
    then have A∩U≠0 by auto
  }
  moreover
  {
    assume R:∀y∈X. ⟨y,x⟩∈r⟶x=y
    {
      fix y assume y∈LeftRayX(X,r,b)
      then have y:⟨y,b⟩∈r y∈X-–b″ unfolding LeftRayX˙def by auto
      {
        assume A:y≠x
        then have ⟨y,x⟩∉r using R y(2) by auto
        then have ⟨x,y⟩∈r using assms(1) unfolding IsLinOrder˙def IsTotal˙def
          using ‹x∈X› y(2) by auto
        with A y have y∈IntervalX(X,r,x,b) unfolding IntervalX˙def Interval˙def
          by auto
        then have False using B by auto
      }
      then have y=x by auto
    }
    then have LeftRayX(X,r,b)=–x″ using V(3) b(2) by blast
```

```
        moreover
        {
          fix t assume T:t∈X
          {
            assume t=x
            then have ⟨x,t⟩∈r using assms(1) unfolding IsLinOrder˙def
              using Order˙ZF˙1˙L1 T by auto
          }
          moreover
          {
            assume t≠x
            then have ⟨t,x⟩∉r using R T by auto
            then have ⟨x,t⟩∈r using assms(1) unfolding IsLinOrder˙def IsTotal˙def
              using T ‹x∈X› by auto
          }
          ultimately have ⟨x,t⟩∈r by auto
        }
        with ‹x∈X› have HM:HasAminimum(r,X) unfolding HasAminimum˙def by
auto
      then have Minimum(r,X)∈X∀t∈X. ⟨Minimum(r,X),t⟩∈r using Order˙ZF˙4˙L4
assms(1) unfolding IsLinOrder˙def
          by auto
        with R ‹x∈X› have xm:x=Minimum(r,X) by auto
        moreover note b(2)
        ultimately have V=–Minimum(r,X)″ by auto
         then have –Minimum(r,X)″∈(OrdTopology X r) using base˙sets˙open[OF
Ordtopology˙is˙a˙topology(2)[OF assms(1)]]
          V(1) by auto
        with HM have Minimum(r,X)∈A using assms(5) by auto
        with xm have x∈A by auto
        with V(2,3) have A∩U≠0 by auto
      }
      ultimately have A∩U≠0 by auto
    }
    moreover
    {
      assume IntervalX(X, r, x, b) ≠ 0
      with disj have ∃z∈A-–b,x″. ⟨x,z⟩∈r∧⟨z,b⟩∈r by auto
      then obtain z where z∈Az≠b⟨z,b⟩∈r by auto
      then have z∈Az∈LeftRayX(X,r,b) unfolding LeftRayX˙def using assms(3) by
auto
      then have z∈A∩U using V(2) b(2) by auto
      then have A∩U≠0 by auto
    }
    ultimately have A∩U≠0 by auto
  }
  ultimately have A∩U≠0 by auto
 }
 then have ∀U∈(OrdTopology X r). x∈U ⟶ U∩A≠0 by auto
```

```
  moreover note ‹x∈X› moreover
  note assms(3) topology0.inter˙neigh˙cl[OF topology0˙ordtopology[OF assms(1)]]
  union˙ordtopology[OF assms(1,4)] ultimately have x∈Closure(A,OrdTopology X
r)
    by auto
  }
  then have X⊆Closure(A,OrdTopology X r) by auto
  with topology0.Top˙3˙L11(1)[OF topology0˙ordtopology[OF assms(1)]]
    assms(3) union˙ordtopology[OF assms(1,4)] show thesis by auto
qed
```

The conclusion is that an order topology is κ-separable iff there is a set A with cardinality strictly less than κ which is weakly-dense in X.

```
theorem separable˙imp˙wdense:
  assumes (OrdTopology X r)–is separable of cardinal″Q ∃x y. x ≠ y ∧ x ∈ X ∧ y
∈ X
    IsLinOrder(X,r)
  shows ∃A∈Pow(X). A≺Q ∧ (A–is weakly dense in″X–with respect to″r)
proof-
 from assms obtain U where U∈Pow(⋃(OrdTopology X r)) Closure(U,OrdTopology
X r)=⋃(OrdTopology X r) U≺Q
    unfolding IsSeparableOfCard˙def by auto
 then have U∈Pow(X) Closure(U,OrdTopology X r)=X U≺Q using union˙ordtopology[OF
assms(3,2)]
    by auto
  with dense˙top˙imp˙Wdense˙ord[OF assms(3) ˙ ˙ assms(2)] show thesis by auto
qed

theorem wdense˙imp˙separable:
  assumes ∃x y. x ≠ y ∧ x ∈ X ∧ y ∈ X (A–is weakly dense in″X–with respect to″r)
    IsLinOrder(X,r) A≺Q InfCard(Q) A⊆X
  shows (OrdTopology X r)–is separable of cardinal″Q
proof-
  {
    assume Hmin:HasAmaximum(r,X)
    then have MaxX:Maximum(r,X)∈X using Order˙ZF˙4˙L3(1) assms(3) unfolding
IsLinOrder˙def
      by auto
    {
      assume HMax:HasAminimum(r,X)
      then have MinX:Minimum(r,X)∈X using Order˙ZF˙4˙L4(1) assms(3) unfolding
IsLinOrder˙def
        by auto
      let A=A ∪–Maximum(r,X),Minimum(r,X)″
      have Finite(–Maximum(r,X),Minimum(r,X)″) by auto
      then have –Maximum(r,X),Minimum(r,X)″≺nat using n˙lesspoll˙nat
        unfolding Finite˙def using eq˙lesspoll˙trans by auto
      moreover
      from assms(5) have nat≺Q∨nat=Q unfolding InfCard˙def
```

```
      using lt˙Card˙imp˙lesspoll[of Qnat] unfolding lt˙def succ˙def
      using Card˙is˙Ord[of Q] by auto
     ultimately have –Maximum(r,X),Minimum(r,X)″≺Q using lesspoll˙trans by
auto
    with assms(4,5) have C:A≺Q using less˙less˙imp˙un˙less
      by auto
     have WeakDense:A–is weakly dense in″X–with respect to″r using assms(2)
unfolding
      IsWeaklyDenseSub˙def by auto
    from MaxX MinX assms(6) have S:A⊆X by auto
    then have Closure(A,OrdTopology X r)=X using Wdense˙ord˙imp˙dense˙top
      [OF assms(3) WeakDense ˙ assms(1)] by auto
   then have thesis unfolding IsSeparableOfCard˙def using union˙ordtopology[OF
assms(3,1)]
      S C by auto
  }
  moreover
  {
    assume nmin:¬HasAminimum(r,X)
     let A=A ∪–Maximum(r,X)″
    have Finite(–Maximum(r,X)″) by auto
    then have –Maximum(r,X)″≺nat using n˙lesspoll˙nat
      unfolding Finite˙def using eq˙lesspoll˙trans by auto
    moreover
    from assms(5) have nat≺Q∨nat=Q unfolding InfCard˙def
      using lt˙Card˙imp˙lesspoll[of Qnat] unfolding lt˙def succ˙def
      using Card˙is˙Ord[of Q] by auto
    ultimately have –Maximum(r,X)″≺Q using lesspoll˙trans by auto
    with assms(4,5) have C:A≺Q using less˙less˙imp˙un˙less
      by auto
     have WeakDense:A–is weakly dense in″X–with respect to″r using assms(2)
unfolding
      IsWeaklyDenseSub˙def by auto
    from MaxX assms(6) have S:A⊆X by auto
    then have Closure(A,OrdTopology X r)=X using Wdense˙ord˙imp˙dense˙top
      [OF assms(3) WeakDense ˙ assms(1)] nmin by auto
   then have thesis unfolding IsSeparableOfCard˙def using union˙ordtopology[OF
assms(3,1)]
      S C by auto
  }
  ultimately have thesis by auto
 }
 moreover
 {
  assume nmax:¬HasAmaximum(r,X)
  {
    assume HMin:HasAminimum(r,X)
   then have MinX:Minimum(r,X)∈X using Order˙ZF˙4˙L4(1) assms(3) unfolding
IsLinOrder˙def
```

```
      by auto
    let A=A ∪–Minimum(r,X)″
    have Finite(–Minimum(r,X)″) by auto
    then have –Minimum(r,X)″≺nat using n˙lesspoll˙nat
      unfolding Finite˙def using eq˙lesspoll˙trans by auto
    moreover
    from assms(5) have nat≺Q∨nat=Q unfolding InfCard˙def
      using lt˙Card˙imp˙lesspoll[of Qnat] unfolding lt˙def succ˙def
      using Card˙is˙Ord[of Q] by auto
    ultimately have –Minimum(r,X)″≺Q using lesspoll˙trans by auto
    with assms(4,5) have C:A≺Q using less˙less˙imp˙un˙less
      by auto
     have WeakDense:A–is weakly dense in″X–with respect to″r using assms(2)
unfolding
      IsWeaklyDenseSub˙def by auto
    from MinX assms(6) have S:A⊆X by auto
    then have Closure(A,OrdTopology X r)=X using Wdense˙ord˙imp˙dense˙top
      [OF assms(3) WeakDense ˙ assms(1)] nmax by auto
   then have thesis unfolding IsSeparableOfCard˙def using union˙ordtopology[OF
assms(3,1)]
      S C by auto
  }
  moreover
  {
    assume nmin:¬HasAminimum(r,X)
    let A=A
    from assms(4,5) have C:A≺Q by auto
     have WeakDense:A–is weakly dense in″X–with respect to″r using assms(2)
unfolding
      IsWeaklyDenseSub˙def by auto
    from assms(6) have S:A⊆X by auto
    then have Closure(A,OrdTopology X r)=X using Wdense˙ord˙imp˙dense˙top
      [OF assms(3) WeakDense ˙ assms(1)] nmin nmax by auto
   then have thesis unfolding IsSeparableOfCard˙def using union˙ordtopology[OF
assms(3,1)]
      S C by auto
  }
  ultimately have thesis by auto
 }
 ultimately show thesis by auto
qed


end
```

67 Properties in topology 2

```
theory Topology˙ZF˙properties˙2 imports Topology˙ZF˙7 Topology˙ZF˙1b
 Finite˙ZF˙1 Topology˙ZF˙11
```

begin

67.1 Local properties.

This theory file deals with local topological properties; and applies local compactness to the one point compactification.

We will say that a topological space is locally @term"P" iff every point has a neighbourhood basis of subsets that have the property @term"P" as subspaces.

definition
IsLocally (˙–is locally"˙ 90)
where T–is a topology" $\Longrightarrow$ T–is locally"P $\equiv$ ($\forall$ x$\in\bigcup$T. $\forall$ b$\in$T. x$\in$b $\longrightarrow$ ($\exists$ c$\in$Pow(b). x$\in$Interior(c,T) $\wedge$ P(c,T)))

67.2 First examples

Our first examples deal with the locally finite property. Finiteness is a property of sets, and hence it is preserved by homeomorphisms; which are in particular bijective.

The discrete topology is locally finite.

lemma discrete˙locally˙finite:
shows Pow(A)–is locally"(λA.(λB. Finite(A)))
proof-
have $\forall$ b$\in$Pow(A). $\bigcup$(Pow(A)–restricted to"b)=b unfolding RestrictedTo˙def by blast
then have $\forall$ b$\in$—x". x$\in$A". Finite(b) by auto moreover
have reg:$\forall$ S$\in$Pow(A). Interior(S,Pow(A))=S unfolding Interior˙def by auto
{
fix x b assume x$\in\bigcup$Pow(A) b$\in$Pow(A) x$\in$b
then have –x"$\subseteq$b x$\in$Interior(–x",Pow(A)) Finite(–x") using reg by auto
then have $\exists$ c$\in$Pow(b). x$\in$Interior(c,Pow(A))$\wedge$Finite(c) by blast
}
then have $\forall$ x$\in\bigcup$Pow(A). $\forall$ b$\in$Pow(A). x$\in$b $\longrightarrow$ ($\exists$ c$\in$Pow(b). x$\in$Interior(c,Pow(A)) $\wedge$ Finite(c)) by auto
then show thesis using IsLocally˙def[OF Pow˙is˙top] by auto
qed

The included set topology is locally finite when the set is finite.

lemma included˙finite˙locally˙finite:
assumes Finite(A) and A$\subseteq$X
shows (IncludedSet(X,A))–is locally"(λA.(λB. Finite(A)))
proof-
have $\forall$ b$\in$Pow(X). b$\cap$A$\subseteq$b by auto moreover
note assms(1)
ultimately have rr:$\forall$ b$\in$–A$\cup$–x". x$\in$X". Finite(b) by force

{
fix x b assume x∈⋃(IncludedSet(X,A)) b∈(IncludedSet(X,A)) x∈b
then have A∪–x″⊆b A∪–x″∈–A∪–x″. x∈X″ and sub: b⊆X unfolding IncludedSet˙def by auto
moreover have A ∪ –x″ ⊆ X using assms(2) sub ‹x∈b› by auto
then have x∈Interior(A∪–x″,IncludedSet(X,A)) using interior˙set˙includedset[of A∪–x″XA] by auto
ultimately have ∃c∈Pow(b). x∈Interior(c,IncludedSet(X,A))∧ Finite(c) using rr by blast
}
then have ∀x∈⋃(IncludedSet(X,A)). ∀b∈(IncludedSet(X,A)). x∈b ⟶ (∃c∈Pow(b). x∈Interior(c,IncludedSet(X,A))∧ Finite(c)) by auto
then show thesis using IsLocally˙def includedset˙is˙topology by auto
qed

67.3 Local compactness

definition
IsLocallyComp (˙–is locally-compact″ 70)
where T–is locally-compact″≡T–is locally″(λB. λT. B–is compact in″T)

We center ourselves in local compactness, because it is a very important tool in topological groups and compactifications.

If a subset is compact of some cardinal for a topological space, it is compact of the same cardinal in the subspace topology.

lemma compact˙imp˙compact˙subspace:
assumes A–is compact of cardinal″K–in″T A⊆B
shows A–is compact of cardinal″K–in″(T–restricted to″B) unfolding IsCompactOfCard˙def
proof
from assms show C:Card(K) unfolding IsCompactOfCard˙def by auto
from assms have A⊆⋃T unfolding IsCompactOfCard˙def by auto
then have AA:A⊆⋃(T–restricted to″B) using assms(2) unfolding RestrictedTo˙def by auto moreover
{
fix M assume M∈Pow(T–restricted to″B) A⊆⋃M
let M=–S∈T. B∩S∈M″
from ‹M∈Pow(T–restricted to″B)› have ⋃M⊆⋃M unfolding RestrictedTo˙def by auto
with ‹A⊆⋃M› have A⊆⋃MM∈Pow(T) by auto
with assms have ∃N∈Pow(M). A⊆⋃N∧N≺K unfolding IsCompactOfCard˙def by auto
then obtain N where N∈Pow(M) A⊆⋃N N≺K by auto
then have N–restricted to″B⊆M unfolding RestrictedTo˙def FinPow˙def by auto
moreover
let f=–⟨𝔅,B∩𝔅⟩. 𝔅∈N″
have f:N→(N–restricted to″B) unfolding Pi˙def function˙def domain˙def RestrictedTo˙def by auto

then have f∈surj(N,N–restricted to″B) unfolding surj˙def RestrictedTo˙def using apply˙equality
by auto
from ‹N≺K› have N≲K unfolding lesspoll˙def by auto
with ‹f∈surj(N,N–restricted to″B)› have N–restricted to″B≲N using surj˙fun˙inv˙2 Card˙is˙Ord C by auto
with ‹N≺K› have N–restricted to″B≺K using lesspoll˙trans1 by auto
moreover from ‹A⊆⋃N› have A⊆⋃(N–restricted to″B) using assms(2) unfolding RestrictedTo˙def by auto
ultimately have ∃N∈Pow(M). A⊆⋃N ∧ N≺K by auto
}
with AA show A ⊆ ⋃(T –restricted to″ B) ∧ (∀M∈Pow(T –restricted to″ B). A ⊆ ⋃M ⟶ (∃N∈Pow(M). A ⊆ ⋃N ∧ N≺K)) by auto
qed

The converse of the previous result is not always true. For compactness, it holds because the axiom of finite choice always holds.

lemma compact˙subspace˙imp˙compact:
assumes A–is compact in″(T–restricted to″B) A⊆B
shows A–is compact in″T unfolding IsCompact˙def
proof
from assms show A⊆⋃T unfolding IsCompact˙def RestrictedTo˙def by auto
next
{
fix M assume M∈Pow(T) A⊆⋃M
let M=M–restricted to″B
from ‹M∈Pow(T)› have M∈Pow(T–restricted to″B) unfolding RestrictedTo˙def by auto
from ‹A⊆⋃M› have A⊆⋃M unfolding RestrictedTo˙def using assms(2) by auto
with assms ‹M∈Pow(T–restricted to″B)› obtain N where N∈FinPow(M) A⊆⋃N unfolding IsCompact˙def by blast
from ‹N∈FinPow(M)› have N≺nat unfolding FinPow˙def Finite˙def using n˙lesspoll˙nat eq˙lesspoll˙trans
by auto
then have Finite(N) using lesspoll˙nat˙is˙Finite by auto
then obtain n where n∈nat N≈n unfolding Finite˙def by auto
then have N≲n using eqpoll˙imp˙lepoll by auto
moreover
{
fix BB assume BB∈N
with ‹N∈FinPow(M)› have BB∈M unfolding FinPow˙def by auto
then obtain S where S∈M and BB=B∩S unfolding RestrictedTo˙def by auto
then have S∈–S∈M. B∩S=BB″ by auto
then obtain –S∈M. B∩S=BB″≠0 by auto
}
then have ∀BB∈N. ((λW∈N. –S∈M. B∩S=W″)BB)≠0 by auto moreover
from ‹n∈nat› have (N ≲ n ∧ (∀t∈N. (λW∈N. –S∈M. B∩S=W″) t ≠ 0) ⟶ (∃f. f ∈ Pi(N,λt. (λW∈N. –S∈M. B∩S=W″) t) ∧ (∀t∈N. f t ∈ (λW∈N. –S∈M. B∩S=W″) t))) using finite˙choice unfolding AxiomCardinalChoiceGen˙def by blast

ultimately
obtain f where AA:f∈Pi(N,λt. (λW∈N. –S∈M. B∩S=W″) t) ∀t∈N. ft∈(λW∈N. –S∈M. B∩S=W″) t by blast
from AA(2) have ss:∀t∈N. ft∈–S∈M. B∩S=t″ using beta˙if by auto
then have –ft. t∈N″⊆M by auto
{
fix t assume t∈N
with ss have ft∈–S∈M. B∩S∈N″ by auto
}
with AA(1) have FF:f:N→–S∈M. B∩S∈N″ unfolding Pi˙def Sigma˙def using beta˙if by auto moreover
{
fix aa bb assume AAA:aa∈N bb∈N faa=fbb
from AAA(1) ss have B∩ (faa) =aa by auto
with AAA(3) have B∩(fbb)=aa by auto
with ss AAA(2) have aa=bb by auto
}
ultimately have f∈inj(N,–S∈M. B∩S∈N″) unfolding inj˙def by auto
then have f∈bij(N,range(f)) using inj˙bij˙range by auto
then have f∈bij(N,fN) using range˙image˙domain FF by auto
then have f∈bij(N,–ft. t∈N″) using func˙imagedef FF by auto
then have N≈–ft. t∈N″ unfolding eqpoll˙def by auto
with ‹N≈n› have –ft. t∈N″≈n using eqpoll˙sym eqpoll˙trans by blast
with ‹n∈nat› have Finite(–ft. t∈N″) unfolding Finite˙def by auto
with ss have –ft. t∈N″∈FinPow(M) unfolding FinPow˙def by auto moreover
{
fix aa assume aa∈A
with ‹A⊆⋃N› obtain b where b∈N and aa∈b by auto
with ss have B∩(fb)=b by auto
with ‹aa∈b› have aa∈B∩(fb) by auto
then have aa∈ fb by auto
with ‹b∈N› have aa∈⋃–ft. t∈N″ by auto
}
then have A⊆⋃–ft. t∈N″ by auto ultimately
have ∃R∈FinPow(M). A⊆⋃R by auto
}
then show ∀M∈Pow(T). A ⊆ ⋃M ⟶ (∃N∈FinPow(M). A ⊆ ⋃N) by auto
qed

If the axiom of choice holds for some cardinal, then we can drop the compact sets of that cardial are compact of the same cardinal as subspaces of every superspace.

lemma Kcompact˙subspace˙imp˙Kcompact:
assumes A–is compact of cardinal″Q–in″(T–restricted to″B) A⊆B (–the axiom of″ Q –choice holds″)
shows A–is compact of cardinal″Q–in″T
proof -
from assms(1) have a1:Card(Q) unfolding IsCompactOfCard˙def RestrictedTo˙def by auto

from assms(1) have a2:A⊆⋃T unfolding IsCompactOfCard˙def RestrictedTo˙def by auto
{
fix M assume M∈Pow(T) A⊆⋃M
let M=M–restricted to″B
from ‹M∈Pow(T)› have M∈Pow(T–restricted to″B) unfolding RestrictedTo˙def by auto
from ‹A⊆⋃M› have A⊆⋃M unfolding RestrictedTo˙def using assms(2) by auto
with assms ‹M∈Pow(T–restricted to″B)› obtain N where N:N∈Pow(M) A⊆⋃N N ≺ Q unfolding IsCompactOfCard˙def by blast
from N(3) have N≲Q using lesspoll˙imp˙lepoll by auto moreover
{
fix BB assume BB∈N
with ‹N∈Pow(M)› have BB∈M unfolding FinPow˙def by auto
then obtain S where S∈M and BB=B∩S unfolding RestrictedTo˙def by auto
then have S∈–S∈M. B∩S=BB″ by auto
then obtain –S∈M. B∩S=BB″≠0 by auto
}
then have ∀BB∈N. ((λW∈N. –S∈M. B∩S=W″)BB)≠0 by auto moreover
have (N ≲ Q ∧ (∀t∈N. (λW∈N. –S∈M. B∩S=W″) t ≠ 0) ⟶ (∃f. f ∈ Pi(N,λt. (λW∈N. –S∈M. B∩S=W″) t) ∧ (∀t∈N. f t ∈ (λW∈N. –S∈M. B∩S=W″) t)))
using assms(3) unfolding AxiomCardinalChoiceGen˙def by blast
ultimately
obtain f where AA:f∈Pi(N,λt. (λW∈N. –S∈M. B∩S=W″) t) ∀t∈N. ft∈(λW∈N. –S∈M. B∩S=W″) t by blast
from AA(2) have ss:∀t∈N. ft∈–S∈M. B∩S=t″ using beta˙if by auto
then have –ft. t∈N″⊆M by auto
{
fix t assume t∈N
with ss have ft∈–S∈M. B∩S∈N″ by auto
}
with AA(1) have FF:f:N→–S∈M. B∩S∈N″ unfolding Pi˙def Sigma˙def using beta˙if by auto moreover
{
fix aa bb assume AAA:aa∈N bb∈N faa=fbb
from AAA(1) ss have B∩ (faa) =aa by auto
with AAA(3) have B∩(fbb)=aa by auto
with ss AAA(2) have aa=bb by auto
}
ultimately have f∈inj(N,–S∈M. B∩S∈N″) unfolding inj˙def by auto
then have f∈bij(N,range(f)) using inj˙bij˙range by auto
then have f∈bij(N,fN) using range˙image˙domain FF by auto
then have f∈bij(N,–ft. t∈N″) using func˙imagedef FF by auto
then have N≈–ft. t∈N″ unfolding eqpoll˙def by auto
with ‹N≺Q› have –ft. t∈N″≺Q using eqpoll˙sym eq˙lesspoll˙trans by blast moreover
with ss have –ft. t∈N″∈Pow(M) unfolding FinPow˙def by auto moreover
{
fix aa assume aa∈A

```
    with ‹A⊆⋃N› obtain b where b∈N and aa∈b by auto
    with ss have B∩(fb)=b by auto
    with ‹aa∈b› have aa∈B∩(fb) by auto
    then have aa∈ fb by auto
    with ‹b∈N› have aa∈⋃–ft. t∈N″ by auto
  }
  then have A⊆⋃–ft. t∈N″ by auto ultimately
  have ∃R∈Pow(M). A⊆⋃R ∧ R≺Q by auto
 }
 then show thesis using a1 a2 unfolding IsCompactOfCard˙def by auto
qed
```

Every set, with the cofinite topology is compact.

```
lemma cofinite˙compact:
 shows X –is compact in″(CoFinite X) unfolding IsCompact˙def
proof
 show X⊆⋃(CoFinite X) using union˙cocardinal unfolding Cofinite˙def by auto
next
 {
  fix M assume M∈Pow(CoFinite X) X⊆⋃M
  {
   assume M=0∨M=–0″
   then have M∈FinPow(M) unfolding FinPow˙def by auto
   with ‹X⊆⋃M› have ∃N∈FinPow(M). X⊆⋃N by auto
  }
  moreover
  {
   assume M≠0M≠–0″
   then obtain U where U∈MU≠0 by auto
   with ‹M∈Pow(CoFinite X)› have U∈CoFinite X by auto
    with ‹U≠0› have U⊆X (X-U)≺nat unfolding Cofinite˙def CoCardinal˙def by
auto
   then have Finite(X-U) using lesspoll˙nat˙is˙Finite by auto
    then have (X-U)–is in the spectrum of″(λT. (⋃T)–is compact in″T) using
compact˙spectrum
    by auto
   then have ((⋃(CoFinite (X-U)))≈X-U) ⟶ ((⋃(CoFinite (X-U)))–is compact
in″(CoFinite (X-U))) unfolding Spec˙def
    using InfCard˙nat CoCar˙is˙topology unfolding Cofinite˙def by auto
   then have com:(X-U)–is compact in″(CoFinite (X-U)) using union˙cocardinal
unfolding Cofinite˙def by auto
   have (X-U)∩X=X-U by auto
    then have (CoFinite X)–restricted to″(X-U)=(CoFinite (X-U)) using sub-
space˙cocardinal unfolding Cofinite˙def by auto
  with com have (X-U)–is compact in″(CoFinite X) using compact˙subspace˙imp˙compact[of
X-UCoFinite XX-U] by auto
   moreover have X-U⊆⋃M using ‹X⊆⋃M› by auto
   moreover note ‹M∈Pow(CoFinite X)›
   ultimately have ∃N∈FinPow(M). X-U⊆⋃N unfolding IsCompact˙def by auto
```

then obtain N where N⊆M Finite(N) X-U⊆⋃N unfolding FinPow˙def by auto
with ‹U∈M› have N ∪–U″⊆M Finite(N ∪–U″) X⊆⋃(N ∪–U″) by auto
then have ∃N∈FinPow(M). X⊆⋃N unfolding FinPow˙def by blast
}
ultimately
have ∃N∈FinPow(M). X⊆⋃N by auto
}
then show ∀M∈Pow(CoFinite X). X ⊆ ⋃M ⟶ (∃N∈FinPow(M). X ⊆ ⋃N) by auto
qed

A corollary is then that the cofinite topology is locally compact; since every subspace of a cofinite space is cofinite.

corollary cofinite˙locally˙compact:
shows (CoFinite X)–is locally-compact″
proof-
have cof:topology0(CoFinite X) and cof1:(CoFinite X)–is a topology″
using CoCar˙is˙topology InfCard˙nat Cofinite˙def unfolding topology0˙def by auto
{
fix x B assume x∈⋃(CoFinite X) B∈(CoFinite X) x∈B
then have x∈Interior(B,CoFinite X) using topology0.Top˙2˙L3[OF cof] by auto moreover
from ‹B∈(CoFinite X)› have B⊆X unfolding Cofinite˙def CoCardinal˙def by auto
then have B∩X=B by auto
then have (CoFinite X)–restricted to″B=CoFinite B using subspace˙cocardinal unfolding Cofinite˙def by auto
then have B–is compact in″((CoFinite X)–restricted to″B) using cofinite˙compact union˙cocardinal unfolding Cofinite˙def by auto
then have B–is compact in″(CoFinite X) using compact˙subspace˙imp˙compact by auto
ultimately have ∃c∈Pow(B). x∈Interior(c,CoFinite X)∧ c–is compact in″(CoFinite X) by auto
}
then have (∀x∈⋃(CoFinite X). ∀b∈(CoFinite X). x∈b ⟶ (∃c∈Pow(b). x∈Interior(c,CoFinite X) ∧ c–is compact in″(CoFinite X)))
by auto
then show thesis unfolding IsLocallyComp˙def IsLocally˙def[OF cof1] by auto
qed

In every locally compact space, by definition, every point has a compact neighbourhood.

theorem (in topology0) locally˙compact˙exist˙compact˙neig:
assumes T–is locally-compact″
shows ∀x∈⋃T. ∃A∈Pow(⋃T). A–is compact in″T ∧ x∈int(A)
proof-
{
fix x assume x∈⋃T moreover
then have ⋃T≠0 by auto

```
    have ⋃T∈T using union˙open topSpaceAssum by auto
    ultimately have ∃c∈Pow(⋃T). x∈int(c)∧ c–is compact in″T using assms
      IsLocally˙def topSpaceAssum unfolding IsLocallyComp˙def by auto
    then have ∃c∈Pow(⋃T). c–is compact in″T ∧ x∈int(c) by auto
  }
  then show thesis by auto
qed
```

In Hausdorff spaces, the previous result is an equivalence.

```
theorem (in topology0) exist˙compact˙neig˙T2˙imp˙locally˙compact:
  assumes ∀x∈⋃T. ∃A∈Pow(⋃T). x∈int(A) ∧ A–is compact in″T T–is T2″
  shows T–is locally-compact″
proof-
  {
    fix x assume x∈⋃T
    with assms(1) obtain A where A∈Pow(⋃T) x∈int(A) and Acom:A–is compact
in″T by blast
    then have Acl:A–is closed in″T using in˙t2˙compact˙is˙cl assms(2) by auto
    then have sub:A⊆⋃T unfolding IsClosed˙def by auto
    {
      fix U assume U∈T x∈U
      let V=int(A∩U)
      from ‹x∈U› ‹x∈int(A)› have x∈U∩(int (A)) by auto
        moreover from ‹U∈T› have U∩(int(A))∈T using Top˙2˙L2 topSpaceAssum
unfolding IsATopology˙def
        by auto moreover
      have U∩(int(A))⊆A∩U using Top˙2˙L1 by auto
      ultimately have x∈V using Top˙2˙L5 by blast
      have V⊆A using Top˙2˙L1 by auto
      then have cl(V)⊆A using Acl Top˙3˙L13 by auto
      then have A∩cl(V)=cl(V) by auto moreover
      have clcl:cl(V)–is closed in″T using cl˙is˙closed ‹V⊆A› ‹A⊆⋃T› by auto
       ultimately have comp:cl(V)–is compact in″T using Acom compact˙closed[of
AnatTcl(V)] Compact˙is˙card˙nat
        by auto
      {
     then have cl(V)–is compact in″(T–restricted to″cl(V)) using compact˙imp˙compact˙subspace[of
cl(V)natT] Compact˙is˙card˙nat
         by auto moreover
        have ⋃(T–restricted to″cl(V))=cl(V) unfolding RestrictedTo˙def using clcl
unfolding IsClosed˙def by auto moreover
          ultimately have (⋃(T–restricted to″cl(V)))–is compact in″(T–restricted
to″cl(V)) by auto
      }
       then have (⋃(T–restricted to″cl(V)))–is compact in″(T–restricted to″cl(V))
by auto moreover
        have (T–restricted to″cl(V))–is T2″ using assms(2) T2˙here clcl unfolding
IsClosed˙def by auto
    ultimately have (T–restricted to″cl(V))–is T4″ using topology0.T2˙compact˙is˙normal
```

unfolding topology0˙def
using Top˙1˙L4 unfolding isT4˙def using T2˙is˙T1 by auto
then have clvreg:(T–restricted to″cl(V))–is regular″ using topology0.T4˙is˙T3 unfolding topology0˙def isT3˙def using Top˙1˙L4
by auto
have V⊆cl(V) using cl˙contains˙set ‹V⊆A› ‹A⊆⋃T› by auto
then have V∈(T–restricted to″cl(V)) unfolding RestrictedTo˙def using Top˙2˙L2 by auto
with ‹x∈V› obtain W where Wop:W∈(T–restricted to″cl(V)) and clcont:Closure(W,(T–restricted to″cl(V)))⊆V and cinW:x∈W
using topology0.regular˙imp˙exist˙clos˙neig unfolding topology0˙def using Top˙1˙L4 clvreg
by blast
from clcont Wop have W⊆V using topology0.cl˙contains˙set unfolding topology0˙def using Top˙1˙L4 by auto
with Wop have W∈(T–restricted to″cl(V))–restricted to″V unfolding RestrictedTo˙def by auto
moreover from ‹V⊆A› ‹A⊆⋃T› have V⊆⋃T by auto
then have V⊆cl(V)cl(V)⊆⋃T using ‹V⊆cl(V)› Top˙3˙L11(1) by auto
then have (T–restricted to″cl(V))–restricted to″V=(T–restricted to″V) using subspace˙of˙subspace by auto
ultimately have W∈(T–restricted to″V) by auto
then obtain UU where UU∈T W=UU∩V unfolding RestrictedTo˙def by auto
then have W∈T using Top˙2˙L2 topSpaceAssum unfolding IsATopology˙def by auto moreover
have W⊆Closure(W,(T–restricted to″cl(V))) using topology0.cl˙contains˙set unfolding topology0˙def
using Top˙1˙L4 Wop by auto
ultimately have A1:x∈int(Closure(W,(T–restricted to″cl(V)))) using Top˙2˙L6 cinW by auto
from clcont have A2:Closure(W,(T–restricted to″cl(V)))⊆U using Top˙2˙L1 by auto
have clwcl:Closure(W,(T–restricted to″cl(V))) –is closed in″(T–restricted to″cl(V))
using topology0.cl˙is˙closed Top˙1˙L4 Wop unfolding topology0˙def by auto
from comp have cl(V)–is compact in″(T–restricted to″cl(V)) using compact˙imp˙compact˙subspace[of cl(V)natT] Compact˙is˙card˙nat
by auto
with clwcl have ((cl(V)∩(Closure(W,(T–restricted to″cl(V))))))–is compact in″(T–restricted to″cl(V))
using compact˙closed Compact˙is˙card˙nat by auto moreover
from clcont have cont:(Closure(W,(T–restricted to″cl(V))))⊆cl(V) using cl˙contains˙set ‹V⊆A›‹A⊆⋃T›
by blast
then have ((cl(V)∩(Closure(W,(T–restricted to″cl(V))))))=Closure(W,(T–restricted to″cl(V))) by auto
ultimately have Closure(W,(T–restricted to″cl(V)))–is compact in″(T–restricted to″cl(V)) by auto
then have Closure(W,(T–restricted to″cl(V)))–is compact in″T using com-

pact˙subspace˙imp˙compact[of Closure(W,T–restricted to″cl(V))]
cont by auto
with A1 A2 have ∃c∈Pow(U). x∈int(c)∧c–is compact in″T by auto
}
then have ∀U∈T. x∈U ⟶ (∃c∈Pow(U). x∈int(c)∧c–is compact in″T) by auto
}
then show thesis unfolding IsLocally˙def[OF topSpaceAssum] IsLocallyComp˙def by auto
qed

67.4 Compactification by one point

Given a topological space, we can always add one point to the space and get a new compact topology; as we will check in this section.

definition
OPCompactification (–one-point compactification of″˙ 90)
where –one-point compactification of″T≡T∪—⋃T″∪((⋃T)-K). K∈–B∈Pow(⋃T). B–is compact in″T ∧ B–is closed in″T″″

Firstly, we check that what we defined is indeed a topology.

theorem (in topology0) op˙comp˙is˙top:
shows (–one-point compactification of″T)–is a topology″ unfolding IsATopology˙def
proof(safe)
fix M assume M⊆–one-point compactification of″T
then have disj:M⊆T∪—⋃T″∪((⋃T)-K). K∈–B∈Pow(⋃T). B–is compact in″T ∧ B–is closed in″T″″ unfolding OPCompactification˙def by auto
let MT=–A∈M. A∈T″
have MT⊆T by auto
then have c1:⋃MT∈T using topSpaceAssum unfolding IsATopology˙def by auto
let MK=–A∈M. A∉T″
have ⋃M=⋃MK ∪ ⋃MT by auto
from disj have MK⊆–A∈M. A∈—⋃T″∪((⋃T)-K). K∈–B∈Pow(⋃T). B–is compact in″T ∧ B–is closed in″T″″″ by auto
moreover have N:⋃T∉(⋃T) using mem˙not˙refl by auto
{
fix B assume B∈M B∈—⋃T″∪((⋃T)-K). K∈–B∈Pow(⋃T). B–is compact in″T ∧ B–is closed in″T″″
then obtain K where K∈Pow(⋃T) B=–⋃T″∪((⋃T)-K) by auto
with N have ⋃T∈B by auto
with N have B∉T by auto
with ‹B∈M› have B∈MK by auto
}
then have –A∈M. A∈—⋃T″∪((⋃T)-K). K∈–B∈Pow(⋃T). B–is compact in″T ∧ B–is closed in″T″″″⊆MK by auto
ultimately have MK˙def:MK=–A∈M. A∈—⋃T″∪((⋃T)-K). K∈–B∈Pow(⋃T). B–is compact in″T ∧ B–is closed in″T″″″ by auto
let KK=–K∈Pow(⋃T). –⋃T″∪((⋃T)-K)∈MK″

{
assume MK=0
then have ⋃M=⋃MT by auto
then have ⋃M∈T using c1 by auto
then have ⋃M∈–one-point compactification of″T unfolding OPCompactification˙def by auto
}
moreover
{
assume MK≠0
then obtain A where A∈MK by auto
then obtain K1 where A=–⋃T″∪((⋃T)-K1) K1∈Pow(⋃T) K1–is closed in″T K1–is compact in″T using MK˙def by auto
with ‹A∈MK› have ⋂KK⊆K1 by auto
from ‹A∈MK› ‹A=–⋃T″∪((⋃T)-K1)› ‹K1∈Pow(⋃T)› have KK≠0 by blast
{
fix K assume K∈KK
then have –⋃T″∪((⋃T)-K)∈MK K⊆⋃T by auto
then obtain KK where A:–⋃T″∪((⋃T)-K)=–⋃T″∪((⋃T)-KK) KK⊆⋃T KK–is compact in″T KK–is closed in″T using MK˙def by auto
note A(1) moreover
have (⋃T)-K⊆–⋃T″∪((⋃T)-K) (⋃T)-KK⊆–⋃T″∪((⋃T)-KK) by auto
ultimately have (⋃T)-K⊆–⋃T″∪((⋃T)-KK) (⋃T)-KK⊆–⋃T″∪((⋃T)-K) by auto moreover
from N have ⋃T∉(⋃T)-K ⋃T∉(⋃T)-KK by auto ultimately
have (⋃T)-K⊆((⋃T)-KK) (⋃T)-KK⊆((⋃T)-K) by auto
then have (⋃T)-K=(⋃T)-KK by auto moreover
from ‹K⊆⋃T› have K=(⋃T)-((⋃T)-K) by auto ultimately
have K=(⋃T)-((⋃T)-KK) by auto
with ‹KK⊆⋃T› have K=KK by auto
with A(4) have K–is closed in″T by auto
}
then have ∀K∈KK. K–is closed in″T by auto
with ‹KK≠0› have (⋂KK)–is closed in″T using Top˙3˙L4 by auto
with ‹K1–is compact in″T› have (K1∩(⋂KK))–is compact in″T using Compact˙is˙card˙nat
compact˙closed[of K1natT⋂KK] by auto moreover
from ‹⋂KK⊆K1› have K1∩(⋂KK)=(⋂KK) by auto ultimately
have (⋂KK)–is compact in″T by auto
with ‹(⋂KK)–is closed in″T› ‹⋂KK⊆K1› ‹K1∈Pow(⋃T)› have (–⋃T″∪((⋃T)-(⋂KK)))∈(–one-point compactification of″T)
unfolding OPCompactification˙def by blast
have t:⋃MK=⋃–A∈M. A∈—⋃T″∪((⋃T)-K). K∈–B∈Pow(⋃T). B–is compact in″T ∧ B–is closed in″T″″″
using MK˙def by auto
{
fix x assume x∈⋃MK
with t have x∈⋃–A∈M. A∈—⋃T″∪((⋃T)-K). K∈–B∈Pow(⋃T). B–is compact in″T ∧ B–is closed in″T″″″ by auto

then have ∃AA∈–A∈M. A∈—⋃T″∪((⋃T)-K). K∈–B∈Pow(⋃T). B–is compact in″T ∧ B–is closed in″T″″″. x∈AA
using Union˙iff by auto
then obtain AA where AAp:AA∈–A∈M. A∈—⋃T″∪((⋃T)-K). K∈–B∈Pow(⋃T). B–is compact in″T ∧ B–is closed in″T″″″ x∈AA by auto
then obtain K2 where AA=–⋃T″∪((⋃T)-K2) K2∈Pow(⋃T)K2–is compact in″T K2–is closed in″T by auto
with ‹x∈AA› have x=⋃T ∨ (x∈(⋃T) ∧ x∉K2) by auto
from ‹K2∈Pow(⋃T)› ‹AA=–⋃T″∪((⋃T)-K2)› AAp(1) MK˙def have K2∈KK by auto
then have ⋂KK⊆K2 by auto
with ‹x=⋃T ∨ (x∈(⋃T) ∧ x∉K2)› have x=⋃T∨(x∈⋃T ∧ x∉⋂KK) by auto
then have x∈–⋃T″∪((⋃T)-(⋂KK)) by auto
}
then have ⋃MK⊆–⋃T″∪((⋃T)-(⋂KK)) by auto
moreover
{
fix x assume x∈–⋃T″∪((⋃T)-(⋂KK))
then have x=⋃T∨(x∈(⋃T)∧ x∉⋂KK) by auto
with ‹KK≠0› obtain K2 where K2∈KK x=⋃T∨(x∈⋃T∧ x∉K2) by auto
then have –⋃T″∪((⋃T)-K2)∈MK by auto
with ‹x=⋃T∨(x∈⋃T∧ x∉K2)› have x∈⋃MK by auto
}
then have –⋃T″∪((⋃T)-(⋂KK))⊆⋃MK by (safe,auto)
ultimately have ⋃MK=–⋃T″∪((⋃T)-(⋂KK)) by blast
from ‹⋃MT∈T› have ⋃T-(⋃T-⋃MT)=⋃MT by auto
with ‹⋃MT∈T› have (⋃T-⋃MT)–is closed in″T unfolding IsClosed˙def by auto
have ((⋃T)-(⋂KK))∪(⋃T-(⋃T-⋃MT))=(⋃T)-((⋂KK)∩(⋃T-⋃MT)) by auto
then have (–⋃T″∪((⋃T)-(⋂KK)))∪(⋃T-(⋃T-⋃MT))=–⋃T″∪((⋃T)-((⋂KK)∩(⋃T-⋃MT))) by auto
with ‹⋃MK=–⋃T″∪((⋃T)-(⋂KK))›‹⋃T-(⋃T-⋃MT)=⋃MT› have ⋃MK∪⋃MT=–⋃T″∪((⋃T)-((⋂KK
by auto
with ‹⋃M=⋃MK ∪⋃MT› have unM:⋃M=–⋃T″∪((⋃T)-((⋂KK)∩(⋃T-⋃MT))) by auto
have ((⋂KK)∩(⋃T-⋃MT)) –is closed in″T using ‹(⋂KK)–is closed in″T›‹(⋃T-(⋃MT))–is closed in″T›
Top˙3˙L5 by auto
moreover
note ‹(⋃T-(⋃MT))–is closed in″T› ‹(⋂KK)–is compact in″T›
then have ((⋂KK)∩(⋃T-⋃MT))–is compact of cardinal″nat–in″T using compact˙closed[of ⋂KKnatT(⋃T-⋃MT)] Compact˙is˙card˙nat
by auto
then have ((⋂KK)∩(⋃T-⋃MT))–is compact in″T using Compact˙is˙card˙nat by auto
ultimately have –⋃T″∪(⋃T-((⋂KK)∩(⋃T-⋃MT)))∈–one-point compactification of″T
unfolding OPCompactification˙def IsClosed˙def by auto
with unM have ⋃M∈–one-point compactification of″T by auto

}
ultimately show ⋃M∈–one-point compactification of″T by auto
next
fix U V assume U∈–one-point compactification of″T and V∈–one-point compactification of″T
then have A:U∈T∨(∃KU∈Pow(⋃T). U=–⋃T″∪(⋃T-KU)∧KU–is closed in″T∧KU–is compact in″T)
V∈T∨(∃KV∈Pow(⋃T). V=–⋃T″∪(⋃T-KV)∧KV–is closed in″T∧KV–is compact in″T) unfolding OPCompactification˙def
by auto
have N:⋃T∉(⋃T) using mem˙not˙refl by auto
{
assume U∈TV∈T
then have U∩V∈T using topSpaceAssum unfolding IsATopology˙def by auto
then have U∩V∈–one-point compactification of″T unfolding OPCompactification˙def
by auto
}
moreover
{
assume U∈TV∉T
then obtain KV where V:KV–is closed in″TKV–is compact in″TV=–⋃T″∪(⋃T-KV)
using A(2) by auto
with N ‹U∈T› have ⋃T∉U by auto
then have ⋃T∉U∩V by auto
then have U∩V=U∩(⋃T-KV) using V(3) by auto
moreover have ⋃T-KV∈T using V(1) unfolding IsClosed˙def by auto
with ‹U∈T› have U∩(⋃T-KV)∈T using topSpaceAssum unfolding IsATopology˙def by auto
with ‹U∩V=U∩(⋃T-KV)› have U∩V∈T by auto
then have U∩V∈–one-point compactification of″T unfolding OPCompactification˙def by auto
}
moreover
{
assume U∉TV∈T
then obtain KV where V:KV–is closed in″TKV–is compact in″TU=–⋃T″∪(⋃T-KV)
using A(1) by auto
with N ‹V∈T› have ⋃T∉V by auto
then have ⋃T∉U∩V by auto
then have U∩V=(⋃T-KV)∩V using V(3) by auto
moreover have ⋃T-KV∈T using V(1) unfolding IsClosed˙def by auto
with ‹V∈T› have (⋃T-KV)∩V∈T using topSpaceAssum unfolding IsATopology˙def by auto
with ‹U∩V=(⋃T-KV)∩V› have U∩V∈T by auto
then have U∩V∈–one-point compactification of″T unfolding OPCompactification˙def by auto
}
moreover

```
  {
    assume U∉TV∉T
   then obtain KV KU where V:KV–is closed in″TKV–is compact in″TV=–⋃T″∪(⋃T-KV)
     and U:KU–is closed in″TKU–is compact in″TU=–⋃T″∪(⋃T-KU)
    using A by auto
    with V(3) U(3) have ⋃T∈U∩V by auto
    then have U∩V=–⋃T″∪((⋃T-KV)∩(⋃T-KU)) using V(3) U(3) by auto
     moreover have ⋃T-KV∈T⋃T-KU∈T using V(1) U(1) unfolding IsClosed˙def
by auto
     then have (⋃T-KV)∩(⋃T-KU)∈T using topSpaceAssum unfolding IsATopol-
ogy˙def by auto
      then have (⋃T-KV)∩(⋃T-KU)=⋃T-(⋃T-((⋃T-KV)∩(⋃T-KU))) by auto
moreover
    with ‹(⋃T-KV)∩(⋃T-KU)∈T› have (⋃T-(⋃T-KV)∩(⋃T-KU))–is closed in″T
unfolding IsClosed˙def
      by auto moreover
   from V(1) U(1) have (⋃T-(⋃T-KV)∩(⋃T-KU))=KV∪KU unfolding IsClosed˙def
by auto
   with V(2) U(2) have (⋃T-(⋃T-KV)∩(⋃T-KU))–is compact in″T using union˙compact[of
KVnatTKU] Compact˙is˙card˙nat
      InfCard˙nat by auto ultimately
    have U∩V∈–one-point compactification of″T unfolding OPCompactification˙def
by auto
  }
  ultimately show U∩V∈–one-point compactification of″T by auto
qed
```

The original topology is an open subspace of the new topology.

```
theorem (in topology0) open˙subspace:
   shows ⋃T∈–one-point compactification of″T and (–one-point compactification
of″T)–restricted to″⋃T=T
proof-
  show ⋃T∈–one-point compactification of″T
 unfolding OPCompactification˙def using topSpaceAssum unfolding IsATopology˙def
by auto
  have T⊆(–one-point compactification of″T)–restricted to″⋃T unfolding OPCom-
pactification˙def RestrictedTo˙def by auto
  moreover
  {
    fix A assume A∈(–one-point compactification of″T)–restricted to″⋃T
   then obtain R where R∈(–one-point compactification of″T) A=⋃T∩R unfolding
RestrictedTo˙def by auto
   then obtain K where K:R∈T ∨ (R=–⋃T″∪(⋃T-K) ∧ K–is closed in″T) unfolding
OPCompactification˙def by auto
      with ‹A=⋃T∩R› have (A=R∧R∈T)∨(A=⋃T-K ∧ K–is closed in″T) using
mem˙not˙refl unfolding IsClosed˙def by auto
    with K have A∈T unfolding IsClosed˙def by auto
  }
  ultimately
```

show (–one-point compactification of″T)–restricted to″⋃T=T by auto
qed

We added only one new point to the space.

lemma (in topology0) op˙compact˙total:
shows ⋃(–one-point compactification of″T)=–⋃T″∪(⋃T)
proof-
have 0–is compact in″T unfolding IsCompact˙def FinPow˙def by auto
moreover note Top˙3˙L2 ultimately have TT:0∈–A∈Pow(⋃T). A–is compact in″T ∧A–is closed in″T″ by auto
have ⋃(–one-point compactification of″T)=(⋃T)∪(⋃—⋃T″∪(⋃T-K). K∈–B∈Pow(⋃T). B–is compact in″T∧B–is closed in″T″″) unfolding OPCompactification˙def
by blast
also have . . .=(⋃T)∪–⋃T″∪(⋃–(⋃T-K). K∈–B∈Pow(⋃T). B–is compact in″T∧B–is closed in″T″″) using TT by auto
ultimately show ⋃(–one-point compactification of″T)=–⋃T″∪(⋃T) by auto
qed

The one point compactification, gives indeed a compact topological space.

theorem (in topology0) compact˙op:
shows (–⋃T″∪(⋃T))–is compact in″(–one-point compactification of″T) unfolding IsCompact˙def
proof(safe)
have 0–is compact in″T unfolding IsCompact˙def FinPow˙def by auto
moreover note Top˙3˙L2 ultimately have 0∈–A∈Pow(⋃T). A–is compact in″T ∧A–is closed in″T″ by auto
then have –⋃T″∪(⋃T)∈–one-point compactification of″T unfolding OPCompactification˙def by auto
then show ⋃T ∈ ⋃–one-point compactification of″T by auto
next
fix x B assume x∈BB∈T
then show x∈⋃(–one-point compactification of″T) using open˙subspace by auto
next
fix M assume A:M⊆(–one-point compactification of″T) –⋃T″ ∪ ⋃T ⊆ ⋃M
then obtain R where R∈M⋃T∈R by auto
have ⋃T∉⋃T using mem˙not˙refl by auto
with ‹R∈M› ‹⋃T∈R› A(1) obtain K where K:R=–⋃T″∪(⋃T-K) K–is compact in″TK–is closed in″T
unfolding OPCompactification˙def by auto
from K(1,2) have B:–⋃T″ ∪ (⋃T) = R ∪ K unfolding IsCompact˙def by auto
with A(2) have K⊆⋃M by auto
from K(2) have K–is compact in″((–one-point compactification of″T)–restricted to″⋃T) using open˙subspace(2)
by auto
then have K–is compact in″(–one-point compactification of″T) using compact˙subspace˙imp˙compact
‹K–is closed in″T› unfolding IsClosed˙def by auto
with ‹K⊆⋃M› A(1) have (∃N∈FinPow(M). K ⊆ ⋃N) unfolding IsCompact˙def by auto
then obtain N where N∈FinPow(M) K⊆⋃N by auto

with ‹R∈M› have (N ∪–R″)∈FinPow(M)R∪K⊆⋃(N∪–R″) unfolding FinPow˙def by auto
with B show ∃N∈FinPow(M). –⋃T″ ∪ (⋃T)⊆⋃N by auto
qed

The one point compactification is Hausdorff iff the original space is also Hausdorff and locally compact.

lemma (in topology0) op˙compact˙T2˙1:
assumes (–one-point compactification of″T)–is T_2″
shows T–is T_2″
using T2˙here[OF assms, of ⋃T] open˙subspace by auto

lemma (in topology0) op˙compact˙T2˙2:
assumes (–one-point compactification of″T)–is T_2″
shows T–is locally-compact″
proof-
{
fix x assume x∈⋃T
then have x∈–⋃T″∪(⋃T) by auto
moreover have ⋃T∈–⋃T″∪(⋃T) by auto moreover
from ‹x∈⋃T› have x≠⋃T using mem˙not˙refl by auto
ultimately have ∃U∈–one-point compactification of″T. ∃V∈–one-point compactification of″T. x ∈ U ∧ (⋃T) ∈ V ∧ U ∩ V = 0
using assms op˙compact˙total unfolding isT2˙def by auto
then obtain U V where UV:U∈–one-point compactification of″TV∈–one-point compactification of″T
x∈U⋃T∈VU∩V=0 by auto
from ‹V∈–one-point compactification of″T› ‹⋃T∈V› mem˙not˙refl obtain K where K:V=–⋃T″∪(⋃T-K)K–is closed in″TK–is compact in″T
unfolding OPCompactification˙def by auto
from ‹U∈–one-point compactification of″T› have U⊆–⋃T″∪(⋃T) unfolding OPCompactification˙def
using op˙compact˙total by auto
with ‹U∩V=0› K have U⊆KK⊆⋃T unfolding IsClosed˙def by auto
then have (⋃T)∩U=U by auto moreover
from UV(1) have ((⋃T)∩U)∈(–one-point compactification of″T)–restricted to″⋃T
unfolding RestrictedTo˙def by auto
ultimately have U∈T using open˙subspace(2) by auto
with ‹x∈U›‹U⊆K› have x∈int(K) using Top˙2˙L6 by auto
with ‹K⊆⋃T› ‹K–is compact in″T› have ∃A∈Pow(⋃T). x∈int(A)∧ A–is compact in″T by auto
}
then have ∀x∈⋃T. ∃A∈Pow(⋃T). x∈int(A)∧ A–is compact in″T by auto
then show thesis using op˙compact˙T2˙1[OF assms] exist˙compact˙neig˙T2˙imp˙locally˙compact by auto
qed

lemma (in topology0) op˙compact˙T2˙3:

assumes T–is locally-compact″ T–is T_2″
shows (–one-point compactification of″T)–is T_2″
proof-
{
fix x y assume x≠yx∈⋃(–one-point compactification of″T)y∈⋃(–one-point compactification of″T)
then have S:x∈–⋃T″∪(⋃T)y∈–⋃T″∪(⋃T) using op˙compact˙total by auto
{
assume x∈⋃Ty∈⋃T
with ‹x≠y› have ∃U∈T. ∃V∈T. x∈U∧y∈V∧U∩V=0 using assms(2) unfolding isT2˙def by auto
then have ∃U∈(–one-point compactification of″T). ∃V∈(–one-point compactification of″T). x∈U∧y∈V∧U∩V=0
unfolding OPCompactification˙def by auto
}
moreover
{
assume x∉⋃T∨y∉⋃T
with S have x=⋃T∨y=⋃T by auto
with ‹x≠y› have (x=⋃T∧y≠⋃T)∨(y=⋃T∧x≠⋃T) by auto
with S have (x=⋃T∧y∈⋃T)∨(y=⋃T∧x∈⋃T) by auto
then obtain Ky Kx where (x=⋃T∧ Ky–is compact in″T∧y∈int(Ky))∨(y=⋃T∧ Kx–is compact in″T∧x∈int(Kx))
using assms(1) locally˙compact˙exist˙compact˙neig by blast
then have (x=⋃T∧ Ky–is compact in″T∧ Ky–is closed in″T∧y∈int(Ky))∨(y=⋃T∧ Kx–is compact in″T∧ Kx–is closed in″T∧x∈int(Kx))
using in˙t2˙compact˙is˙cl assms(2) by auto
then have (x∈–⋃T″∪(⋃T-Ky)∧y∈int(Ky)∧ Ky–is compact in″T∧ Ky–is closed in″T)∨(y∈–⋃T″∪(⋃T-Kx)∧x∈int(Kx)∧ Kx–is compact in″T∧ Kx–is closed in″T)
by auto moreover
{
fix K
assume A:K–is closed in″TK–is compact in″T
then have K⊆⋃T unfolding IsClosed˙def by auto
moreover have ⋃T∉⋃T using mem˙not˙refl by auto
ultimately have (–⋃T″∪(⋃T-K))∩K=0 by auto
then have (–⋃T″∪(⋃T-K))∩int(K)=0 using Top˙2˙L1 by auto moreover
from A have –⋃T″∪(⋃T-K)∈(–one-point compactification of″T) unfolding OPCompactification˙def
IsClosed˙def by auto moreover
have int(K)∈(–one-point compactification of″T) using Top˙2˙L2 unfolding OPCompactification˙def
by auto ultimately
have int(K)∈(–one-point compactification of″T)∧–⋃T″∪(⋃T-K)∈(–one-point compactification of″T)∧(–⋃T″∪(⋃T-K))∩int(K)=0
by auto
}
ultimately have (–⋃T″ ∪ (⋃T - Ky)∈(–one-point compactification of″T)∧int(Ky)∈(–one-point compactification of″T)∧x ∈ –⋃T″ ∪ (⋃T - Ky) ∧ y ∈ int(Ky) ∧ (–⋃T″∪(⋃T-Ky))∩int(Ky)=0)

∨
(–⋃T″ ∪ (⋃T - Kx)∈(–one-point compactification of″T)∧int(Kx)∈(–one-point compactification of″T)∧y ∈ –⋃T″ ∪ (⋃T - Kx) ∧ x ∈ int(Kx) ∧ (–⋃T″∪(⋃T-Kx))∩int(Kx)=0) by auto
moreover
{
assume (–⋃T″ ∪ (⋃T - Ky)∈(–one-point compactification of″T)∧int(Ky)∈(–one-point compactification of″T)∧x ∈ –⋃T″ ∪ (⋃T - Ky) ∧ y ∈ int(Ky) ∧ (–⋃T″∪(⋃T-Ky))∩int(Ky)=0)
then have ∃U∈(–one-point compactification of″T). ∃V∈(–one-point compactification of″T). x∈U∧y∈V∧U∩V=0 using exI[OF exI[of ˙ int(Ky)],of λU V. U∈(–one-point compactification of″T)∧V∈(–one-point compactification of″T) ∧ x∈U∧y∈V∧U∩V=0 –⋃T″∪(⋃T-Ky)]
by auto
} moreover
{
assume (–⋃T″ ∪ (⋃T - Kx)∈(–one-point compactification of″T)∧int(Kx)∈(–one-point compactification of″T)∧y ∈ –⋃T″ ∪ (⋃T - Kx) ∧ x ∈ int(Kx) ∧ (–⋃T″∪(⋃T-Kx))∩int(Kx)=0)
then have ∃U∈(–one-point compactification of″T). ∃V∈(–one-point compactification of″T). x∈U∧y∈V∧U∩V=0 using exI[OF exI[of ˙ –⋃T″∪(⋃T-Kx)],of λU V. U∈(–one-point compactification of″T)∧V∈(–one-point compactification of″T) ∧ x∈U∧y∈V∧U∩V=0int(Kx)]
by blast
}
ultimately have ∃U∈(–one-point compactification of″T). ∃V∈(–one-point compactification of″T). x∈U∧y∈V∧U∩V=0 by auto
}
ultimately have ∃U∈(–one-point compactification of″T). ∃V∈(–one-point compactification of″T). x∈U∧y∈V∧U∩V=0 by auto
}
then show thesis unfolding isT2˙def by auto
qed

In conclusion, every locally compact Hausdorff topological space is regular; since this property is hereditary.

corollary (in topology0) locally˙compact˙T2˙imp˙regular:
assumes T–is locally-compact″ T–is T_2″
shows T–is regular″
proof-
from assms have (–one-point compactification of″T) –is T_2″ using op˙compact˙T2˙3 by auto
then have (–one-point compactification of″T) –is T_4″ unfolding isT4˙def using T2˙is˙T1 topology0.T2˙compact˙is˙normal
op˙comp˙is˙top unfolding topology0˙def using op˙compact˙total compact˙op by auto
then have (–one-point compactification of″T) –is T_3″ using topology0.T4˙is˙T3 op˙comp˙is˙top unfolding topology0˙def
by auto
then have (–one-point compactification of″T) –is regular″ using isT3˙def by auto
moreover

have ⋃T⊆⋃(–one-point compactification of″T) using op˙compact˙total by auto
ultimately have ((–one-point compactification of″T)–restricted to″⋃T) –is regular″ using regular˙here by auto
then show T–is regular″ using open˙subspace(2) by auto
qed

This last corollary has an explanation: In Hausdorff spaces, compact sets are closed and regular spaces are exactly the "locally closed spaces"(those which have a neighbourhood basis of closed sets). So the neighbourhood basis of compact sets also works as the neighbourhood basis of closed sets we needed to find.

definition
IsLocallyClosed (˙–is locally-closed″)
where T–is locally-closed″ ≡ T–is locally″(λB TT. B–is closed in″TT)

lemma (in topology0) regular˙locally˙closed:
shows T–is regular″ ⟷ (T–is locally-closed″)
proof
assume T–is regular″
then have a:∀x∈⋃T. ∀U∈T. (x∈U) ⟶ (∃V∈T. x ∈ V ∧ cl(V) ⊆ U) using regular˙imp˙exist˙clos˙neig by auto
{
fix x b assume x∈⋃Tb∈Tx∈b
with a obtain V where V∈Tx∈Vcl(V)⊆b by blast
note ‹cl(V)⊆b› moreover
from ‹V∈T› have V⊆⋃T by auto
then have V⊆cl(V) using cl˙contains˙set by auto
with ‹x∈V›‹V∈T› have x∈int(cl(V)) using Top˙2˙L6 by auto moreover
from ‹V⊆⋃T› have cl(V)–is closed in″T using cl˙is˙closed by auto
ultimately have x∈int(cl(V))cl(V)⊆bcl(V)–is closed in″T by auto
then have ∃K∈Pow(b). x∈int(K)∧K–is closed in″T by auto
}
then show T–is locally-closed″ unfolding IsLocally˙def[OF topSpaceAssum] IsLocallyClosed˙def
by auto
next
assume T–is locally-closed″
then have a:∀x∈⋃T. ∀b∈T. x∈b ⟶ (∃K∈Pow(b). x∈int(K)∧K–is closed in″T) unfolding IsLocally˙def[OF topSpaceAssum]
IsLocallyClosed˙def by auto
{
fix x b assume x∈⋃Tb∈Tx∈b
with a obtain K where K:K⊆bx∈int(K)K–is closed in″T by blast
have int(K)⊆K using Top˙2˙L1 by auto
with K(3) have cl(int(K))⊆K using Top˙3˙L13 by auto
with K(1) have cl(int(K))⊆b by auto moreover
have int(K)∈T using Top˙2˙L2 by auto moreover
note ‹x∈int(K)› ultimately have ∃V∈T. x∈V∧ cl(V)⊆b by auto
}

then have ∀x∈⋃T. ∀b∈T. x∈b ⟶ (∃V∈T. x∈V∧ cl(V)⊆b) by auto
then show T–is regular″ using exist˙clos˙neig˙imp˙regular by auto
qed

67.5 Hereditary properties and local properties

In this section, we prove a relation between a property and its local property for hereditary properties. Then we apply it to locally-Hausdorff or locally-T_2. We also prove the relation between locally-T_2 and another property that appeared when considering anti-properties, the anti-hyperconnectness.

If a property is hereditary in open sets, then local properties are equivalent to find just one open neighbourhood with that property instead of a whole local basis.

lemma (in topology0) her˙P˙is˙loc˙P:
assumes ∀TT. ∀B∈Pow(⋃TT). ∀A∈TT. TT–is a topology″∧P(B,TT) ⟶ P(B∩A,TT)
shows (T–is locally″P) ⟷ (∀x∈⋃T. ∃A∈T. x∈A∧P(A,T))
proof
assume A:T–is locally″P
{
fix x assume x:x∈⋃T
with A have ∀b∈T. x∈b ⟶ (∃c∈Pow(b). x∈int(c)∧P(c,T)) unfolding IsLocally˙def[OF topSpaceAssum]
by auto moreover
note x moreover
have ⋃T∈T using topSpaceAssum unfolding IsATopology˙def by auto
ultimately have ∃c∈Pow(⋃T). x∈int(c)∧ P(c,T) by auto
then obtain c where c:c⊆⋃Tx∈int(c)P(c,T) by auto
have P:int(c)∈T using Top˙2˙L2 by auto moreover
from c(1,3) topSpaceAssum assms have ∀A∈T. P(c∩A,T) by auto
ultimately have P(c∩int(c),T) by auto moreover
from Top˙2˙L1[of c] have int(c)⊆c by auto
then have c∩int(c)=int(c) by auto
ultimately have P(int(c),T) by auto
with P c(2) have ∃V∈T. x∈V∧P(V,T) by auto
}
then show ∀x∈⋃T. ∃V∈T. x∈V∧P(V,T) by auto
next
assume A:∀x∈⋃T. ∃A∈T. x ∈ A ∧ P(A, T)
{
fix x assume x:x∈⋃T
{
fix b assume b:x∈bb∈T
from x A obtain A where A˙def:A∈Tx∈AP(A,T) by auto
from A˙def(1,3) assms topSpaceAssum have ∀G∈T. P(A∩G,T) by auto
with b(2) have P(A∩b,T) by auto
moreover from b(1) A˙def(2) have x∈A∩b by auto moreover
have A∩b∈T using b(2) A˙def(1) topSpaceAssum IsATopology˙def by auto

then have int(A∩b)=A∩b using Top˙2˙L3 by auto
ultimately have x∈int(A∩b)∧P(A∩b,T) by auto
then have ∃c∈Pow(b). x∈int(c)∧P(c,T) by auto
}
then have ∀b∈T. x∈b⟶(∃c∈Pow(b). x∈int(c)∧P(c,T)) by auto
}
then show T–is locally˝P unfolding IsLocally˙def[OF topSpaceAssum] by auto
qed

definition
IsLocallyT2 (˙–is locally-T_2˝ 70)
where T–is locally-T_2˝≡T–is locally˝(λB. λT. (T–restricted to˝B)–is T_2˝)

Since T_2 is an hereditary property, we can apply the previous lemma.

corollary (in topology0) loc˙T2:
shows (T–is locally-T_2˝) ⟷ (∀x∈⋃T. ∃A∈T. x∈A∧(T–restricted to˝A)–is T_2˝)
proof-
{
fix TT B A assume TT:TT–is a topology˝ (TT–restricted to˝B)–is T_2˝ A∈TTB∈Pow(⋃TT)
then have s:B∩A⊆BB⊆⋃TT by auto
then have (TT–restricted to˝(B∩A))=(TT–restricted to˝B)–restricted to˝(B∩A)
using subspace˙of˙subspace
by auto moreover
have ⋃(TT–restricted to˝B)=B unfolding RestrictedTo˙def using s(2) by auto
then have B∩A⊆⋃(TT–restricted to˝B) using s(1) by auto moreover
note TT(2) ultimately have (TT–restricted to˝(B∩A))–is T_2˝ using T2˙here
by auto
}
then have ∀TT. ∀B∈Pow(⋃TT). ∀A∈TT. TT–is a topology˝∧(TT–restricted to˝B)–is T_2˝ ⟶ (TT–restricted to˝(B∩A))–is T_2˝
by auto
with her˙P˙is˙loc˙P[where P=λA. λTT. (TT–restricted to˝A)–is T_2˝] show thesis unfolding IsLocallyT2˙def by auto
qed

First, we prove that a locally-T_2 space is anti-hyperconnected.

Before starting, let's prove that an open subspace of an hyperconnected space is hyperconnected.

lemma(in topology0) open˙subspace˙hyperconn:
assumes T–is hyperconnected˝ U∈T
shows (T–restricted to˝U)–is hyperconnected˝
proof-
{
fix A B assume A∈(T–restricted to˝U)B∈(T–restricted to˝U)A∩B=0
then obtain AU BU where A=U∩AUB=U∩BU AU∈TBU∈T unfolding RestrictedTo˙def by auto

then have A∈TB∈T using topSpaceAssum assms(2) unfolding IsATopology˙def by auto
with ‹A∩B=0› have A=0∨B=0 using assms(1) unfolding IsHConnected˙def by auto
}
then show thesis unfolding IsHConnected˙def by auto
qed

lemma(in topology0) locally˙T2˙is˙antiHConn:
assumes T–is locally-T_2″
shows T–is anti-″IsHConnected
proof-
{
fix A assume A:A∈Pow(⋃T)(T–restricted to″A)–is hyperconnected″
{
fix x assume x∈A
with A(1) have x∈⋃T by auto moreover
have ⋃T∈T using topSpaceAssum unfolding IsATopology˙def by auto ultimately
have ∃c∈Pow(⋃T). x ∈ int(c) ∧ (T –restricted to″ c) –is T_2″ using assms
unfolding IsLocallyT2˙def IsLocally˙def[OF topSpaceAssum] by auto
then obtain c where c:c∈Pow(⋃T)x∈int(c)(T –restricted to″ c) –is T_2″ by auto
have ⋃(T –restricted to″ c)=(⋃T)∩c unfolding RestrictedTo˙def by auto
with ‹c∈Pow(⋃T)›‹⋃T∈T› have tot:⋃(T –restricted to″ c)=c by auto
have int(c)∈T using Top˙2˙L2 by auto
then have A∩(int(c))∈(T–restricted to″A) unfolding RestrictedTo˙def by auto
with A(2) have ((T–restricted to″A)–restricted to″(A∩(int(c))))–is hyperconnected″
using topology0.open˙subspace˙hyperconn unfolding topology0˙def using Top˙1˙L4
by auto
then have (T–restricted to″(A∩(int(c))))–is hyperconnected″ using subspace˙of˙subspace[of A∩(int(c))
AT] A(1) by force moreover
have int(c)⊆c using Top˙2˙L1 by auto
then have sub:A∩(int(c))⊆c by auto
then have A∩(int(c))⊆⋃(T –restricted to″ c) using tot by auto
then have ((T –restricted to″ c)–restricted to″(A∩(int(c)))) –is T_2″ using
T2˙here[OF c(3)] by auto
with sub have (T –restricted to″(A∩(int(c))))–is T_2″ using subspace˙of˙subspace[of A∩(int(c))
cT] ‹c∈Pow(⋃T)› by auto
ultimately have (T–restricted to″(A∩(int(c))))–is hyperconnected″(T –restricted to″(A∩(int(c))))–is T_2″
by auto
then have (T–restricted to″(A∩(int(c))))–is hyperconnected″(T –restricted to″(A∩(int(c))))–is anti-″IsHConnected
using topology0.T2˙imp˙anti˙HConn unfolding topology0˙def using Top˙1˙L4

```
by auto
    moreover
     have ⋃(T–restricted to″(A∩(int(c))))=(⋃T)∩A∩(int(c)) unfolding Restrict-
edTo˙def by auto
    with A(1) Top˙2˙L2 have ⋃(T–restricted to″(A∩(int(c))))=A∩(int(c)) by auto
    then have A∩(int(c))⊆⋃(T–restricted to″(A∩(int(c)))) by auto moreover
    have A∩(int(c))⊆⋃T using A(1) Top˙2˙L2 by auto
  then have (T–restricted to″(A∩(int(c))))–restricted to″(A∩(int(c)))=(T–restricted
to″(A∩(int(c))))
      using subspace˙of˙subspace[of A∩(int(c))A∩(int(c))T] by auto
      ultimately have (A∩(int(c)))–is in the spectrum of″IsHConnected unfolding
antiProperty˙def
      by auto
    then have A∩(int(c))≲1 using HConn˙spectrum by auto
    then have (A∩(int(c))=–x″) using lepoll˙1˙is˙sing ‹x∈A›‹x∈int(c)› by auto
     then have –x″∈(T–restricted to″A) using ‹(A∩(int(c))∈(T–restricted to″A))›
by auto
  }
  then have pointOpen:∀x∈A. –x″∈(T–restricted to″A) by auto
  {
    fix x y assume x≠yx∈Ay∈A
  with pointOpen have –x″∈(T–restricted to″A)–y″∈(T–restricted to″A)–x″∩–y″=0
      by auto
    with A(2) have –x″=0∨–y″=0 unfolding IsHConnected˙def by auto
    then have False by auto
  }
  then have uni:∀x∈A. ∀y∈A. x=y by auto
  {
    assume A≠0
    then obtain x where x∈A by auto
    with uni have A=–x″ by auto
    then have A≈1 using singleton˙eqpoll˙1 by auto
    then have A≲1 using eqpoll˙imp˙lepoll by auto
  }
  moreover
  {
    assume A=0
    then have A≈0 by auto
    then have A≲1 using empty˙lepollI eq˙lepoll˙trans by auto
  }
  ultimately have A≲1 by auto
 then have A–is in the spectrum of″IsHConnected using HConn˙spectrum by auto
 }
 then show thesis unfolding antiProperty˙def by auto
qed
```

Now we find a counter-example for: Every anti-hyperconnected space is locally-Hausdorff.

The example we are going to consider is the following. Put in X an anti-

hyperconnected topology, where an infinite number of points don't have finite sets as neighbourhoods. Then add a new point to the set, $p \notin X$. Consider the open sets on $X \cup p$ as the anti-hyperconnected topology and the open sets that contain p are $p \cup A$ where $X \setminus A$ is finite.

This construction equals the one-point compactification iff X is anti-compact; i.e., the only compact sets are the finite ones. In general this topology is contained in the one-point compactification topology, making it compact too.

It is easy to check that any open set containing p meets infinite other non-empty open set. The question is if such a topology exists.

```
theorem (in topology0) COF˙comp˙is˙top:
  assumes T–is T1″¬(⋃T≺nat)
   shows (((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T) –is a
topology″
proof-
  have N:⋃T∉(⋃T) using mem˙not˙refl by auto
  {
   fix M assume M:M⊆(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)
    let MT=–A∈M. A∈T″
    let MK=–A∈M. A∉T″
    have MM:(⋃MT)∪(⋃MK)=⋃M by auto
    have MN:⋃MT∈T using topSpaceAssum unfolding IsATopology˙def by auto
    then have sub:MK⊆(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″
      using M by auto
    then have MK⊆(–one-point compactification of″(CoFinite (⋃T))) by auto
    then have CO:⋃MK∈(–one-point compactification of″(CoFinite (⋃T))) using
     topology0.op˙comp˙is˙top[OF topology0˙CoCardinal[OF InfCard˙nat]] unfolding
Cofinite˙def
      IsATopology˙def by auto
    {
      assume AS:⋃MK=–⋃T″
      moreover have ∀R∈MK. R⊆⋃MK by auto
      ultimately have ∀R∈MK. R⊆–⋃T″ by auto
      then have ∀R∈MK. R=–⋃T″∨R=0 by force moreover
      with sub have ∀R∈MK. R=0 by auto
      then have ⋃MK=0 by auto
      with AS have False by auto
    }
   with CO have CO2:⋃MK∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″
by auto
    {
      assume ⋃MK∈(CoFinite (⋃T))
      then have ⋃MK∈T using assms(1) T1˙cocardinal˙coarser by auto
      with MN have –⋃MT,⋃MK″⊆(T) by auto
         then have (⋃MT)∪(⋃MK)∈T using union˙open[OF topSpaceAssum, of
–⋃MT,⋃MK″] by auto
      then have ⋃M∈T using MM by auto
```

}
moreover
{
assume ⋃MK∉(CoFinite (⋃T))
with CO obtain B where B–is compact in″(CoFinite (⋃T))B–is closed in″(CoFinite (⋃T))
⋃MK=–⋃CoFinite ⋃T″∪(⋃(CoFinite ⋃T)-B) unfolding OPCompactification˙def by auto
then have MK:⋃MK=–⋃T″∪(⋃T-B)B–is closed in″(CoFinite (⋃T))
using union˙cocardinal unfolding Cofinite˙def by auto
then have B:B⊆⋃T B≺nat∨B=⋃T using closed˙sets˙cocardinal unfolding Cofinite˙def by auto
{
assume B=⋃T
with MK have ⋃MK=–⋃T″ by auto
then have False using CO2 by auto
}
with B have B⊆⋃T and natB:B≺nat by auto
have (⋃T-(⋃MT))∩B⊆B by auto
then have (⋃T-(⋃MT))∩B≲B using subset˙imp˙lepoll by auto
then have (⋃T-(⋃MT))∩B≺nat using natB lesspoll˙trans1 by auto
then have ((⋃T-(⋃MT))∩B)–is closed in″(CoFinite (⋃T)) using closed˙sets˙cocardinal B(1) unfolding Cofinite˙def by auto
then have ⋃T-((⋃T-(⋃MT))∩B)∈(CoFinite (⋃T)) unfolding IsClosed˙def using union˙cocardinal unfolding Cofinite˙def by auto
also have ⋃T-((⋃T-(⋃MT))∩B)=(⋃T-(⋃T-(⋃MT)))∪(⋃T-B) by auto
also have ...=(⋃MT)∪(⋃T-B) by auto
ultimately have P:(⋃MT)∪(⋃T-B)∈(CoFinite (⋃T)) by auto
then have eq:⋃T-(⋃T-((⋃MT)∪(⋃T-B)))=(⋃MT)∪(⋃T-B) by auto
from P eq have (⋃T-((⋃MT)∪(⋃T-B)))–is closed in″(CoFinite (⋃T)) unfolding IsClosed˙def
using union˙cocardinal[of nat⋃T] unfolding Cofinite˙def by auto moreover
have (⋃T-((⋃MT)∪(⋃T-B)))∩⋃T=(⋃T-((⋃MT)∪(⋃T-B))) by auto
then have (CoFinite ⋃T)–restricted to″(⋃T-((⋃MT)∪(⋃T-B)))=CoFinite (⋃T-((⋃MT)∪(⋃T-B))) using subspace˙cocardinal unfolding Cofinite˙def by auto
then have (⋃T-((⋃MT)∪(⋃T-B)))–is compact in″((CoFinite ⋃T)–restricted to″(⋃T-((⋃MT)∪(⋃T-B)))) using cofinite˙compact
union˙cocardinal unfolding Cofinite˙def by auto
then have (⋃T-((⋃MT)∪(⋃T-B)))–is compact in″(CoFinite ⋃T) using compact˙subspace˙imp˙compact by auto ultimately
have –⋃T″∪(⋃T-(⋃T-((⋃MT)∪(⋃T-B))))∈(–one-point compactification of″(CoFinite (⋃T)))
unfolding OPCompactification˙def using union˙cocardinal unfolding Cofinite˙def by auto
with eq have –⋃T″∪((⋃MT)∪(⋃T-B))∈(–one-point compactification of″(CoFinite (⋃T))) by auto
moreover have AA:–⋃T″∪((⋃MT)∪(⋃T-B))=((⋃MT)∪(⋃MK)) using MK(1) by auto
ultimately have AA2:((⋃MT)∪(⋃MK))∈(–one-point compactification of″(CoFinite

(⋃T))) by auto
{
assume AS:(⋃MT)∪(⋃MK)=–⋃T″
from MN have T:⋃T∉⋃MT using N by auto
{
fix x assume G:x∈⋃MT
then have x∈(⋃MT)∪(⋃MK) by auto
with AS have x∈–⋃T″ by auto
then have x=⋃T by auto
with T have False using G by auto
}
then have ⋃MT=0 by auto
with AS have (⋃MK)=–⋃T″ by auto
then have False using CO2 by auto
}
with AA2 have ((⋃MT)∪(⋃MK))∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″ by auto
with MM have ⋃M∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″ by auto
}
ultimately
have ⋃M∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T by auto
}
then have ∀M∈Pow(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T). ⋃M∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T
by auto moreover
{
fix U V assume U∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪TV∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T moreover
{
assume U∈TV∈T
then have U∩V∈T using topSpaceAssum unfolding IsATopology˙def by auto
then have U∩V∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T by auto
}
moreover
{
assume UV:U∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)V∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)
then have O:U∩V∈(–one-point compactification of″(CoFinite (⋃T))) using topology0.op˙comp˙is˙top[OF topology0˙CoCardinal[OF InfCard˙nat]] unfolding Cofinite˙def
IsATopology˙def by auto
then have ⋃T∩(U∩V)∈(–one-point compactification of″(CoFinite (⋃T)))–restricted to″⋃T
unfolding RestrictedTo˙def by auto
then have ⋃T∩(U∩V)∈CoFinite ⋃T using topology0.open˙subspace(2)[OF topology0˙CoCardinal[OF InfCard˙nat]]

union˙cocardinal unfolding Cofinite˙def by auto
from UV have U≠–⋃T″V≠–⋃T″⋃T∩U∈(–one-point compactification of″(CoFinite (⋃T)))–restricted to″⋃T⋃T∩V∈(–one-point compactification of″(CoFinite (⋃T)))–restricted to″⋃T
unfolding RestrictedTo˙def by auto
then have R:U≠–⋃T″V≠–⋃T″⋃T∩U∈CoFinite ⋃T⋃T∩V∈CoFinite ⋃T using topology0.open˙subspace(2)[OF topology0˙CoCardinal[OF InfCard˙nat]]
union˙cocardinal unfolding Cofinite˙def by auto
from UV have U⊆⋃(–one-point compactification of″(CoFinite (⋃T)))V⊆⋃(–one-point compactification of″(CoFinite (⋃T))) by auto
then have U⊆–⋃T″∪⋃TV⊆–⋃T″∪⋃T using topology0.op˙compact˙total[OF topology0˙CoCardinal[OF InfCard˙nat]]
union˙cocardinal unfolding Cofinite˙def by auto
then have E:U=(⋃T∩U)∪(–⋃T″∩U)V=(⋃T∩V)∪(–⋃T″∩V)U∩V=(⋃T∩U∩V)∪(–⋃T″∩U∩V) by auto
{
assume Q:U∩V=–⋃T″
then have RR:⋃T∩(U∩V)=0 using N by auto
{
assume ⋃T∩U=0
with E(1) have U=–⋃T″∩U by auto
also have ...⊆–⋃T″ by auto
ultimately have U⊆–⋃T″ by auto
then have U=0∨U=–⋃T″ by auto
with R(1) have U=0 by auto
then have U∩V=0 by auto
then have False using Q by auto
}
moreover
{
assume ⋃T∩V=0
with E(2) have V=–⋃T″∩V by auto
also have ...⊆–⋃T″ by auto
ultimately have V⊆–⋃T″ by auto
then have V=0∨V=–⋃T″ by auto
with R(2) have V=0 by auto
then have U∩V=0 by auto
then have False using Q by auto
}
moreover
{
assume ⋃T∩U≠0⋃T∩V≠0
with R(3,4) have (⋃T∩U)∩(⋃T∩V)≠0 using Cofinite˙nat˙HConn[OF assms(2)]
unfolding IsHConnected˙def by auto
then have ⋃T∩(U∩V)≠0 by auto
then have False using RR by auto
}
ultimately have False by auto

}
with O have U∩V∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T by auto
}
moreover
{
assume UV:U∈TV∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″
from UV(2) obtain B where V∈(CoFinite ⋃T)∨(V=–⋃T″∪(⋃T-B)∧B–is closed in″(CoFinite (⋃T))) unfolding OPCompactification˙def
using union˙cocardinal unfolding Cofinite˙def by auto
with assms(1) have V∈T∨(V=–⋃T″∪(⋃T-B)∧B–is closed in″(CoFinite (⋃T))) using T1˙cocardinal˙coarser by auto
then have V∈T∨(U∩V=U∩(⋃T-B)∧B–is closed in″(CoFinite (⋃T))) using UV(1) N by auto
then have V∈T∨(U∩V=U∩(⋃T-B)∧(⋃T-B)∈(CoFinite (⋃T))) unfolding IsClosed˙def using union˙cocardinal unfolding Cofinite˙def by auto
then have V∈T∨(U∩V=U∩(⋃T-B)∧(⋃T-B)∈T) using assms(1) T1˙cocardinal˙coarser by auto
with UV(1) have U∩V∈T using topSpaceAssum unfolding IsATopology˙def by auto
then have U∩V∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T by auto
}
moreover
{
assume UV:U∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″V∈T
from UV(1) obtain B where U∈(CoFinite ⋃T)∨(U=–⋃T″∪(⋃T-B)∧B–is closed in″(CoFinite (⋃T))) unfolding OPCompactification˙def
using union˙cocardinal unfolding Cofinite˙def by auto
with assms(1) have U∈T∨(U=–⋃T″∪(⋃T-B)∧B–is closed in″(CoFinite (⋃T))) using T1˙cocardinal˙coarser by auto
then have U∈T∨(U∩V=(⋃T-B)∩V∧B–is closed in″(CoFinite (⋃T))) using UV(2) N by auto
then have U∈T∨(U∩V=(⋃T-B)∩V∧(⋃T-B)∈(CoFinite (⋃T))) unfolding IsClosed˙def using union˙cocardinal unfolding Cofinite˙def by auto
then have U∈T∨(U∩V=(⋃T-B)∩V∧(⋃T-B)∈T) using assms(1) T1˙cocardinal˙coarser by auto
with UV(2) have U∩V∈T using topSpaceAssum unfolding IsATopology˙def by auto
then have U∩V∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T by auto
}
ultimately
have U∩V∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T by auto
}
ultimately show thesis unfolding IsATopology˙def by auto
qed

The previous construction preserves anti-hyperconnectedness.

theorem (in topology0) COF˙comp˙antiHConn:
assumes T–is anti-″IsHConnected ¬(⋃T≺nat)
shows (((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T) –is anti-″IsHConnected
proof-
have N:⋃T∉(⋃T) using mem˙not˙refl by auto
from assms(1) have T1:T–is T_1″ using anti˙HConn˙imp˙T1 by auto
have tot1:⋃(–one-point compactification of″(CoFinite (⋃T)))=–⋃T″∪⋃T using
topology0.op˙compact˙total[OF topology0˙CoCardinal[OF InfCard˙nat], of ⋃T]
union˙cocardinal[of nat⋃T] unfolding Cofinite˙def by auto
then have (⋃(–one-point compactification of″(CoFinite (⋃T))))∪⋃T=–⋃T″∪⋃T
by auto moreover
have ⋃((–one-point compactification of″(CoFinite (⋃T)))∪T)=(⋃(–one-point
compactification of″(CoFinite (⋃T))))∪⋃T
by auto
ultimately have tot2:⋃((–one-point compactification of″(CoFinite (⋃T)))∪T)=–⋃T″∪⋃T
by auto
have –⋃T″∪⋃T∈(–one-point compactification of″(CoFinite (⋃T))) using union˙open[OF
topology0.op˙comp˙is˙top[OF topology0˙CoCardinal[OF InfCard˙nat]],of –one-point
compactification of″(CoFinite (⋃T))]
tot1 unfolding Cofinite˙def by auto moreover
{
assume ⋃T=0
with assms(2) have ¬(0≺nat) by auto
then have False unfolding lesspoll˙def using empty˙lepollI eqpoll˙0˙is˙0
eqpoll˙sym by auto
}
then have ⋃T≠0 by auto
with N have Not:¬(⋃T⊆–⋃T″) by auto
{
assume –⋃T″∪⋃T=–⋃T″ moreover
have ⋃T⊆–⋃T″∪⋃T by auto ultimately
have ⋃T⊆–⋃T″ by auto
with Not have False by auto
}
then have –⋃T″∪⋃T≠–⋃T″ by auto ultimately
have –⋃T″∪⋃T∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″ by
auto
then have –⋃T″∪⋃T∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T
by auto
then have –⋃T″∪⋃T⊆⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)
by auto moreover
have (–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T⊆(–one-point
compactification of″(CoFinite (⋃T)))∪T by auto
then have ⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)⊆⋃((–one-point
compactification of″(CoFinite (⋃T)))∪T) by auto
with tot2 have ⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)⊆–⋃T″∪⋃T
by auto
ultimately have TOT:⋃(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)=–⋃T″∪⋃T
by auto

{
fix A assume AS:A⊆⋃T ((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″A) –is hyperconnected″
from AS(1,2) have e0:(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″A=((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″⋃T)–restricted to″A
using subspace˙of˙subspace[of A⋃T(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)] TOT by auto
have e1:((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″(⋃T))=(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)–restricted to″⋃T)∪(T–restricted to″⋃T)
unfolding RestrictedTo˙def by auto
{
fix A assume A∈T–restricted to″⋃T
then obtain B where B∈TA=B∩⋃T unfolding RestrictedTo˙def by auto
then have A=B by auto
with ‹B∈T› have A∈T by auto
}
then have T–restricted to″⋃T⊆T by auto moreover
{
fix A assume A∈T
then have ⋃T∩A=A by auto
with ‹A∈T› have A∈T–restricted to″⋃T unfolding RestrictedTo˙def by auto
}
ultimately have T–restricted to″⋃T=T by auto moreover
{
fix A assume A∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)–restricted to″⋃T
then obtain B where B∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″⋃T∩B=A unfolding RestrictedTo˙def by auto
then have B∈(–one-point compactification of″(CoFinite (⋃T)))⋃T∩B=A by auto
then have A∈(–one-point compactification of″(CoFinite (⋃T)))–restricted to″⋃T unfolding RestrictedTo˙def by auto
then have A∈(CoFinite (⋃T)) using topology0.open˙subspace(2)[OF topology0˙CoCardinal[OF InfCard˙nat]]
union˙cocardinal unfolding Cofinite˙def by auto
with T1 have A∈T using T1˙cocardinal˙coarser by auto
}
then have ((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)–restricted to″⋃T⊆T by auto
moreover note e1 ultimately
have (((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″ ∪ T) –restricted to″ (⋃T)) =T by auto
with e0 have (((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″A=T–restricted to″A by auto
with assms(1) AS have A–is in the spectrum of″IsHConnected unfolding antiProperty˙def by auto
}

then have reg:∀ A∈Pow(⋃T). (((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″A) –is hyperconnected″) ⟶(A–is in the spectrum of″IsHConnected) by auto
have ⋃T∈T using topSpaceAssum unfolding IsATopology˙def by auto
then have P:⋃T∈(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T) by auto
{
fix B assume sub:B∈Pow(⋃T ∪–⋃T″) and hyp:(((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″B) –is hyperconnected″)
from P have subop:⋃T∩B∈((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″B) unfolding RestrictedTo˙def by auto
with hyp have hypSub:(((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″B)–restricted to″(⋃T∩B))–is hyperconnected″ using topology0.open˙subspace˙hyperconn topology0.Top˙1˙L4 COF˙comp˙is˙top[OF T1 assms(2)] unfolding topology0˙def by auto
from sub TOT have B ⊆ ⋃((–one-point compactification of″(CoFinite ⋃T)) -—⋃T″″ ∪ T) by auto
then have ((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″(⋃T∩B))=((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″B)–restricted to″(⋃T∩B)
using subspace˙of˙subspace[of ⋃T∩BB(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)] by auto
with hypSub have (((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″ ∪ T) –restricted to″ (⋃T ∩ B))–is hyperconnected″ by auto
with reg have (⋃T∩B)–is in the spectrum of″IsHConnected by auto
then have le:⋃T∩B≲1 using HConn˙spectrum by auto
{
fix x assume x:x∈⋃T∩B
with le have sing:⋃T∩B=–x″ using lepoll˙1˙is˙sing by auto
{
fix y assume y:y∈B
then have y∈⋃T ∪–⋃T″ using sub by auto
with y have y∈⋃T∩B∨y=⋃T by auto
with sing have y=x∨y=⋃T by auto
}
then have B⊆–x,⋃T″ by auto
with x have disj:B=–x″∨B=–x,⋃T″ by auto
{
assume ⋃T∈B
with disj have B:B=–x,⋃T″ by auto
from sing subop have singOp:–x″∈((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″B)
by auto
have –x″–is closed in″(CoFinite ⋃T) using topology0.T1˙iff˙singleton˙closed[OF topology0˙CoCardinal[OF InfCard˙nat]] cocardinal˙is˙T1[OF InfCard˙nat]
x union˙cocardinal unfolding Cofinite˙def by auto
moreover
have Finite(–x″) by auto
then have spec:–x″–is in the spectrum of″ (λT. (⋃T) –is compact in″T) using compact˙spectrum by auto

have ((CoFinite ⋃T)–restricted to″–x″)–is a topology″⋃((CoFinite ⋃T)–restricted to″–x″)=–x″
using topology0.Top˙1˙L4[OF topology0˙CoCardinal[OF InfCard˙nat]] unfolding RestrictedTo˙def Cofinite˙def
using x union˙cocardinal by auto
with spec have –x″–is compact in″((CoFinite ⋃T)–restricted to″–x″) unfolding Spec˙def
by auto
then have –x″–is compact in″(CoFinite ⋃T) using compact˙subspace˙imp˙compact
by auto moreover note x
ultimately have –⋃T″∪(⋃T-–x″)∈–one-point compactification of″(CoFinite (⋃T)) unfolding OPCompactification˙def
using union˙cocardinal unfolding Cofinite˙def by auto moreover
{
assume A:–⋃T″∪(⋃T-–x″)=–⋃T″
{
fix y assume P:y∈⋃T-–x″
then have y∈–⋃T″∪(⋃T-–x″) by auto
then have y=⋃T using A by auto
with N P have False by auto
}
then have ⋃T-–x″=0 by auto
with x have ⋃T=–x″ by auto
then have ⋃T≈1 using singleton˙eqpoll˙1 by auto moreover
have 1≺nat using n˙lesspoll˙nat by auto
ultimately have ⋃T≺nat using eq˙lesspoll˙trans by auto
then have False using assms(2) by auto
}
ultimately have –⋃T″∪(⋃T-–x″)∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″ by auto
then have –⋃T″∪(⋃T-–x″)∈((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)) by auto
then have B∩(–⋃T″∪(⋃T-–x″))∈((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″B) unfolding RestrictedTo˙def by auto
moreover have B∩(–⋃T″∪(⋃T-–x″))=–⋃T″ using B by auto
ultimately have –⋃T″∈((((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)–restricted to″B) by auto
with singOp hyp N x have False unfolding IsHConnected˙def by auto
}
with disj have B=–x″ by auto
then have B≈1 using singleton˙eqpoll˙1 by auto
then have B≲1 using eqpoll˙imp˙lepoll by auto
}
then have ⋃T∩B≠0⟶B≲1 by blast
moreover
{
assume ⋃T∩B=0
with sub have B⊆–⋃T″ by auto
then have B≲–⋃T″ using subset˙imp˙lepoll by auto

```
    then have B≲1 using singleton˙eqpoll˙1 lepoll˙eq˙trans by auto
   }
   ultimately have B≲1 by auto
  then have B–is in the spectrum of″IsHConnected using HConn˙spectrum by auto
 }
 then show thesis unfolding antiProperty˙def using TOT by auto
qed
```

The previous construction, applied to a densely ordered topology, gives the desired counterexample. What happends is that every neighbourhood of $\bigcup$T is dense; because there are no finite open sets, and hence meets every non-empty open set. In conclusion, $\bigcup$T cannot be separated from other points by disjoint open sets.

Every open set that contains $\bigcup$T is dense, when considering the order topology in a densely ordered set with more than two points.

```
theorem neigh˙infPoint˙dense:
 fixes T X r
 defines T˙def:T ≡ (OrdTopology X r)
 assumes IsLinOrder(X,r) X–is dense with respect to″r
  ∃x y. x≠y∧x∈X∧y∈X U∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T
⋃T∈U
   V∈((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T V≠0
 shows U∩V≠0
proof
 have N:⋃T∉(⋃T) using mem˙not˙refl by auto
 have tot1:⋃(–one-point compactification of″(CoFinite (⋃T)))=–⋃T″∪⋃T using
topology0.op˙compact˙total[OF topology0˙CoCardinal[OF InfCard˙nat], of ⋃T]
       union˙cocardinal[of nat⋃T] unfolding Cofinite˙def by auto
 then have (⋃(–one-point compactification of″(CoFinite (⋃T))))∪⋃T=–⋃T″∪⋃T
by auto moreover
  have ⋃((–one-point compactification of″(CoFinite (⋃T)))∪T)=(⋃(–one-point
compactification of″(CoFinite (⋃T))))∪⋃T
   by auto
 ultimately have tot2:⋃((–one-point compactification of″(CoFinite (⋃T)))∪T)=–⋃T″∪⋃T
by auto
 have –⋃T″∪⋃T∈(–one-point compactification of″(CoFinite (⋃T))) using union˙open[OF
topology0.op˙comp˙is˙top[OF topology0˙CoCardinal[OF InfCard˙nat]],of –one-point
compactification of″(CoFinite (⋃T))]
   tot1 unfolding Cofinite˙def by auto moreover
 {
   assume ⋃T=0
   then have X=0 unfolding T˙def using union˙ordtopology[OF assms(2)] assms(4)
by auto
   then have False using assms(4) by auto
 }
 then have ⋃T≠0 by auto
 with N have Not:¬(⋃T⊆–⋃T″) by auto
 {
```

assume –⋃T″∪⋃T=–⋃T″ moreover
have ⋃T⊆–⋃T″∪⋃T by auto ultimately
have ⋃T⊆–⋃T″ by auto
with Not have False by auto
}
then have –⋃T″∪⋃T≠–⋃T″ by auto ultimately
have –⋃T″∪⋃T∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″ by auto
then have –⋃T″∪⋃T∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T by auto
then have –⋃T″∪⋃T⊆⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T) by auto moreover
have (–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T⊆(–one-point compactification of″(CoFinite (⋃T)))∪T by auto
then have ⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)⊆⋃((–one-point compactification of″(CoFinite (⋃T)))∪T) by auto
with tot2 have ⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)⊆–⋃T″∪⋃T by auto
ultimately have TOT:⋃(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)=–⋃T″∪⋃T by auto
assume A:U∩V=0
with assms(6) have NN:⋃T∉V by auto
with assms(7) have V∈(CoFinite ⋃T)∪T unfolding OPCompactification˙def using union˙cocardinal
unfolding Cofinite˙def by auto
moreover have T–is T_2″ unfolding T˙def using order˙top˙T2[OF assms(2)] assms(4) by auto
then have T1:T–is T_1″ using T2˙is˙T1 by auto
ultimately have VopT:V∈T using topology0.T1˙cocardinal˙coarser[OF topology0˙ordtopology(1)[OF assms(2)]]
unfolding T˙def by auto
from A assms(7) have V⊆⋃(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)-U by auto
then have V⊆(–⋃T″∪⋃T)-U using TOT by auto
then have V⊆(⋃T)-U using NN by auto
from N have U∉T using assms(6) by auto
then have U∉(CoFinite ⋃T)∪T using T1 topology0.T1˙cocardinal˙coarser[OF topology0˙ordtopology(1)[OF assms(2)]]
unfolding T˙def using union˙cocardinal union˙ordtopology[OF assms(2)] assms(4) by auto
with assms(5,6) obtain B where U:U=–⋃T″∪(⋃T-B) B–is closed in″(CoFinite ⋃T) B≠⋃T
unfolding OPCompactification˙def using union˙cocardinal unfolding Cofinite˙def by auto
then have U=–⋃T″∪(⋃T-B) B=⋃T ∨ B≺nat B≠⋃T using closed˙sets˙cocardinal unfolding Cofinite˙def
by auto
then have U=–⋃T″∪(⋃T-B) B≺nat by auto
with N have ⋃T-U=⋃T-(⋃T-B) by auto

then have ⋃T-U=B using U(2) unfolding IsClosed˙def using union˙cocardinal unfolding Cofinite˙def
by auto
with ‹B≺nat› have Finite(⋃T-U) using lesspoll˙nat˙is˙Finite by auto
with ‹V⊆(⋃T)-U› have Finite(V) using subset˙Finite by auto
from assms(8) obtain v where v∈V by auto
with VopT have ∃R∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪ –LeftRayX(X, r, b) . b ∈ X″ ∪–RightRayX(X, r, b) . b ∈ X″. R ⊆ V ∧ v ∈ R using
point˙open˙base˙neigh[OF Ordtopology˙is˙a˙topology(2)[OF assms(2)]] unfolding T˙def by auto
then obtain R where R˙def:R∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″ ∪ –LeftRayX(X, r, b) . b ∈ X″ ∪–RightRayX(X, r, b) . b ∈ X″ R⊆V v∈R by blast
moreover
{
assume R∈–IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X″
then obtain b c where lim:b∈Xc∈XR=IntervalX(X, r, b, c) by auto
with ‹v∈R› have ¬ Finite(R) using dense˙order˙inf˙intervals[OF assms(2) ˙ ˙ ˙ assms(3)]
by auto
with ‹R⊆V› ‹Finite(V)› have False using subset˙Finite by auto
} moreover
{
assume R∈–LeftRayX(X, r, b) . b ∈ X″
then obtain b where lim:b∈XR=LeftRayX(X, r, b) by auto
with ‹v∈R› have ¬ Finite(R) using dense˙order˙inf˙lrays[OF assms(2) ˙ ˙ assms(3)]
by auto
with ‹R⊆V› ‹Finite(V)› have False using subset˙Finite by auto
} moreover
{
assume R∈–RightRayX(X, r, b) . b ∈ X″
then obtain b where lim:b∈XR=RightRayX(X, r, b) by auto
with ‹v∈R› have ¬ Finite(R) using dense˙order˙inf˙rrays[OF assms(2)˙ ˙ assms(3)]
by auto
with ‹R⊆V› ‹Finite(V)› have False using subset˙Finite by auto
} ultimately
show False by auto
qed

A densely ordered set with more than one point gives an order topology. Applying the previous construction to this topology we get a non locally-Hausdorff space.

theorem OPComp˙cofinite˙dense˙order˙not˙loc˙T2:
fixes T X r
defines T˙def:T ≡ (OrdTopology X r)
assumes IsLinOrder(X,r) X–is dense with respect to″r
∃x y. x≠y∧x∈X∧y∈X
shows ¬(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)–is locally-T_2″)
proof
have N:⋃T∉(⋃T) using mem˙not˙refl by auto

have tot1:⋃(–one-point compactification of″(CoFinite (⋃T)))=–⋃T″∪⋃T using
topology0.op˙compact˙total[OF topology0˙CoCardinal[OF InfCard˙nat], of ⋃T]
union˙cocardinal[of nat⋃T] unfolding Cofinite˙def by auto
then have (⋃(–one-point compactification of″(CoFinite (⋃T))))∪⋃T=–⋃T″∪⋃T
by auto moreover
have ⋃((–one-point compactification of″(CoFinite (⋃T)))∪T)=(⋃(–one-point
compactification of″(CoFinite (⋃T))))∪⋃T
by auto
ultimately have tot2:⋃((–one-point compactification of″(CoFinite (⋃T)))∪T)=–⋃T″∪⋃T
by auto
have –⋃T″∪⋃T∈(–one-point compactification of″(CoFinite (⋃T))) using union˙open[OF
topology0.op˙comp˙is˙top[OF topology0˙CoCardinal[OF InfCard˙nat]],of –one-point
compactification of″(CoFinite (⋃T))]
tot1 unfolding Cofinite˙def by auto moreover
{
assume ⋃T=0
then have X=0 unfolding T˙def using union˙ordtopology[OF assms(2)] assms(4)
by auto
then have False using assms(4) by auto
}
then have ⋃T≠0 by auto
with N have Not:¬(⋃T⊆–⋃T″) by auto
{
assume –⋃T″∪⋃T=–⋃T″ moreover
have ⋃T⊆–⋃T″∪⋃T by auto ultimately
have ⋃T⊆–⋃T″ by auto
with Not have False by auto
}
then have –⋃T″∪⋃T≠–⋃T″ by auto ultimately
have –⋃T″∪⋃T∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″ by
auto
then have –⋃T″∪⋃T∈(–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T
by auto
then have –⋃T″∪⋃T⊆⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)
by auto moreover
have (–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T⊆(–one-point
compactification of″(CoFinite (⋃T)))∪T by auto
then have ⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)⊆⋃((–one-point
compactification of″(CoFinite (⋃T)))∪T) by auto
with tot2 have ⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)⊆–⋃T″∪⋃T
by auto
ultimately have TOT:⋃(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″)∪T)=–⋃T″∪⋃T
by auto
have T1:T–is T_1″ using order˙top˙T2[OF assms(2,4)] T2˙is˙T1 unfolding T˙def by
auto moreover
from assms(4) obtain b c where B:b∈Xc∈Xb≠c by auto
{
assume ⟨b,c⟩∉r
with assms(2) have ⟨c,b⟩∈r unfolding IsLinOrder˙def IsTotal˙def using ‹b∈X›‹c∈X›

by auto
with assms(3) B obtain z where z∈X-–b,c″⟨c,z⟩∈r⟨z,b⟩∈r unfolding IsDense˙def
by auto
then have IntervalX(X,r,c,b)≠0 unfolding IntervalX˙def using Order˙ZF˙2˙L1 by
auto
then have ¬(Finite(IntervalX(X,r,c,b))) using dense˙order˙inf˙intervals[OF assms(2)
˙ ‹c∈X›‹b∈X› assms(3)]
by auto moreover
have IntervalX(X,r,c,b)⊆X unfolding IntervalX˙def by auto
ultimately have ¬(Finite(X)) using subset˙Finite by auto
then have ¬(X≺nat) using lesspoll˙nat˙is˙Finite by auto
}
moreover
{
assume ⟨b,c⟩∈r
with assms(3) B obtain z where z∈X-–b,c″⟨b,z⟩∈r⟨z,c⟩∈r unfolding IsDense˙def
by auto
then have IntervalX(X,r,b,c)≠0 unfolding IntervalX˙def using Order˙ZF˙2˙L1 by
auto
then have ¬(Finite(IntervalX(X,r,b,c))) using dense˙order˙inf˙intervals[OF assms(2)
˙ ‹b∈X›‹c∈X› assms(3)]
by auto moreover
have IntervalX(X,r,b,c)⊆X unfolding IntervalX˙def by auto
ultimately have ¬(Finite(X)) using subset˙Finite by auto
then have ¬(X≺nat) using lesspoll˙nat˙is˙Finite by auto
}
ultimately have ¬(X≺nat) by auto
with T1 have top:((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)–is
a topology″ using topology0.COF˙comp˙is˙top[OF topology0˙ordtopology[OF assms(2)]]
unfolding T˙def
using union˙ordtopology[OF assms(2,4)] by auto
assume ((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)–is locally-T_2″
moreover
have ⋃T∈⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T) us-
ing TOT by auto
moreover have ⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)∈((–one-point
compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)
using top unfolding IsATopology˙def by auto
ultimately have ∃c∈Pow(⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)).
⋃T ∈ Interior(c, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪
T) ∧
(((–one-point compactification of″CoFinite ⋃T) - —⋃T″″ ∪ T) –restricted
to″ c) –is T_2″ unfolding IsLocallyT2˙def IsLocally˙def[OF top] by auto
then obtain C where C:C⊆⋃((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)
⋃T ∈ Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″)
∪ T) and T2:(((–one-point compactification of″CoFinite ⋃T) - —⋃T″″ ∪ T)
–restricted to″ C) –is T_2″
by auto
have sub:Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″)

∪ T)⊆C using topology0.Top˙2˙L1
top unfolding topology0˙def by auto
have ((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)–restricted to″C)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T))=(((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T))
using subspace˙of˙subspace[OF sub C(1)] by auto moreover
have (⋃(((–one-point compactification of″CoFinite ⋃T) - —⋃T″″ ∪ T) –restricted to″ C))⊆C unfolding RestrictedTo˙def by auto
with C(1) have (⋃(((–one-point compactification of″CoFinite ⋃T) - —⋃T″″ ∪ T) –restricted to″ C))=C unfolding RestrictedTo˙def by auto
with sub have pp:Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)∈Pow(⋃(((–one-point compactification of″CoFinite ⋃T) - —⋃T″″ ∪ T) –restricted to″ C)) by auto
ultimately have T2˙2:((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)))–is T_2″
using T2˙here[OF T2 pp] by auto
have top2:(((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)))–is a topology″
using topology0.Top˙1˙L4 top unfolding topology0˙def by auto
from C(2) pp have p1:⋃T∈⋃((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)))
unfolding RestrictedTo˙def by auto
from top topology0.Top˙2˙L2 have intOP:(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T))∈((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T unfolding topology0˙def by auto
{
fix x assume x≠⋃T x∈⋃((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)))
with p1 have ∃U∈((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T))). ∃V∈((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T))).
x∈U∧⋃T∈V∧U∩V=0 using T2˙2 unfolding isT2˙def by auto
then obtain U V where UV:U∈((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)))
V∈((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T)))
U≠0⋃T∈VU∩V=0 by auto
from UV(1) obtain UC where U=(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T))∩UCUC∈((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪ T))

```
    unfolding RestrictedTo˙def by auto
   with top intOP have Uop:U∈((–one-point compactification of″(CoFinite ⋃T))
- —⋃T″″) ∪ T unfolding IsATopology˙def by auto
    from UV(2) obtain VC where V=(Interior(C, ((–one-point compactification
of″(CoFinite ⋃T)) - —⋃T″″) ∪ T))∩VCVC∈((((–one-point compactification of″(CoFinite
⋃T)) - —⋃T″″) ∪ T))
    unfolding RestrictedTo˙def by auto
    with top intOP have V∈((–one-point compactification of″(CoFinite ⋃T)) -
—⋃T″″) ∪ T unfolding IsATopology˙def by auto
  with UV(3-5) Uop neigh˙infPoint˙dense[OF assms(2-4),of VU] union˙ordtopology[OF
assms(2,4)]
    have False unfolding T˙def by auto
  }
  then have ⋃((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″) ∪
T)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) -
—⋃T″″) ∪ T)))⊆–⋃T″
   by auto
  with p1 have ⋃((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″)
∪ T)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) -
—⋃T″″) ∪ T)))=–⋃T″
   by auto
 with top2 have –⋃T″∈((((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″)
∪ T)–restricted to″(Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) -
—⋃T″″) ∪ T)))
   unfolding IsATopology˙def by auto
 then obtain W where UT:–⋃T″=(Interior(C, ((–one-point compactification of″(CoFinite
⋃T)) - —⋃T″″) ∪ T))∩WW∈((–one-point compactification of″(CoFinite ⋃T)) -
—⋃T″″) ∪ T
   unfolding RestrictedTo˙def by auto
 from this(2) have (Interior(C, ((–one-point compactification of″(CoFinite ⋃T)) -
—⋃T″″) ∪ T))∩W∈((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″)
∪ T using intOP
   top unfolding IsATopology˙def by auto
 with UT(1) have –⋃T″∈((–one-point compactification of″(CoFinite ⋃T)) - —⋃T″″)
∪ T by auto
 then have –⋃T″∈T by auto
 with N show False by auto
qed
```

This topology, from the previous result, gives a counter-example for anti-hyperconnected implies locally-T_2.

```
theorem antiHConn˙not˙imp˙loc˙T2:
  fixes T X r
  defines T˙def:T ≡ (OrdTopology X r)
  assumes IsLinOrder(X,r) X–is dense with respect to″r
    ∃x y. x≠y∧x∈X∧y∈X
 shows ¬(((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)–is locally-T2″)
 and ((–one-point compactification of″(CoFinite (⋃T)))-—⋃T″″∪T)–is anti-″IsHConnected
 using OPComp˙cofinite˙dense˙order˙not˙loc˙T2[OF assms(2-4)] dense˙order˙infinite[OF
```

```
assms(2-4)] union˙ordtopology[OF assms(2,4)]
  topology0.COF˙comp˙antiHConn[OF topology0˙ordtopology[OF assms(2)] topology0.T2˙imp˙anti˙HConn[OF topology0˙ordtopology[OF assms(2)] order˙top˙T2[OF assms(2,4)]]]
  unfolding T˙def by auto
```

Let's prove that T_2 spaces are locally-T_2, but that there are locally-T_2 spaces which aren't T_2. In conclusion $T_2 \Rightarrow$ locally -$T_2 \Rightarrow$ anti-hyperconnected; all implications proper.

```
theorem(in topology0) T2˙imp˙loc˙T2:
  assumes T–is T₂″
  shows T–is locally-T₂″
proof-
  {
    fix x assume x∈⋃T
    {
      fix b assume b:b∈Tx∈b
      then have (T–restricted to″b)–is T₂″ using T2˙here assms by auto moreover
      from b have x∈int(b) using Top˙2˙L3 by auto
      ultimately have ∃c∈Pow(b). x∈int(c)∧(T–restricted to″c)–is T₂″ by auto
    }
    then have ∀b∈T. x∈b ⟶(∃c∈Pow(b). x∈int(c)∧(T–restricted to″c)–is T₂″) by auto
  }
  then show thesis unfolding IsLocallyT2˙def IsLocally˙def[OF topSpaceAssum] by auto
qed
```

If there is a closed singleton, then we can consider a topology that makes this point doble.

```
theorem(in topology0) doble˙point˙top:
  assumes –m″–is closed in″T
  shows (T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) –is a topology″
proof-
  {
    fix M assume M:M⊆T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″
    let MT=–V∈M. V∈T″
    let Mm=–V∈M. V∉T″
    have unm:⋃M=(⋃MT)∪(⋃Mm) by auto
    have tt:⋃MT∈T using topSpaceAssum unfolding IsATopology˙def by auto
    {
      assume Mm=0
      then have ⋃Mm=0 by auto
      with unm have ⋃M=(⋃MT) by auto
      with tt have ⋃M∈T by auto
      then have ⋃M∈T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
    }
    moreover
    {
```

assume AS:Mm≠0
then obtain V where V:V∈MV∉T by auto
with M have V∈–(U - –m″) ∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by blast
then obtain U W where U:V=(U-–m″)∪–⋃T″∪W U∈Tm∈U W∈T by auto
let U=–⟨V,W⟩∈T×T. m∈V∧ (V-–m″)∪–⋃T″∪W∈Mm″
let fU=–fst(B). B∈U″
let sU=–snd(B). B∈U″
have fU⊆TsU⊆T by auto
then have P:⋃fU∈T⋃sU∈T using topSpaceAssum unfolding IsATopology˙def by auto moreover
have ⟨U,W⟩∈U using U V by auto
then have m∈⋃fU by auto
ultimately have s:⟨⋃fU,⋃sU⟩∈–V∈T. m∈V″×T by auto
moreover have r:∀S. ∀R. S∈–V∈T. m∈V″⟶ R∈T⟶(S-–m″)∪–⋃T″∪R∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″
by auto
ultimately have (⋃fU-–m″)∪–⋃T″∪⋃sU∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
{
fix v assume v∈⋃Mm
then obtain V where v:v∈VV∈Mm by auto
then have V:V∈MV∉T by auto
with M have V∈–U - –m″ ∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by blast
then obtain U W where U:V=(U-–m″)∪–⋃T″∪W U∈Tm∈U W∈T by auto
with v(1) have v∈(U-–m″)∪–⋃T″∪W by auto
then have v∈U-–m″∨v=⋃T∨v∈W by auto
then have (v∈U∧v≠m)∨v=⋃T∨v∈W by auto
moreover from U V have ⟨U,W⟩∈U by auto
ultimately have v∈((⋃fU)-–m″)∪–⋃T″∪(⋃sU) by auto
}
then have ⋃Mm⊆((⋃fU)-–m″)∪–⋃T″∪(⋃sU) by blast moreover
{
fix v assume v:v∈((⋃fU)-–m″)∪–⋃T″∪(⋃sU)
{
assume v=⋃T
then have v∈(U-–m″)∪–⋃T″∪W by auto
with ‹⟨U,W⟩∈U› have v∈⋃Mm by auto
}
moreover
{
assume v≠⋃Tv∉⋃sU
with v have v∈((⋃fU)-–m″) by auto
then have (v∈⋃fU∧v≠m) by auto
then obtain W where (v∈W∧W∈fU∧v≠m) by auto
then have v∈(W-–m″)∪–⋃T″ W∈fU by auto
then obtain B where fst(B)=W B∈U v∈(W-–m″)∪–⋃T″ by blast
then have v∈⋃Mm by auto
}
ultimately have v∈⋃Mm by auto

}
then have ((⋃fU)-–m″)∪–⋃T″∪(⋃sU)⊆⋃Mm by auto
ultimately have ⋃Mm=((⋃fU)-–m″)∪–⋃T″∪(⋃sU) by auto
then have ⋃M=((⋃fU)-–m″)∪–⋃T″∪((⋃sU)∪(⋃MT)) using unm by auto
moreover from P tt have (⋃sU)∪(⋃MT)∈T using topSpaceAssum
union˙open[OF topSpaceAssum, of –⋃sU,⋃MT″] by auto
with s have ⟨⋃fU,(⋃sU)∪(⋃MT)⟩∈–V∈T. m∈V″×T by auto
then have ((⋃fU)-–m″)∪–⋃T″∪((⋃sU)∪(⋃MT))∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ using r
by auto
ultimately have ⋃M∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
then have ⋃M∈T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
}
ultimately
have ⋃M∈T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
}
then have ∀M∈Pow(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″). ⋃M∈T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
moreover
{
fix A B assume ass:A∈T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″B∈T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″
{
assume A:A∈T
{
assume B∈T
with A have A∩B∈T using topSpaceAssum unfolding IsATopology˙def by auto
}
moreover
{
assume B∉T
with ass(2) have B∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
then obtain U W where U:U∈Tm∈UW∈TB=(U-–m″)∪–⋃T″∪W by auto
moreover
from A mem˙not˙refl have ⋃T∉A by auto
ultimately have A∩B=A∩((U-–m″)∪W) by auto
then have eq:A∩B=(A∩(U-–m″))∪(A∩W) by auto
have ⋃T-–m″∈T using assms unfolding IsClosed˙def by auto
with U(1) have O:U∩(⋃T-–m″)∈T using topSpaceAssum unfolding IsATopology˙def
by auto
have U∩(⋃T-–m″)=U-–m″ using U(1) by auto
with O have U-–m″∈T by auto
with A have (A∩(U-–m″))∈T using topSpaceAssum unfolding IsATopology˙def
by auto
moreover
from A U(3) have A∩W∈T using topSpaceAssum unfolding IsATopology˙def
by auto

ultimately have (A∩(U-–m″))∪(A∩W)∈T using
union˙open[OF topSpaceAssum, of –A∩(U-–m″),A∩W″] by auto
with eq have A∩B∈T by auto
}
ultimately have A∩B∈T by auto
}
moreover
{
assume A∉T
with ass(1) have A:A∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
{
assume B:B∈T
from A obtain U W where U:U∈Tm∈UW∈TA=(U-–m″)∪–⋃T″∪W by auto moreover
from B mem˙not˙refl have ⋃T∉B by auto
ultimately have A∩B=((U-–m″)∪W)∩B by auto
then have eq:A∩B=((U-–m″)∩B)∪(W∩B) by auto
have ⋃T-–m″∈T using assms unfolding IsClosed˙def by auto
with U(1) have O:U∩(⋃T-–m″)∈T using topSpaceAssum unfolding IsATopology˙def
by auto
have U∩(⋃T-–m″)=U-–m″ using U(1) by auto
with O have U-–m″∈T by auto
with B have ((U-–m″)∩B)∈T using topSpaceAssum unfolding IsATopology˙def
by auto
moreover
from B U(3) have W∩B∈T using topSpaceAssum unfolding IsATopology˙def
by auto
ultimately have ((U-–m″)∩B)∪(W∩B)∈T using
union˙open[OF topSpaceAssum, of –((U-–m″)∩B),(W∩B)″] by auto
with eq have A∩B∈T by auto
}
moreover
{
assume B∉T
with ass(2) have B∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
then obtain U W where U:U∈Tm∈UW∈TB=(U-–m″)∪–⋃T″∪W by auto moreover
from A obtain UA WA where UA:UA∈Tm∈UAWA∈TA=(UA-–m″)∪–⋃T″∪WA by auto
ultimately have A∩B=(((UA-–m″)∪WA)∩((U-–m″)∪W))∪–⋃T″ by auto
then have eq:A∩B=((UA-–m″)∩(U-–m″))∪(WA∩(U-–m″))∪((UA-–m″)∩W)∪(WA∩W)∪–⋃T″ by auto
have ⋃T-–m″∈T using assms unfolding IsClosed˙def by auto
with U(1) UA(1) have O:U∩(⋃T-–m″)∈TUA∩(⋃T-–m″)∈T using topSpaceAssum unfolding IsATopology˙def
by auto
have U∩(⋃T-–m″)=U-–m″UA∩(⋃T-–m″)=UA-–m″ using U(1) UA(1) by

```
auto
        with O have OO:U-–m″∈TUA-–m″∈T by auto
        then have ((UA-–m″)∩(U-–m″))=UA∩U-–m″ by auto
        moreover
         have UA∩U∈Tm∈UA∩U using U(1,2) UA(1,2) topSpaceAssum unfolding
IsATopology˙def
         by auto
        moreover
        from OO U(3) UA(3) have TT:WA∩(U-–m″)∈T(UA-–m″)∩W∈TWA∩W∈T
using topSpaceAssum unfolding IsATopology˙def
         by auto
           from TT(2,3) have ((UA-–m″)∩W)∪(WA∩W)∈T using union˙open[OF
topSpaceAssum,
         of –(UA-–m″)∩W,WA∩W″] by auto
           with TT(1) have (WA∩(U-–m″))∪(((UA-–m″)∩W)∪(WA∩W))∈T using
union˙open[OF topSpaceAssum,
         of –WA∩(U-–m″),((UA-–m″)∩W)∪(WA∩W)″] by auto
        ultimately
     have A∩B=(UA∩U-–m″)∪–⋃T″∪((WA∩(U-–m″))∪(((UA-–m″)∩W)∪(WA∩W)))
            (WA∩(U-–m″))∪(((UA-–m″)∩W)∪(WA∩W))∈T UA∩U∈–V∈T. m∈V″
using eq by auto
        then have ∃W∈T. A∩B=(UA∩U-–m″)∪–⋃T″∪W UA∩U∈–V∈T. m∈V″ by
auto
        then have A∩B∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
      }
      ultimately
      have A∩B∈T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
    }
    ultimately have A∩B∈T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by
auto
  }
 then have ∀A∈T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″. ∀B∈T∪–(U-–m″)∪–⋃T″∪W.
⟨U,W⟩∈–V∈T. m∈V″×T″.
    A∩B∈T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by blast
  ultimately show thesis unfolding IsATopology˙def by auto
qed
```

The previous topology is defined over a set with one more point.

```
lemma(in topology0) union˙doublepoint˙top:
  assumes –m″–is closed in″T
  shows ⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)=⋃T ∪–⋃T″
proof
  {
    fix x assume x∈⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)
   then obtain R where x:x∈RR∈T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″
by blast
    {
      assume R∈T
      with x(1) have x∈⋃T by auto
```

```
    }
    moreover
    {
      assume R∉T
      with x(2) have R∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
      then obtain U W where R=(U-–m″)∪–⋃T″∪WW∈TU∈Tm∈U by auto
      with x(1) have x=⋃T∨x∈⋃T by auto
    }
    ultimately have x∈⋃T ∪–⋃T″ by auto
  }
  then show ⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)⊆⋃T ∪–⋃T″
by auto
  {
    fix x assume x∈⋃T ∪–⋃T″
    then have dis:x∈⋃T∨x=⋃T by auto
    {
      assume x∈⋃T
      then have x∈⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) by auto
    }
    moreover
    {
      assume x∉⋃T
      with dis have x=⋃T by auto
      moreover from assms have ⋃T-–m″∈Tm∈⋃T unfolding IsClosed˙def by auto
      moreover have 0∈T using empty˙open topSpaceAssum by auto
    ultimately have x∈(⋃T-–m″)∪–⋃T″∪0 (⋃T-–m″)∪–⋃T″∪0∈–(U-–m″)∪–⋃T″∪W.
⟨U,W⟩∈–V∈T. m∈V″×T″
        using union˙open[OF topSpaceAssum] by auto
    then have x∈(⋃T-–m″)∪–⋃T″∪0 (⋃T-–m″)∪–⋃T″∪0∈T ∪–(U-–m″)∪–⋃T″∪W.
⟨U,W⟩∈–V∈T. m∈V″×T″
        by auto
      then have x∈⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) by blast
    }
    ultimately have x∈⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) by
auto
  }
  then show ⋃T ∪–⋃T″⊆⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)
by auto
qed
```

In this topology, the previous topological space is an open subspace.

```
theorem(in topology0) open˙subspace˙double˙point:
  assumes –m″–is closed in″T
 shows (T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″⋃T=T
and ⋃T∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)
proof-
  have N:⋃T∉⋃T using mem˙not˙refl by auto
  {
   fix x assume x∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted
```

```
to″⋃T
  then obtain U where U:U∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)x=⋃T∩U
     unfolding RestrictedTo˙def by blast
    {
     assume U∉T
     with U(1) have  U∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
     then obtain V W where VW:U=(V-–m″)∪–⋃T″∪WV∈Tm∈VW∈T by auto
     with N U(2) have x:x=(V-–m″)∪W by auto
     have ⋃T-–m″∈T using assms unfolding IsClosed˙def by auto
     then have V∩(⋃T-–m″)∈T using VW(2) topSpaceAssum unfolding IsATopol-
ogy˙def
       by auto moreover
     have V-–m″=V∩(⋃T-–m″) using VW(2,3) by auto ultimately
     have V-–m″∈T by auto
      with VW(4) have (V-–m″)∪W∈T using union˙open[OF topSpaceAssum, of
–V-–m″,W″]
       by auto
     with x have x∈T by auto
    }
    moreover
    {
     assume A:U∈T
     with U(2) have x=U by auto
     with A have x∈T by auto
    }
    ultimately have x∈T by auto
  }
 then have (T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″⋃T⊆T
by auto
  moreover
  {
   fix x assume x:x∈T
     then have x∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) by auto
moreover
    from x have ⋃T∩x=x by auto ultimately
    have ∃M∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″). ⋃T∩M=x by
blast
     then have x∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted
to″⋃T unfolding RestrictedTo˙def
     by auto
  }
  ultimately show (T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted
to″⋃T=T by auto
  have P:⋃T∈T using topSpaceAssum unfolding IsATopology˙def by auto
  then show ⋃T∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) by auto
qed
```

The previous topology construction applied to a T_2 non-discrite space topology, gives a counter-example to: Every locally-T_2 space is T_2.

If there is a singleton which is not open, but closed; then the construction on that point is not T_2.

```
theorem(in topology0) loc˙T2˙imp˙T2˙counter˙1:
  assumes –m″∉T –m″–is closed in″T
  shows ¬((T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) –is T₂″)
proof
  assume ass:(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) –is T₂″
 then have tot1:⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)=⋃T ∪–⋃T″
using union˙doublepoint˙top
    assms(2) by auto
 have m≠⋃T using mem˙not˙refl assms(2) unfolding IsClosed˙def by auto moreover
 from ass tot1 have ∀x y. x∈⋃T ∪–⋃T″ ∧ y∈⋃T ∪–⋃T″∧x≠y ⟶ (∃𝔘∈(T∪–(U-–m″)∪–⋃T″∪W.
⟨U,W⟩∈–V∈T. m∈V″×T″).
   ∃𝔙∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″). x∈𝔘∧y∈𝔙∧𝔘∩𝔙=0)
unfolding isT2˙def by auto
  moreover
  from assms(2) have m∈⋃T ∪–⋃T″ unfolding IsClosed˙def by auto moreover
  have ⋃T∈⋃T ∪–⋃T″ by auto ultimately
 have ∃𝔘∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″). ∃𝔙∈(T∪–(U-–m″)∪–⋃T″∪W.
⟨U,W⟩∈–V∈T. m∈V″×T″). m∈𝔘∧⋃T∈𝔙∧𝔘∩𝔙=0
    by auto
 then obtain 𝔘 𝔙 where UV:𝔘∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)
     𝔙∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)m∈𝔘⋃T∈𝔙𝔘∩𝔙=0
using tot1 by blast
  then have ⋃T∉𝔘 by auto
  with UV(1) have P:𝔘∈T by auto
  {
    assume 𝔙∈T
    then have 𝔙⊆⋃T by auto
    with UV(4) have ⋃T∈⋃T using tot1 by auto
    then have False using mem˙not˙refl by auto
  }
  with UV(2) have 𝔙∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
  then obtain U W where V:𝔙=(U-–m″)∪–⋃T″∪W U∈Tm∈UW∈T by auto
  from V(2,3) P have int:U∩𝔘∈Tm∈U∩𝔘 using UV(3) topSpaceAssum
    unfolding IsATopology˙def by auto
  have (U∩𝔘-–m″)⊆𝔘 (U∩𝔘-–m″)⊆𝔙 using V(1) by auto
  then have (U∩𝔘-–m″)=0 using UV(5) by auto
  with int(2) have U∩𝔘=–m″ by auto
  with int(1) assms(1) show False by auto
qed
```

This topology is locally-T_2.

```
theorem(in topology0) loc˙T2˙imp˙T2˙counter˙2:
  assumes –m″∉T m∈⋃T T–is T₂″
  shows (T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) –is locally-T₂″
proof-
  from assms(3) have T–is T₁″ using T2˙is˙T1 by auto
  with assms(2) have mc:–m″–is closed in″T using T1˙iff˙singleton˙closed by auto
```

have N:⋃T∉⋃T using mem˙not˙refl by auto
have res:(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″⋃T=T
and P:⋃T∈T and Q:⋃T∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)
using open˙subspace˙double˙point mc
topSpaceAssum unfolding IsATopology˙def by auto
{
fix A assume ass:A∈⋃T ∪–⋃T″
{
assume A≠⋃T
with ass have A∈⋃T by auto
with Q res assms(3) have ⋃T∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T.
m∈V″×T″)∧ A∈⋃T ∧ (((T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted
to″⋃T)–is T_2″) by auto
then have ∃Z∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″). A∈Z∧(((T∪–(U-–m″)∪–⋃T″∪W.
⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″Z)–is T_2″)
by blast
}
moreover
{
assume A:A=⋃T
have ⋃T∈Tm∈⋃T0∈T using assms(2) empty˙open[OF topSpaceAssum] un-
folding IsClosed˙def using P by auto
then have (⋃T-–m″)∪–⋃T″∪0∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″
by auto
then have opp:(⋃T-–m″)∪–⋃T″∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T.
m∈V″×T″) by auto
{
fix A1 A2 assume points:A1∈(⋃T-–m″)∪–⋃T″A2∈(⋃T-–m″)∪–⋃T″A1≠A2
from points(1,2) have notm:A1≠mA2≠m using assms(2) unfolding IsClosed˙def

using mem˙not˙refl by auto
{
assume or:A1∈⋃TA2∈⋃T
with points(3) assms(3) obtain U V where UV:U∈TV∈TA1∈UA2∈V
U∩V=0 unfolding isT2˙def by blast
from UV(1,2) have U∩((⋃T-–m″)∪–⋃T″)∈(T∪–(U-–m″)∪–⋃T″∪W.
⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)
V∩((⋃T-–m″)∪–⋃T″)∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted
to″((⋃T-–m″)∪–⋃T″)
unfolding RestrictedTo˙def by auto moreover
then have U∩(⋃T-–m″)=U∩((⋃T-–m″)∪–⋃T″) V∩(⋃T-–m″)=V∩((⋃T-–m″)∪–⋃T″)
using UV(1,2) mem˙not˙refl[of ⋃T]
by auto
ultimately have opUV:U∩(⋃T-–m″)∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T.
m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)
V∩(⋃T-–m″)∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted
to″((⋃T-–m″)∪–⋃T″) by auto
moreover have U∩(⋃T-–m″)∩(V∩(⋃T-–m″))=0 using UV(5) by auto
moreover

from UV(3) or(1) notm(1) have A1∈U∩(⋃T-–m″) by auto moreover
from UV(4) or(2) notm(2) have A2∈V∩(⋃T-–m″) by auto ultimately
have ∃V. V∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)∧ A1∈U∩(⋃T-–m″)∧A2∈V∧(U∩(⋃T-–m″))∩V=0 using exI[where x=V∩(⋃T-–m″) and P=λW. W∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)∧A1∈(U∩(⋃T-–m″))∧A2∈W∧(U∩(⋃T-–m″))∩W=0]
using opUV(2) by auto
then have ∃U. U∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)∧(∃V. V∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)∧
A1∈U∧A2∈V∧U∩V=0) using exI[where x=U∩(⋃T-–m″) and P=λW. W∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)∧(∃V. V∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)∧ A1∈W∧A2∈V∧W∩V=0)]
using opUV(1) by auto
then have ∃U∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″). (∃V. V∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)∧A1∈U∧A2∈V∧U∩V=0) by blast
then have ∃U∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″). (∃V∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″). A1∈U∧A2∈V∧U∩V=0) by blast
}
moreover
{
assume A1∉⋃T
then have ig:A1=⋃T using points(1) by auto
{
assume A2∉⋃T
then have A2=⋃T using points(2) by auto
with points(3) ig have False by auto
}
then have igA2:A2∈⋃T by auto moreover
have m∈⋃T using assms(2) unfolding IsClosed˙def by auto
moreover note notm(2) assms(3) ultimately obtain U V where UV:U∈TV∈T
m∈UA2∈VU∩V=0 unfolding isT2˙def by blast
from UV(1,3) have U∈–W∈T. m∈W″ by auto moreover
have 0∈T using empty˙open topSpaceAssum by auto ultimately
have (U-–m″)∪–⋃T″∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
then have Uop:(U-–m″)∪–⋃T″∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) by auto
from UV(2) have Vop:V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) by auto
from UV(1-3,5) have sub:V⊆(⋃T-–m″)∪–⋃T″ ((U-–m″)∪–⋃T″)⊆(⋃T-–m″)∪–⋃T″ by auto
from sub(1) have V=((⋃T-–m″)∪–⋃T″)∩V by auto
then have VV:V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″) unfolding RestrictedTo˙def
using Vop by blast moreover

from sub(2) have ((U-–m″)∪–⋃T″)=((⋃T-–m″)∪–⋃T″)∩((U-–m″)∪–⋃T″) by auto
then have UU:((U-–m″)∪–⋃T″)∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″) unfolding RestrictedTo˙def
using Uop by blast moreover
from UV(2) have ((U-–m″)∪–⋃T″)∩V=(U-–m″)∩V using mem˙not˙refl by auto
then have ((U-–m″)∪–⋃T″)∩V=0 using UV(5) by auto
with UV(4) VV ig igA2 have ∃V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″).
A1∈(U-–m″)∪–⋃T″∧A2∈V∧((U-–m″)∪–⋃T″)∩V=0 by auto
with UU ig have ∃U. U∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)∧ (∃V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″).
A1∈U∧A2∈V∧U∩V=0) using exI[where x=((U-–m″)∪–⋃T″) and P=λU. U∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)∧ (∃V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″).
A1∈U∧A2∈V∧U∩V=0)] by auto
then have ∃U∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″). (∃V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″).
A1∈U∧A2∈V∧U∩V=0) by blast
}
moreover
{
assume A2∉⋃T
then have ig:A2=⋃T using points(2) by auto
{
assume A1∉⋃T
then have A1=⋃T using points(1) by auto
with points(3) ig have False by auto
}
then have igA2:A1∈⋃T by auto moreover
have m∈⋃T using assms(2) unfolding IsClosed˙def by auto
moreover note notm(1) assms(3) ultimately obtain U V where UV:U∈TV∈T m∈UA1∈VU∩V=0 unfolding isT2˙def by blast
from UV(1,3) have U∈–W∈T. m∈W″ by auto moreover
have 0∈T using empty˙open topSpaceAssum by auto ultimately
have (U-–m″)∪–⋃T″∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
then have Uop:(U-–m″)∪–⋃T″∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) by auto
from UV(2) have Vop:V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) by auto
from UV(1-3,5) have sub:V⊆(⋃T-–m″)∪–⋃T″ ((U-–m″)∪–⋃T″)⊆(⋃T-–m″)∪–⋃T″ by auto
from sub(1) have V=((⋃T-–m″)∪–⋃T″)∩V by auto
then have VV:V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″) unfolding RestrictedTo˙def

using Vop by blast moreover
from sub(2) have ((U-–m″)∪–⋃T″)=((⋃T-–m″)∪–⋃T″)∩((U-–m″)∪–⋃T″) by auto
then have UU:((U-–m″)∪–⋃T″)∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″) unfolding RestrictedTo˙def
using Uop by blast moreover
from UV(2) have V∩((U-–m″)∪–⋃T″)=V∩(U-–m″) using mem˙not˙refl by auto
then have V∩((U-–m″)∪–⋃T″)=0 using UV(5) by auto
with UU UV(4) ig igA2 have ∃U∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″).
A1∈V∧A2∈U∧V∩U=0 by auto
with VV igA2 have ∃U. U∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)∧ (∃V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″).
A1∈U∧A2∈V∧U∩V=0) using exI[where x=V and P=λU. U∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)∧ (∃V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″).
A1∈U∧A2∈V∧U∩V=0)] by auto
then have ∃U∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″). (∃V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″).
A1∈U∧A2∈V∧U∩V=0) by blast
}
ultimately have ∃U∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″). (∃V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″).
A1∈U∧A2∈V∧U∩V=0) by blast
}
then have ∀A1∈(⋃T-–m″)∪–⋃T″. ∀A2∈(⋃T-–m″)∪–⋃T″. A1≠A2 ⟶ (∃U∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″). (∃V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″).
A1∈U∧A2∈V∧U∩V=0)) by auto moreover
have ⋃((T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″))=(⋃(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″))∩((⋃T-–m″)∪–⋃T″) unfolding RestrictedTo˙def by auto
then have ⋃((T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″))=(⋃T ∪–⋃T″)∩((⋃T-–m″)∪–⋃T″) using union˙doublepoint˙top mc by auto
then have ⋃((T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″))=(⋃T-–m″)∪–⋃T″ by auto
ultimately have ∀A1∈⋃((T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)). ∀A2∈⋃((T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″)). A1≠A2 ⟶ (∃U∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″). (∃V∈(T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″).
A1∈U∧A2∈V∧U∩V=0)) by auto
then have ((T ∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″((⋃T-–m″)∪–⋃T″))–is T_2″ unfolding isT2˙def

by force
with opp A have ∃Z∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″). A∈Z∧(((T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″Z)–is T_2″)
by blast
}
ultimately
have ∃Z∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″). A∈Z∧(((T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″Z)–is T_2″)
by blast
}
then have ∀A∈⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″). ∃Z∈T ∪ –U - –m″ ∪ –⋃T″ ∪ W . ⟨U,W⟩ ∈ –V ∈ T . m ∈ V″ × T″.
A ∈ Z ∧ ((T ∪ –U - –m″ ∪ –⋃T″ ∪ W . ⟨U,W⟩ ∈ –V ∈ T . m ∈ V″ × T″) –restricted to″ Z) –is T_2″
using union˙doublepoint˙top mc by auto
with topology0.loc˙T2 show (T ∪ –U - –m″ ∪ –⋃T″ ∪ W . ⟨U,W⟩ ∈ –V ∈ T . m ∈ V″ × T″)–is locally-T_2″
unfolding topology0˙def using doble˙point˙top mc by auto
qed

There can be considered many more local properties, which; as happens with locally-T_2; can distinguish between spaces other properties cannot.

end

68 Properties in Topology 3

theory Topology˙ZF˙properties˙3 imports Topology˙ZF˙7 Finite˙ZF˙1 Topology˙ZF˙1b Topology˙ZF˙9
Topology˙ZF˙properties˙2 FinOrd˙ZF
begin

This theory file deals with more topological properties and the relation with the previous ones in other theory files.

68.1 More anti-properties

In this section we study more anti-properties.

68.2 First examples

A first example of an anti-compact space is the discrete space.

lemma pow˙compact˙imp˙finite:
assumes B–is compact in″Pow(A)
shows Finite(B)
proof-
from assms have B:B⊆A ∀M∈Pow(Pow(A)). B⊆⋃M ⟶(∃N∈FinPow(M). B⊆⋃N)

```
    unfolding IsCompact˙def by auto
  from B(1) have —x″. x∈B″∈Pow(Pow(A)) B⊆⋃—x″. x∈B″ by auto
  with B(2) have ∃N∈FinPow(—x″. x∈B″). B⊆⋃N by auto
  then obtain N where N∈FinPow(—x″. x∈B″) B⊆⋃N by auto
  then have Finite(N) N⊆—x″. x∈B″ B⊆⋃N unfolding FinPow˙def by auto
  then have Finite(N) ∀b∈N. Finite(b) B⊆⋃N by auto
  then have B⊆⋃N Finite(⋃N) using Finite˙Union[of N] by auto
  then show Finite(B) using subset˙Finite by auto
qed

theorem pow˙anti˙compact:
  shows Pow(A)–is anti-compact″
proof-
  {
   fix B assume as:B⊆⋃Pow(A) (⋃(Pow(A)–restricted to″B))–is compact in″(Pow(A)–restricted
to″B)
    then have sub:B⊆A by auto
    then have Pow(B)=Pow(A)–restricted to″B unfolding RestrictedTo˙def by blast
    with as(2) have (⋃Pow(B))–is compact in″Pow(B) by auto
    then have B–is compact in″Pow(B) by auto
    then have Finite(B) using pow˙compact˙imp˙finite by auto
     then have B–is in the spectrum of″(λT. (⋃T)–is compact in″T) using com-
pact˙spectrum by auto
  }
  then show thesis unfolding IsAntiComp˙def antiProperty˙def by auto
qed
```

In a previous file, Topology˙ZF˙5.thy, we proved that the spectrum of the lindelf property depends on the axiom of countable choice on subsets of the power set of the natural number.

In this context, the examples depend on wether this choice principle holds or not. This is the reason that the examples of anti-lindeloef topologies are left for the next section.

68.3 Structural results

We first differenciate the spectrum of the lindeloef property depending on some axiom of choice.

```
lemma lindeloef˙spec1:
  assumes –the axiom of″ nat –choice holds for subsets″(Pow(nat))
  shows (A –is in the spectrum of″ (λT. ((⋃T)–is lindeloef in″T))) ⟷ (A≲nat)
  using compactK˙spectrum[OF assms Card˙nat] unfolding IsLindeloef˙def.

lemma lindeloef˙spec2:
  assumes ¬(–the axiom of″ nat –choice holds for subsets″(Pow(nat)))
  shows (A –is in the spectrum of″ (λT. ((⋃T)–is lindeloef in″T))) ⟷ Finite(A)
proof
```

assume Finite(A)
then have A:A–is in the spectrum of″ (λT. ((⋃T)–is compact in″T)) using compact˙spectrum by auto
have s:nat≲csucc(nat) using le˙imp˙lesspoll[OF Card˙csucc[OF Ord˙nat]] lt˙csucc[OF Ord˙nat] le˙iff by auto
{
fix T assume T–is a topology″ (⋃T)–is compact in″T
then have (⋃T)–is compact of cardinal″nat–in″T using Compact˙is˙card˙nat by auto
then have (⋃T)–is compact of cardinal″csucc(nat)–in″T using s compact˙greater˙card Card˙csucc[OF Ord˙nat] by auto
then have (⋃T)–is lindeloef in″T unfolding IsLindeloef˙def by auto
}
then have ∀T. T–is a topology″ ⟶ ((⋃T)–is compact in″T) ⟶ ((⋃T)–is lindeloef in″T) by auto
with A show A –is in the spectrum of″ (λT. ((⋃T)–is lindeloef in″T)) using P˙imp˙Q˙spec˙inv[
where Q=λT. ((⋃T)–is compact in″T) and P=λT. ((⋃T)–is lindeloef in″T)] by auto
next
assume A:A –is in the spectrum of″ (λT. ((⋃T)–is lindeloef in″T))
then have reg:∀T. T–is a topology″∧⋃T≈A ⟶ ((⋃T)–is compact of cardinal″ csucc(nat)–in″T) using Spec˙def
unfolding IsLindeloef˙def by auto
then have A–is compact of cardinal″ csucc(nat) –in″ Pow(A) using Pow˙is˙top[of A] by auto
then have ∀M∈Pow(Pow(A)). A⊆⋃M ⟶ (∃N∈Pow(M). A⊆⋃N ∧ N≺csucc(nat)) unfolding IsCompactOfCard˙def by auto
moreover
have —x″. x∈A″∈Pow(Pow(A)) by auto
moreover
have A=⋃—x″. x∈A″ by auto
ultimately have ∃N∈Pow(—x″. x∈A″). A⊆⋃N ∧ N≺csucc(nat) by auto
then obtain N where N∈Pow(—x″. x∈A″) A⊆⋃N N≺csucc(nat) by auto
then have N⊆—x″. x∈A″ N≺csucc(nat) A⊆⋃N using FinPow˙def by auto
{
fix t
assume t∈—x″. x∈A″
then obtain x where x∈At=–x″ by auto
with ‹A⊆⋃N› have x∈⋃N by auto
then obtain B where B∈Nx∈B by auto
with ‹N⊆—x″. x∈A″› have B=–x″ by auto
with ‹t=–x″›‹B∈N› have t∈N by auto
}
with ‹N⊆—x″. x∈A″› have N=—x″. x∈A″ by auto
let B=–⟨x,–x″⟩. x∈A″
from ‹N=—x″. x∈A″› have B:A→ N unfolding Pi˙def function˙def by auto
with ‹N=—x″. x∈A″› have B:inj(A,N) unfolding inj˙def using apply˙equality by auto

```
  then have A≲N using lepoll˙def by auto
  with ‹N≺csucc(nat)› have A≺csucc(nat) using lesspoll˙trans1 by auto
  then have A≲nat using Card˙less˙csucc˙eq˙le Card˙nat by auto
  then have A≺nat∨A≈nat using lepoll˙iff˙leqpoll by auto moreover
  {
    assume A≈nat
    then have nat≈A using eqpoll˙sym by auto
     with A have nat –is in the spectrum of″ (λT. ((⋃T)–is lindeloef in″T)) using
equipollent˙spect[
      where P=(λT. ((⋃T)–is lindeloef in″T))] by auto
    moreover
    have Pow(nat)–is a topology″ using Pow˙is˙top by auto
    moreover
    have ⋃Pow(nat)=nat by auto
    then have ⋃Pow(nat)≈nat using eqpoll˙refl by auto
    ultimately
     have nat –is compact of cardinal″ csucc(nat)–in″Pow(nat) using Spec˙def un-
folding IsLindeloef˙def by auto
     then have False using Q˙disc˙comp˙csuccQ˙eq˙Q˙choice˙csuccQ[OF InfCard˙nat]
assms by auto
  }
  ultimately have A≺nat by auto
  then show Finite(A) using lesspoll˙nat˙is˙Finite by auto
qed
```

If the axiom of countable choice on subsets of the pow of the natural numbers doesn't hold, then anti-lindeloef spaces are anti-compact.

```
theorem(in topology0) no˙choice˙imp˙anti˙lindeloef˙is˙anti˙comp:
 assumes ¬(–the axiom of″ nat –choice holds for subsets″(Pow(nat)) ) T–is anti-lindeloef″
  shows T–is anti-compact″
proof-
 have s:nat≲csucc(nat) using le˙imp˙lesspoll[OF Card˙csucc[OF Ord˙nat]] lt˙csucc[OF
Ord˙nat] le˙iff by auto
  {
    fix T assume T–is a topology″ (⋃T)–is compact in″T
    then have (⋃T)–is compact of cardinal″nat–in″T using Compact˙is˙card˙nat by
auto
   then have (⋃T)–is compact of cardinal″csucc(nat)–in″T using s compact˙greater˙card
Card˙csucc[OF Ord˙nat] by auto
    then have (⋃T)–is lindeloef in″T unfolding IsLindeloef˙def by auto
  }
   then have ∀T. T–is a topology″ ⟶ ((⋃T)–is compact in″T) ⟶ ((⋃T)–is
lindeloef in″T) by auto
 from eq˙spect˙rev˙imp˙anti[OF this] lindeloef˙spec2[OF assms(1)] compact˙spectrum
    show thesis using assms(2) unfolding IsAntiLin˙def IsAntiComp˙def by auto
qed
```

If the axiom of countable choice holds for subsets of the power set of the natural numbers, then there exists a topological space that is anti-lindeloef

but no anti-compact.

```
theorem no˙choice˙imp˙anti˙lindeloef˙is˙anti˙comp:
  assumes (–the axiom of" nat –choice holds for subsets"(Pow(nat)))
  shows (–one-point compactification of"Pow(nat))–is anti-lindeloef"
proof-
 have t:⋃(–one-point compactification of"Pow(nat))=–nat"∪nat using topology0.op˙compact˙total
    unfolding topology0˙def using Pow˙is˙top by auto
  have –nat"≈1 using singleton˙eqpoll˙1 by auto
  then have –nat"≺nat using n˙lesspoll˙nat eq˙lesspoll˙trans by auto moreover
 have s:nat≺csucc(nat) using lt˙Card˙imp˙lesspoll[OF Card˙csucc] lt˙csucc[OF Ord˙nat]
by auto
  ultimately have –nat"≺csucc(nat) using lesspoll˙trans by blast
 with s have –nat"∪nat≺csucc(nat) using less˙less˙imp˙un˙less[OF ˙ ˙ InfCard˙csucc[OF
InfCard˙nat]]
    by auto
  then have –nat"∪nat≲nat using Card˙less˙csucc˙eq˙le[OF Card˙nat] by auto
  with t have r:⋃(–one-point compactification of"Pow(nat))≲nat by auto
  {
   fix A assume A:A∈Pow(⋃(–one-point compactification of"Pow(nat))) (⋃((–one-point
compactification of"Pow(nat))–restricted to"A))–is lindeloef in"((–one-point com-
pactification of"Pow(nat))–restricted to"A)
    from A(1) have A⊆⋃(–one-point compactification of"Pow(nat)) by auto
    with r have A≲nat using subset˙imp˙lepoll lepoll˙trans by blast
    then have A–is in the spectrum of"(λT. ((⋃T)–is lindeloef in"T)) using assms
      lindeloef˙spec1 by auto
  }
  then show thesis unfolding IsAntiLin˙def antiProperty˙def by auto
qed

theorem op˙comp˙pow˙nat˙no˙anti˙comp:
  shows ¬((–one-point compactification of"Pow(nat))–is anti-compact")
proof
  let T=(–one-point compactification of"Pow(nat))–restricted to"(–nat" ∪ nat)
  assume antiComp:(–one-point compactification of"Pow(nat))–is anti-compact"
  have (–nat" ∪ nat)–is compact in"(–one-point compactification of"Pow(nat))
     using topology0.compact˙op[of Pow(nat)] Pow˙is˙top[of nat] unfolding topol-
ogy0˙def
    by auto
  then have (–nat" ∪ nat)–is compact in"T using compact˙imp˙compact˙subspace
Compact˙is˙card˙nat by auto
 moreover have ⋃T=(⋃(–one-point compactification of"Pow(nat)))∩(–nat" ∪ nat)
unfolding RestrictedTo˙def by auto
  then have ⋃T=–nat" ∪ nat using topology0.op˙compact˙total unfolding topol-
ogy0˙def
    using Pow˙is˙top by auto
  ultimately have (⋃T)–is compact in"T by auto
  with antiComp have (–nat" ∪ nat)–is in the spectrum of"(λT. (⋃T)–is compact
in"T) unfolding IsAntiComp˙def
   antiProperty˙def using topology0.op˙compact˙total unfolding topology0˙def using
```

```
Pow˙is˙top by auto
  then have Finite(–nat˝ ∪ nat) using compact˙spectrum by auto
  then have Finite(nat) using subset˙Finite by auto
  then show False using nat˙not˙Finite by auto
qed
```

In coclusion, we reached another equivalence of this choice principle.

The axiom of countable choice holds for subsets of the power set of the natural numbers if and only if there exists a topological space which is anti-lindeloef but not anti-compact; this space can be chosen as the one-point compactification of the discrete topology on ℕ.

```
theorem acc˙pow˙nat˙equiv1:
 shows (–the axiom of˝ nat –choice holds for subsets˝(Pow(nat))) ⟷ ((–one-point
compactification of˝Pow(nat))–is anti-lindeloef˝)
  using op˙comp˙pow˙nat˙no˙anti˙comp no˙choice˙imp˙anti˙lindeloef˙is˙anti˙comp
  topology0.no˙choice˙imp˙anti˙lindeloef˙is˙anti˙comp topology0.op˙comp˙is˙top
  Pow˙is˙top[of nat] unfolding topology0˙def by auto

theorem acc˙pow˙nat˙equiv2:
  shows (–the axiom of˝ nat –choice holds for subsets˝(Pow(nat))) ⟷ (∃ T. T–is
a topology˝
  ∧ (T–is anti-lindeloef˝) ∧ ¬(T–is anti-compact˝))
  using op˙comp˙pow˙nat˙no˙anti˙comp no˙choice˙imp˙anti˙lindeloef˙is˙anti˙comp
  topology0.no˙choice˙imp˙anti˙lindeloef˙is˙anti˙comp topology0.op˙comp˙is˙top
  Pow˙is˙top[of nat] unfolding topology0˙def by auto
```

In the file Topology˙ZF˙properties.thy, it is proven that ℕ is lindeloef if and only if the axiom of countable choice holds for subsets of $Pow(\mathbb{N})$. Now we check that, in ZF, this space is always anti-lindeloef.

```
theorem nat˙anti˙lindeloef:
  shows Pow(nat)–is anti-lindeloef˝
proof-
  {
   fix A assume A:A∈Pow(⋃Pow(nat)) (⋃(Pow(nat)–restricted to˝A))–is lindeloef
in˝(Pow(nat)–restricted to˝A)
    from A(1) have A⊆nat by auto
     then have Pow(nat)–restricted to˝A=Pow(A) unfolding RestrictedTo˙def by
blast
    with A(2) have lin:A–is lindeloef in˝Pow(A) using subset˙imp˙lepoll by auto
    {
      fix T assume T:T–is a topology˝ ⋃T≈A
      then have A≈⋃T using eqpoll˙sym by auto
      then obtain f where f:f∈bij(A,⋃T) unfolding eqpoll˙def by auto
      then have f∈surj(A,⋃T) unfolding bij˙def by auto
      moreover then have IsContinuous(Pow(A),T,f) unfolding IsContinuous˙def
        surj˙def using func1˙1˙L3 by blast
      moreover have two˙top˙spaces0(Pow(A),T,f) unfolding two˙top˙spaces0˙def
        using f T(1) Pow˙is˙top unfolding bij˙def inj˙def by auto
```

```
    ultimately have (⋃T)–is lindeloef in″T using two˙top˙spaces0.cont˙image˙com
      lin unfolding IsLindeloef˙def by auto
  }
   then have A–is in the spectrum of″ (λT. ((⋃T)–is lindeloef in″T)) unfolding
Spec˙def by auto
 }
 then show thesis unfolding IsAntiLin˙def antiProperty˙def by auto
qed
```

This result is interesting because depending on the different axioms we add to ZF, it means two different things:

- Every subspace of $\mathbb{N}$ is Lindeloef.
- Only the compact subspaces of $\mathbb{N}$ are Lindeloef.

Now, we could wonder if the class of compact spaces and the class of lindeloef spaces being equal is consistent in ZF. Let's find a topological space which is lindeloef and no compact without assuming any axiom of choice or any negation of one. This will prove that the class of lindeloef spaces and the class of compact spaces cannot be equal in any model of ZF.

```
theorem lord˙nat:
 shows (LOrdTopology nat Le)=–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ ∪–0″
proof-
 {
  fix U assume U:U⊆–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ U≠0
  {
    assume nat∈U
    with U have ⋃U=nat unfolding LeftRayX˙def by auto
    then have ⋃U∈–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″∪–0″ by auto
  }
  moreover
  {
    assume nat∉U
    with U have UU:U⊆–LeftRayX(nat,Le,n). n∈nat″∪–0″ by auto
    {
      assume A:∃i. i∈nat∧ ⋃U⊆ LeftRayX(nat,Le,i)
      let M=μ i. i∈nat ∧ ⋃U⊆ LeftRayX(nat,Le,i)
          from A have M:M∈nat ⋃U⊆ LeftRayX(nat,Le,M) using LeastI[OF ˙
nat˙into˙Ord, where P=λi. i∈nat ∧ ⋃U⊆ LeftRayX(nat,Le,i)]
        by auto
      {
        fix y assume V:y∈LeftRayX(nat,Le,M)
        then have y:y∈nat unfolding LeftRayX˙def by auto
        {
          assume ∀V∈U. y∉V
          then have ∀m∈–n∈nat. LeftRayX(nat,Le,n)∈U″. y∉LeftRayX(nat,Le,m)
using UU by auto
```

then have ∀m∈–n∈nat. LeftRayX(nat,Le,n)∈U″. ⟨y,m⟩∉Le∨y=m unfolding LeftRayX˙def using y
by auto
then have RR:∀m∈–n∈nat. LeftRayX(nat,Le,n)∈U″. ⟨m,y⟩∈Le using Le˙directs˙nat(1) y unfolding IsLinOrder˙def IsTotal˙def by blast
{
fix rr V assume rr∈⋃U
then obtain V where V:V∈U rr∈V by auto
with UU obtain m where m:V=LeftRayX(nat,Le,m) m∈nat by auto
with V(1) RR have a:⟨m,y⟩∈Le by auto
from V(2) m(1) have b:⟨rr,m⟩∈Le rr∈nat-–m″ unfolding LeftRayX˙def by auto
from a b(1) have ⟨rr,y⟩∈Le using Le˙directs˙nat(1) unfolding IsLinOrder˙def trans˙def by blast moreover
{
assume rr=y
with a b have False using Le˙directs˙nat(1) unfolding IsLinOrder˙def antisym˙def by blast
}
ultimately have rr∈LeftRayX(nat,Le,y) unfolding LeftRayX˙def using b(2) by auto
}
then have ⋃U⊆LeftRayX(nat,Le,y) by auto
with y M(1) have ⟨M,y⟩∈Le using Least˙le by auto
with V have False unfolding LeftRayX˙def using Le˙directs˙nat(1) unfolding IsLinOrder˙def antisym˙def by blast
}
then have y∈⋃U by auto
}
then have LeftRayX(nat,Le,M)⊆⋃U by auto
with M(2) have ⋃U=LeftRayX(nat,Le,M) by auto
with M(1) have ⋃U∈–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ by auto
}
moreover
{
assume ¬(∃i. i ∈ nat ∧ ⋃U ⊆ LeftRayX(nat, Le, i))
then have A:∀i. i∈nat ⟶ ¬(⋃U ⊆ LeftRayX(nat, Le, i)) by auto
{
fix i assume i:i∈nat
with A have AA:¬(⋃U ⊆ LeftRayX(nat, Le, i)) by auto
{
assume i∉⋃U
then have ∀V∈U. i∉V by auto
then have ∀m∈–n∈nat. LeftRayX(nat, Le, n)∈U″. i∉LeftRayX(nat, Le, m) by auto
with i have ∀m∈–n∈nat. LeftRayX(nat, Le, n)∈U″. ⟨i,m⟩∉Le∨i=m unfolding LeftRayX˙def by auto
with i have ∀m∈–n∈nat. LeftRayX(nat, Le, n)∈U″. ¬(i≤m)∨i=m unfolding Le˙def by auto

then have ∀m∈–n∈nat. LeftRayX(nat, Le, n)∈U″. m¡i∨m=i using not˙le˙iff˙lt[OF nat˙into˙Ord[OF i]
nat˙into˙Ord] by auto
then have M:∀m∈–n∈nat. LeftRayX(nat, Le, n)∈U″. m≤i using le˙iff nat˙into˙Ord[OF i] by auto
{
fix s assume s∈⋃U
then obtain n where n:n∈nat s∈LeftRayX(nat, Le, n) LeftRayX(nat, Le, n)∈U
using UU by auto
with M have ni:n≤i by auto
from n(2) have sn:s≤n s≠n unfolding LeftRayX˙def by auto
then have s≤i s≠i using le˙trans[OF sn(1) ni] le˙anti˙sym[OF sn(1)] ni by auto
then have s∈LeftRayX(nat, Le, i) using i le˙in˙nat unfolding LeftRayX˙def by auto
}
with AA have False by auto
}
then have i∈⋃U by auto
}
then have nat⊆⋃U by auto
then have ⋃U=nat using UU unfolding LeftRayX˙def by auto
then have ⋃U∈–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ ∪–0″ by auto
}
ultimately have ⋃U∈–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ ∪–0″ by auto
}
ultimately have ⋃U∈–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ ∪–0″ by auto
}
moreover
{
fix U assume U=0
then have ⋃U∈–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ ∪–0″ by auto
}
ultimately have ∀U. U⊆–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ ⟶ ⋃U∈–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ ∪–0″
by auto
then have –LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ ∪–0″=–⋃U. U∈Pow(–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″)″ by blast
then show thesis using LOrdtopology˙ROrdtopology˙are˙topologies(2)[OF Le˙directs˙nat(1)]
unfolding IsAbaseFor˙def by auto
qed

lemma countable˙lord˙nat:
shows –LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ ∪–0″≺csucc(nat)
proof-
{
fix e
have –e″≈1 using singleton˙eqpoll˙1 by auto

then have –e″≺nat using n˙lesspoll˙nat eq˙lesspoll˙trans by auto moreover
have s:nat≺csucc(nat) using lt˙Card˙imp˙lesspoll[OF Card˙csucc] lt˙csucc[OF Ord˙nat] by auto
ultimately have –e″≺csucc(nat) using lesspoll˙trans by blast
}
then have –nat″ ∪–0″≺csucc(nat) using less˙less˙imp˙un˙less[OF ˙ ˙ InfCard˙csucc[OF InfCard˙nat], of –nat″ –0″]
by auto moreover
let FF=–⟨n,LeftRayX(nat,Le,n)⟩. n∈nat″
have ff:FF:nat→–LeftRayX(nat,Le,n). n∈nat″ unfolding Pi˙def domain˙def function˙def by auto
then have su:FF∈surj(nat,–LeftRayX(nat,Le,n). n∈nat″) unfolding surj˙def using apply˙equality[
OF ˙ ff] by auto
then have –LeftRayX(nat,Le,n). n∈nat″≲nat using surj˙fun˙inv˙2[OF su lepoll˙refl[of nat]] Ord˙nat
by auto
then have –LeftRayX(nat,Le,n). n∈nat″≺csucc(nat) using Card˙less˙csucc˙eq˙le[OF Card˙nat] by auto
ultimately have –LeftRayX(nat, Le, n) . n ∈ nat″ ∪ (–nat″ ∪ –0″) ≺ csucc(nat) using less˙less˙imp˙un˙less[OF ˙ ˙ InfCard˙csucc[OF InfCard˙nat]] by auto
moreover have –LeftRayX(nat, Le, n) . n ∈ nat″ ∪ (–nat″ ∪ –0″)=–LeftRayX(nat, Le, n) . n ∈ nat″ ∪ –nat″ ∪ –0″ by auto
ultimately show thesis by auto
qed

corollary lindelof˙lord˙nat:
shows nat–is lindeloef in″(LOrdTopology nat Le)
unfolding IsLindeloef˙def using countable˙lord˙nat lord˙nat card˙top˙comp[OF Card˙csucc[OF Ord˙nat]]
union˙lordtopology˙rordtopology(1)[OF Le˙directs˙nat(1)] by auto

theorem not˙comp˙lord˙nat:
shows ¬(nat–is compact in″(LOrdTopology nat Le))
proof
assume nat–is compact in″(LOrdTopology nat Le)
with lord˙nat have nat–is compact in″(–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ ∪–0″) by auto
then have ∀M∈Pow(–LeftRayX(nat,Le,n). n∈nat″ ∪–nat″ ∪–0″). nat⊆⋃M ⟶ (∃N∈FinPow(M). nat⊆⋃N)
unfolding IsCompact˙def by auto moreover
{
fix n assume n:n∈nat
then have n¡succ(n) by auto
then have ⟨n,succ(n)⟩∈Le n≠succ(n) using n nat˙succ˙iff by auto
then have n∈LeftRayX(nat,Le,succ(n)) unfolding LeftRayX˙def using n by auto
then have n∈⋃(–LeftRayX(nat,Le,n). n∈nat″) using n nat˙succ˙iff by auto
}
ultimately have ∃N∈FinPow(–LeftRayX(nat,Le,n). n∈nat″). nat⊆⋃N by blast

then obtain N where N∈FinPow(–LeftRayX(nat,Le,n). n∈nat″) nat⊆⋃N by auto
then have N:N⊆–LeftRayX(nat,Le,n). n∈nat″ Finite(N) nat⊆⋃N unfolding FinPow˙def by auto
let F=–⟨n,LeftRayX(nat,Le,n)⟩. n∈–m∈nat. LeftRayX(nat,Le,m)∈N″″
have ff:F:–m∈nat. LeftRayX(nat,Le,m)∈N″ → N unfolding Pi˙def function˙def by auto
then have F∈surj(–m∈nat. LeftRayX(nat,Le,m)∈N″, N) unfolding surj˙def using N(1) apply˙equality[
OF ˙ ff] by blast moreover
{
fix x y assume xyF:x∈–m∈nat. LeftRayX(nat,Le,m)∈N″ y∈–m∈nat. LeftRayX(nat,Le,m)∈N″ Fx=Fy
then have Fx=LeftRayX(nat,Le,x) Fy=LeftRayX(nat,Le,y) using apply˙equality[
OF ˙ ff] by auto
with xyF(3) have lxy:LeftRayX(nat,Le,x)=LeftRayX(nat,Le,y) by auto
{
fix r assume r¡x
then have r≤x r≠x using leI by auto
with xyF(1) have r∈LeftRayX(nat,Le,x) unfolding LeftRayX˙def using le˙in˙nat by auto
then have r∈LeftRayX(nat,Le,y) using lxy by auto
then have r≤y r≠y unfolding LeftRayX˙def by auto
then have r¡y using le˙iff by auto
}
then have ∀r. r¡x ⟶ r¡y by auto
then have r:¬(y¡x) by auto
{
fix r assume r¡y
then have r≤y r≠y using leI by auto
with xyF(2) have r∈LeftRayX(nat,Le,y) unfolding LeftRayX˙def using le˙in˙nat by auto
then have r∈LeftRayX(nat,Le,x) using lxy by auto
then have r≤x r≠x unfolding LeftRayX˙def by auto
then have r¡x using le˙iff by auto
}
then have ¬(x¡y) by auto
with r have x=y using not˙lt˙iff˙le[OF nat˙into˙Ord nat˙into˙Ord] xyF(1,2)
le˙anti˙sym by auto
}
then have F∈inj(–m∈nat. LeftRayX(nat,Le,m)∈N″, N) unfolding inj˙def using ff by auto
ultimately have F∈bij(–m∈nat. LeftRayX(nat,Le,m)∈N″, N) unfolding bij˙def by auto
then have –m∈nat. LeftRayX(nat,Le,m)∈N″≈ N unfolding eqpoll˙def by auto
with N(2) have fin:Finite(–m∈nat. LeftRayX(nat,Le,m)∈N″) using lepoll˙Finite eqpoll˙imp˙lepoll
by auto
from N(3) have N≠0 by auto
then have nE:–m∈nat. LeftRayX(nat,Le,m)∈N″≠0 using N(1) by auto

```
  let M=Maximum(Le,–m∈nat. LeftRayX(nat,Le,m)∈N″)
   have M:M∈nat LeftRayX(nat,Le,M)∈N ∀𝔯∈–m∈nat. LeftRayX(nat,Le,m)∈N″.
⟨𝔯,M⟩∈Le using fin linord˙max˙props(1,3)[OF Le˙directs˙nat(1) ˙ nE]
    unfolding FinPow˙def by auto
  {
    fix V 𝔯 assume V:V∈N 𝔯∈V
   then obtain m where m:V=LeftRayX(nat,Le,m) LeftRayX(nat,Le,m)∈N m∈nat
using N(1) by auto
    with V(2) have xx:⟨𝔯,m⟩∈Le 𝔯≠m unfolding LeftRayX˙def by auto
    from m(2,3) have m∈–m∈nat. LeftRayX(nat,Le,m)∈N″ by auto
    then have mM:⟨m,M⟩∈Le using M(3) by auto
    with xx(1) have ⟨𝔯,M⟩∈Le using le˙trans unfolding Le˙def by auto
    moreover
    {
      assume 𝔯=M
      with xx mM have False using le˙anti˙sym by auto
    }
    ultimately have 𝔯∈LeftRayX(nat,Le,M) unfolding LeftRayX˙def by auto
  }
  then have ⋃N⊆LeftRayX(nat,Le,M) by auto
  with M(2) have ⋃N=LeftRayX(nat,Le,M) by auto
  with N(3) have nat⊆LeftRayX(nat,Le,M) by auto
  moreover from M(1) have succ(M)∈nat using nat˙succI by auto
  ultimately have succ(M)∈LeftRayX(nat,Le,M) by auto
  then have ⟨succ(M),M⟩∈Le unfolding LeftRayX˙def by blast
  then show False by auto
qed
```

68.4 More Separation properties

In this section we study more separation properties.

68.5 Definitions

We start with a property that has already appeared in Topology˙ZF˙1b.thy. A KC-space is a space where compact sets are closed.

```
definition
  IsKC (˙ –is KC″) where
  T–is KC″ ≡ ∀A∈Pow(⋃T). A–is compact in″T ⟶ A–is closed in″T
```

Another type of space is an US-space; those where sequences have at most one limit.

```
definition
  IsUS (˙–is US″) where
 T–is US″ ≡ ∀N x y. (N:nat→⋃T) ∧ NetConvTop(⟨N,Le⟩,x,T) ∧ NetConvTop(⟨N,Le⟩,y,T)
⟶ y=x
```

68.6 First results

The proof in Topology˙ZF˙1b.thy shows that a Hausdorff space is KC.

```
corollary(in topology0) T2˙imp˙KC:
  assumes T–is T₂″
  shows T–is KC″
proof-
  {
    fix A assume A–is compact in″T
    then have A–is closed in″T using in˙t2˙compact˙is˙cl assms by auto
  }
  then show thesis unfolding IsKC˙def by auto
qed
```

From the spectrum of compactness, it follows that any KC-space is T_1.

```
lemma(in topology0) KC˙imp˙T1:
  assumes T–is KC″
  shows T–is T₁″
proof-
  {
    fix x assume A:x∈⋃T
    have Finite(–x″) by auto
    then have –x″–is in the spectrum of″(λT. (⋃T)–is compact in″T)
      using compact˙spectrum by auto moreover
    have (T–restricted to″–x″)–is a topology″ using Top˙1˙L4 by auto
    moreover have ⋃(T–restricted to″–x″)=–x″ using A unfolding RestrictedTo˙def
by auto
    ultimately have com:–x″–is compact in″(T–restricted to″–x″) unfolding Spec˙def
      by auto
    then have –x″–is compact in″T using compact˙subspace˙imp˙compact A by auto
    then have –x″–is closed in″T using assms unfolding IsKC˙def using A by auto
  }
  then show thesis using T1˙iff˙singleton˙closed by auto
qed
```

Even more, if a space is KC, then it is US. We already know that for T_2 spaces, any net or filter has at most one limit; and that this property is equivalent with T_2. The US property is much weaker because we don't know what happends with other nets that are not directed by the order on the natural numbers.

```
theorem(in topology0) KC˙imp˙US:
  assumes T–is KC″
  shows T–is US″
proof-
  {
    fix N x y  assume A:N:nat→⋃T ⟨N,Le⟩→ₙ x ⟨N,Le⟩→ₙ y x≠y
    have dir:Le directs nat using Le˙directs˙nat by auto moreover
    from A(1) have dom:domain(N)=nat using func1˙1˙L1 by auto
```

moreover note A(1) ultimately have Net:⟨N,Le⟩–is a net on″⋃T unfolding IsNet˙def
by auto
from A(3) have y:y∈⋃T unfolding NetConverges˙def[OF Net] by auto
from A(2) have x:x∈⋃T unfolding NetConverges˙def[OF Net] by auto
from A(2) have o1:∀U∈Pow(⋃T). x∈int(U) ⟶ (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U) unfolding NetConverges˙def[OF Net]
using dom by auto
{
assume B:∃n∈nat. ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=y
have –y″–is closed in″T using y T1˙iff˙singleton˙closed KC˙imp˙T1 assms by auto
then have o2:⋃T-–y″∈T unfolding IsClosed˙def by auto
then have int(⋃T-–y″)=⋃T-–y″ using Top˙2˙L3 by auto
with A(4) x have o3:x∈int(⋃T-–y″) by auto
from o2 have ⋃T-–y″∈Pow(⋃T) by auto
with o1 o3 obtain r where r:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈⋃T-–y″ by auto
from B obtain n where n:n∈nat ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=y by auto
from dir r(1) n(1) obtain z where ⟨r,z⟩∈Le⟨n,z⟩∈Lez∈nat unfolding IsDirectedSet˙def by auto
with r(2) n(2) have Nz∈⋃T-–y″ Nz=y by auto
then have False by auto
}
then have reg:∀n∈nat. ∃m∈nat. Nm≠y ∧ ⟨n,m⟩∈Le by auto
let NN=–⟨n,N(μ i. Ni≠y ∧ ⟨n,i⟩∈Le)⟩. n∈nat″
{
fix x z assume A1:⟨x, z⟩ ∈ NN
{
fix y' assume A2:⟨x,y'⟩∈NN
with A1 have z=y' by auto
}
then have ∀y'. ⟨x,y'⟩∈NN ⟶ z=y' by auto
}
then have ∀x z. ⟨x, z⟩ ∈ NN ⟶ (∀y'. ⟨x,y'⟩∈NN ⟶ z=y') by auto
moreover
{
fix n assume as:n∈nat
with reg obtain m where Nm≠y ∧ ⟨n,m⟩∈Le m∈nat by auto
then have LI:N(μ i. Ni≠y ∧ ⟨n,i⟩∈Le)≠y ⟨n,μ i. Ni≠y ∧ ⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠y ∧ ⟨n,m⟩∈Le m]
nat˙into˙Ord[of m] by auto
then have (μ i. Ni≠y ∧ ⟨n,i⟩∈Le)∈nat by auto
then have N(μ i. Ni≠y ∧ ⟨n,i⟩∈Le)∈⋃T using apply˙type[OF A(1)] by auto
with as have ⟨n,N(μ i. Ni≠y ∧ ⟨n,i⟩∈Le)⟩∈nat×⋃T by auto
}
then have NN∈Pow(nat×⋃T) by auto
ultimately have NFun:NN:nat→⋃T unfolding Pi˙def function˙def domain˙def by auto
{

fix n assume as:n∈nat
with reg obtain m where Nm≠y ∧ ⟨n,m⟩∈Le m∈nat by auto
then have LI:N(μ i. Ni≠y ∧ ⟨n,i⟩∈Le)≠y ⟨n,μ i. Ni≠y ∧ ⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠y ∧ ⟨n,m⟩∈Le m]
nat˙into˙Ord[of m] by auto
then have NNn≠y using apply˙equality[OF ˙ NFun] by auto
}
then have noy:∀n∈nat. NNn≠y by auto
have dom2:domain(NN)=nat by auto
then have net2:⟨NN,Le⟩–is a net on″⋃T unfolding IsNet˙def using NFun dir by auto
{
fix U assume U∈Pow(⋃T) x∈int(U)
then have (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U) using o1 by auto
then obtain r where r˙def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U by auto
{
fix s assume AA:⟨r,s⟩∈Le
with reg obtain m where Nm≠y ⟨s,m⟩∈Le by auto
then have ⟨s,μ i. Ni≠y ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠y ∧ ⟨s,m⟩∈Le m]
nat˙into˙Ord by auto
with AA have ⟨r,μ i. Ni≠y ∧ ⟨s,i⟩∈Le⟩∈Le using le˙trans by auto
with r˙def(2) have N(μ i. Ni≠y ∧ ⟨s,i⟩∈Le)∈U by blast
then have NNs∈U using apply˙equality[OF ˙ NFun] AA by auto
}
then have ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
with r˙def(1) have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
}
then have conv2:⟨NN,Le⟩→$_N$ x unfolding NetConverges˙def[OF net2] using x dom2 by auto
let A=–x″∪NNnat
{
fix M assume Acov:A⊆⋃M M⊆T
then have x∈⋃M by auto
then obtain U where U:x∈U U∈M by auto
with Acov(2) have UT:U∈T by auto
then have U=int(U) using Top˙2˙L3 by auto
with U(1) have x∈int(U) by auto
with conv2 obtain r where rr:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U
unfolding NetConverges˙def[OF net2] using dom2 UT by auto
have NresFun:restrict(NN,–n∈nat. ⟨n,r⟩∈Le″):–n∈nat. ⟨n,r⟩∈Le″→⋃T using restrict˙fun
[OF NFun, of –n∈nat. ⟨n,r⟩∈Le″] by auto
then have restrict(NN,–n∈nat. ⟨n,r⟩∈Le″)∈surj(–n∈nat. ⟨n,r⟩∈Le″,range(restrict(NN,–n∈nat. ⟨n,r⟩∈Le″)))
using fun˙is˙surj by auto moreover
have –n∈nat. ⟨n,r⟩∈Le″⊆nat by auto
then have –n∈nat. ⟨n,r⟩∈Le″≲nat using subset˙imp˙lepoll by auto
ultimately have range(restrict(NN,–n∈nat. ⟨n,r⟩∈Le″))≲–n∈nat. ⟨n,r⟩∈Le″

using surj˙fun˙inv˙2 by auto
moreover
have –n∈nat. ⟨n,0⟩∈Le″=–0″ by auto
then have Finite(–n∈nat. ⟨n,0⟩∈Le″) by auto moreover
{
fix j assume as:j∈nat Finite(–n∈nat. ⟨n,j⟩∈Le″)
{
fix t assume t∈–n∈nat. ⟨n,succ(j)⟩∈Le″
then have t∈nat ⟨t,succ(j)⟩∈Le by auto
then have t≤succ(j) by auto
then have t⊆succ(j) using le˙imp˙subset by auto
then have t⊆j ∪–j″ using succ˙explained by auto
then have j∈t∨t⊆j by auto
then have j∈t∨t≤j using subset˙imp˙le ‹t∈nat› ‹j∈nat› nat˙into˙Ord by auto
then have j ∪–j″⊆t∨t≤j using ‹t∈nat› ‹j∈nat› nat˙into˙Ord unfolding Ord˙def
Transset˙def by auto
then have succ(j)⊆t∨t≤j using succ˙explained by auto
with ‹t⊆succ(j)› have t=succ(j)∨t≤j by auto
with ‹t∈nat› ‹j∈nat› have t∈–n∈nat. ⟨n,j⟩∈Le″ ∪ –succ(j)″ by auto
}
then have –n∈nat. ⟨n,succ(j)⟩∈Le″ ⊆–n∈nat. ⟨n,j⟩∈Le″ ∪ –succ(j)″ by auto
moreover have Finite(–n∈nat. ⟨n,j⟩∈Le″ ∪ –succ(j)″) using as(2) Finite˙cons
by auto
ultimately have Finite(–n∈nat. ⟨n,succ(j)⟩∈Le″) using subset˙Finite by auto
}
then have ∀j∈nat. Finite(–n∈nat. ⟨n,j⟩∈Le″) ⟶ Finite(–n∈nat. ⟨n,succ(j)⟩∈Le″)
by auto
ultimately have Finite(range(restrict(NN, –n ∈ nat . ⟨n, r⟩ ∈ Le″)))
using lepoll˙Finite[of range(restrict(NN, –n ∈ nat . ⟨n, r⟩ ∈ Le″))
–n ∈ nat . ⟨n, r⟩ ∈ Le″] ind˙on˙nat[OF ‹r∈nat›, where P=λt. Finite(–n∈nat.
⟨n,t⟩∈Le″)] by auto
then have Finite((restrict(NN, –n ∈ nat . ⟨n, r⟩ ∈ Le″))–n ∈ nat . ⟨n, r⟩ ∈ Le″)
using range˙image˙domain[OF NresFun]
by auto
then have Finite(NN–n ∈ nat . ⟨n, r⟩ ∈ Le″) using restrict˙image by auto
then have (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)–is in the spectrum of″(λT. (⋃T)–is
compact in″T) using compact˙spectrum by auto
moreover have ⋃(T–restricted to″NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)=⋃T∩NN–n ∈
nat . ⟨n, r⟩ ∈ Le″
unfolding RestrictedTo˙def by auto moreover
have ⋃T∩NN–n ∈ nat . ⟨n, r⟩ ∈ Le″=NN–n ∈ nat . ⟨n, r⟩ ∈ Le″
using func1˙1˙L6(2)[OF NFun] by blast
moreover have (T–restricted to″NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)–is a topology″
using Top˙1˙L4 by auto
ultimately have (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)–is compact in″(T–restricted
to″NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)
unfolding Spec˙def by force
then have (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)–is compact in″T using compact˙subspace˙imp˙compact
by auto

moreover from Acov(1) have (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)⊆⋃M by auto
moreover note Acov(2) ultimately
obtain 𝔑 where 𝔑:𝔑∈FinPow(M) (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)⊆⋃𝔑
unfolding IsCompact˙def by blast
from 𝔑(1) have 𝔑 ∪–U″∈FinPow(M) using U(2) unfolding FinPow˙def by auto moreover
{
fix s assume s:s∈A s∉U
with U(1) have s∈NNnat by auto
then have s∈–NNn. n∈nat″ using func˙imagedef[OF NFun] by auto
then obtain n where n:n∈nat s=NNn by auto
{
assume ⟨r,n⟩∈Le
with rr have NNn∈U by auto
with n(2) s(2) have False by auto
}
then have ⟨r,n⟩∉Le by auto
with rr(1) n(1) have ¬(r≤n) by auto
then have n≤r using Ord˙linear˙le[where thesis=⟨n,r⟩∈Le] nat˙into˙Ord[OF rr(1)]
nat˙into˙Ord[OF n(1)] by auto
with rr(1) n(1) have ⟨n,r⟩∈Le by auto
with n(2) have s∈–NNt. t∈–n∈nat. ⟨n,r⟩∈Le″″ by auto moreover
have –n∈nat. ⟨n,r⟩∈Le″⊆nat by auto
ultimately have s∈NN–n∈nat. ⟨n,r⟩∈Le″ using func˙imagedef[OF NFun]
by auto
with 𝔑(2) have s∈⋃𝔑 by auto
}
then have A⊆⋃𝔑 ∪ U by auto
then have A⊆⋃(𝔑 ∪ –U″) by auto ultimately
have ∃𝔑∈FinPow(M). A⊆⋃𝔑 by auto
}
then have ∀M∈Pow(T). A⊆⋃M ⟶ (∃𝔑∈FinPow(M). A⊆⋃𝔑) by auto moreover
have A⊆⋃T using func1˙1˙L6(2)[OF NFun] x by blast ultimately
have A–is compact in″T unfolding IsCompact˙def by auto
with assms have A–is closed in″T unfolding IsKC˙def IsCompact˙def by auto
then have ⋃T-A∈T unfolding IsClosed˙def by auto
then have ⋃T-A=int(⋃T-A) using Top˙2˙L3 by auto moreover
{
assume y∈A
with A(4) have y∈NNnat by auto
then have y∈–NNn. n∈nat″ using func˙imagedef[OF NFun] by auto
then obtain n where n∈natNNn=y by auto
with noy have False by auto
}
with y have y∈⋃T-A by force ultimately
have y∈int(⋃T-A) ⋃T-A∈Pow(⋃T) by auto moreover
have (∀U∈Pow(⋃T). y ∈ int(U) ⟶ (∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m

∈ U))
using A(3) dom unfolding NetConverges˙def[OF Net] by auto
ultimately have ∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ ⋃T-A by blast
then obtain r where r˙def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈⋃T-A by auto
{
fix s assume AA:⟨r,s⟩∈Le
with reg obtain m where Nm≠y ⟨s,m⟩∈Le by auto
then have ⟨s,μ i. Ni≠y ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠y ∧ ⟨s,m⟩∈Le m]
nat˙into˙Ord by auto
with AA have ⟨r,μ i. Ni≠y ∧ ⟨s,i⟩∈Le⟩∈Le using le˙trans by auto
with r˙def(2) have N(μ i. Ni≠y ∧ ⟨s,i⟩∈Le)∈⋃T-A by force
then have NNs∈⋃T-A using apply˙equality[OF ˙ NFun] AA by auto
moreover have NNs∈–NNt. t∈nat″ using AA by auto
then have NNs∈NNnat using func˙imagedef[OF NFun] by auto
then have NNs∈A by auto
ultimately have False by auto
}
moreover have r⊆succ(r) using succ˙explained by auto
then have r≤succ(r) using subset˙imp˙le nat˙into˙Ord ‹r∈nat› nat˙succI by auto
then have ⟨r,succ(r)⟩∈Le using ‹r∈nat› nat˙succI by auto
ultimately have False by auto
}
then have ∀N x y. (N:nat→⋃T) ∧ (⟨N, Le⟩ $\to_N$ x –in″ T) ∧ (⟨N, Le⟩ $\to_N$ y –in″ T)
⟶ x=y by auto
then show thesis unfolding IsUS˙def by auto
qed

US spaces are also T_1.

theorem (in topology0) US˙imp˙T1:
assumes T–is US″
shows T–is T_1″
proof-
{
fix x assume x:x∈⋃T
then have –x″⊆⋃T by auto
{
fix y assume y:y≠x y∈cl(–x″)
then have r:∀U∈T. y∈U ⟶ x∈U using cl˙inter˙neigh[OF ‹–x″⊆⋃T›] by auto
let N=ConstantFunction(nat,x)
have fun:N:nat→⋃T using x func1˙3˙L1 by auto
then have dom:domain(N)=nat using func1˙1˙L1 by auto
with fun have Net:⟨N,Le⟩–is a net on″⋃T using Le˙directs˙nat unfolding IsNet˙def
by auto
{
fix U assume U∈Pow(⋃T) x∈int(U)

then have x∈U using Top˙2˙L1 by auto
then have ∀n∈nat. Nn∈U using func1˙3˙L2 by auto
then have ∀n∈nat. ⟨0,n⟩∈Le ⟶Nn∈U by auto
then have ∃r∈nat. ∀n∈nat. ⟨r,n⟩∈Le ⟶Nn∈U by auto
}
then have ⟨N,Le⟩ $\rightarrow_N$ x unfolding NetConverges˙def[OF Net] using x dom by auto moreover
{
fix U assume U∈Pow(⋃T) y∈int(U)
then have x∈int(U) using r Top˙2˙L2 by auto
then have x∈U using Top˙2˙L1 by auto
then have ∀n∈nat. Nn∈U using func1˙3˙L2 by auto
then have ∀n∈nat. ⟨0,n⟩∈Le ⟶Nn∈U by auto
then have ∃r∈nat. ∀n∈nat. ⟨r,n⟩∈Le ⟶Nn∈U by auto
}
then have ⟨N,Le⟩ $\rightarrow_N$ y unfolding NetConverges˙def[OF Net] using y(2) dom Top˙3˙L11(1)[OF ‹–x″⊆⋃T›] by auto
ultimately have x=y using assms unfolding IsUS˙def using fun by auto
with y(1) have False by auto
}
then have cl(–x″)⊆–x″ by auto
then have cl(–x″)=–x″ using cl˙contains˙set[OF ‹–x″⊆⋃T›] by auto
then have –x″–is closed in″T using Top˙3˙L8 x by auto
}
then show thesis using T1˙iff˙singleton˙closed by auto
qed

68.7 Counter-examples

We need to find counter-examples that prove that this properties are new ones.

We know that $T_2 \Rightarrow loc.T_2 \Rightarrow$ anti-hyperconnected $\Rightarrow T_1$ and $T_2 \Rightarrow KC \Rightarrow US \Rightarrow T_1$. The question is: What is the relation between KC or US and, $loc.T_2$ or anti-hyperconnected?

In the file Topology˙ZF˙properties˙2.thy we built a topological space which is locally-T_2 but no T_2. It happends actually that this space is not even US given the appropiate topology T.

lemma (in topology0) locT2˙not˙US˙1:
assumes –m″∉T –m″–is closed in″T ∃N∈nat→⋃T. (⟨N,Le⟩$\rightarrow_N$ m) ∧ m∉Nnat
shows ∃N∈nat→⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″). (⟨N,Le⟩$\rightarrow_N$ ⋃T –in″ (T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″))
∧ (⟨N,Le⟩$\rightarrow_N$ m –in″ (T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″))
proof-
from assms(3) obtain N where N:N:nat→⋃T ⟨N,Le⟩$\rightarrow_N$ m m∉Nnat by auto
have ⋃T⊆⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) using assms(2) union˙doublepoint˙top
by auto

with N(1) have fun:N:nat→⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) using func1˙1˙L1B by auto
then have dom:domain(N)=nat using func1˙1˙L1 by auto
with fun have Net:⟨N,Le⟩–is a net on″⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) unfolding
IsNet˙def using Le˙directs˙nat by auto
from N(1) dom have Net2:⟨N,Le⟩–is a net on″⋃T unfolding IsNet˙def using Le˙directs˙nat by auto
from N(2) have R:∀U∈Pow(⋃T). m∈int(U) ⟶(∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U)
unfolding NetConverges˙def[OF Net2] using dom by auto
{
fix U assume U:U∈Pow(⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)) m∈Interior(U,T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)
let I=Interior(U,T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)
have I∈T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ using topology0.Top˙2˙L2 assms(2) doble˙point˙top unfolding topology0˙def by blast
then have (⋃T)∩I∈(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–restricted to″⋃T unfolding RestrictedTo˙def by blast
then have (⋃T)∩I∈T using open˙subspace˙double˙point(1) assms(2) by auto moreover
then have int((⋃T)∩I)=(⋃T)∩I using Top˙2˙L3 by auto
with U(2) assms(2) have m∈int((⋃T)∩I) unfolding IsClosed˙def by auto
moreover note R ultimately have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈(⋃T)∩I by blast
then have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈I by blast
then have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U using topology0.Top˙2˙L1[of T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″U] doble˙point˙top assms(2)
unfolding topology0˙def by auto
}
then have ∀U∈Pow(⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)). m∈Interior(U,T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) ⟶(∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U) by auto
moreover have tt:topology0(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) using doble˙point˙top[OF assms(2)] unfolding topology0˙def.
have m∈⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) using assms(2) union˙doublepoint˙top unfolding IsClosed˙def by auto ultimately
have con1:(⟨N,Le⟩→$_N$ m –in″ (T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)) unfolding topology0.NetConverges˙def[OF tt Net]
using dom by auto
{
fix U assume U:U∈Pow(⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)) ⋃T∈Interior(U,T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)
let I=Interior(U,T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)
have I∈T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ using topology0.Top˙2˙L2 assms(2) doble˙point˙top unfolding topology0˙def by blast
with U(2) mem˙not˙refl have I∈–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″ by auto
then obtain V W where VW:I=(V-–m″)∪–⋃T″∪W W∈T V∈T m∈V by auto

```
    from VW(3,4) have m∈int(V) using Top˙2˙L3 by auto moreover
    have V∈Pow(⋃T) using VW(3) by auto moreover
    note R ultimately
    have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈V by blast moreover
   from N(3) have ∀s∈nat. Ns≠m using func˙imagedef[OF N(1)] by auto ultimately
    have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈V-–m″ by blast
    then have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈I using VW(1) by auto
   then have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U using topology0.Top˙2˙L1 assms(2)
doble˙point˙top unfolding topology0˙def by blast
  }
   then have ∀U∈Pow(⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)).
⋃T∈Interior(U,T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) ⟶ (∃r∈nat.
∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U) by auto
  moreover have ⋃T∈⋃(T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″) us-
ing assms(2) union˙doublepoint˙top by auto ultimately
  have (⟨N,Le⟩→N ⋃T –in″ (T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″))
unfolding topology0.NetConverges˙def[OF tt Net]
    using dom by auto
  with con1 fun show thesis by auto
qed

corollary (in topology0) locT2˙not˙US˙2:
  assumes –m″∉T –m″–is closed in″T ∃N∈nat→⋃T. (⟨N,Le⟩→N m) ∧ m∉Nnat
  shows ¬((T∪–(U-–m″)∪–⋃T″∪W. ⟨U,W⟩∈–V∈T. m∈V″×T″)–is US″)
proof-
  have m≠⋃T using assms(2) mem˙not˙refl unfolding IsClosed˙def by auto
  then show thesis using locT2˙not˙US˙1 assms unfolding IsUS˙def by blast
qed
```

In particular, we also know that a locally-T_2 space doesn't need to be KC; since KC⇒US. Also we know that anti-hyperconnected spaces don't need to be KC or US, since locally-T_2 ⇒anti-hyperconnected.

Let's find a KC space that is not T_2, an US space which is not KC and a T_1 space which is not US.

First, let's prove some lemmas about what relation is there between this properties under the influence of other ones. This will help us to find counter-examples.

Anti-compactness ereases the differences between several properties.

```
lemma (in topology0) anticompact˙KC˙equiv˙T1:
  assumes T–is anti-compact″
  shows T–is KC″⟷T–is T1″
proof
  assume T–is KC″
  then show T–is T1″ using KC˙imp˙T1 by auto
next
  assume AS:T–is T1″
```

```
  {
    fix A assume A:A–is compact in″T A∈Pow(⋃T)
   then have A–is compact in″(T–restricted to″A) A∈Pow(⋃T) using compact˙imp˙compact˙subspace
      Compact˙is˙card˙nat by auto
     moreover then have ⋃(T–restricted to″A)=A unfolding RestrictedTo˙def by
auto
      ultimately have (⋃(T–restricted to″A))–is compact in″(T–restricted to″A)
A∈Pow(⋃T) by auto
     with assms have Finite(A) unfolding IsAntiComp˙def antiProperty˙def using
compact˙spectrum by auto
    then obtain n where n∈nat A≈n unfolding Finite˙def by auto
    then have A≺nat using eq˙lesspoll˙trans n˙lesspoll˙nat by auto moreover
    have ⋃T-(⋃T-A)=A using A(2) by auto
    ultimately have ⋃T-(⋃T-A)≺nat by auto
    then have ⋃T-A∈CoFinite ⋃T unfolding Cofinite˙def CoCardinal˙def by auto
    then have ⋃T-A∈T using AS T1˙cocardinal˙coarser by auto
    with A(2) have A–is closed in″T unfolding IsClosed˙def by auto
  }
  then show T–is KC″ unfolding IsKC˙def by auto
qed
```

Then if we find an anti-compact and T_1 but no T_2 space, there is a counter-example for $KC \Rightarrow T_2$. A counter-example for US doesn't need to be KC mustn't be anti-compact.

The cocountable topology on csucc(nat) is such a topology.

The cocountable topology on $\mathbb{N}^+$ is hyperconnected.

```
lemma cocountable˙in˙csucc˙nat˙HConn:
  shows (CoCountable csucc(nat))–is hyperconnected″
proof-
  {
   fix U V assume as:U∈(CoCountable csucc(nat))V∈(CoCountable csucc(nat))U∩V=0
    then have csucc(nat)-U≺csucc(nat)∨U=0csucc(nat)-V≺csucc(nat)∨V=0
      unfolding Cocountable˙def CoCardinal˙def by auto
   then have (csucc(nat)-U)∪(csucc(nat)-V)≺csucc(nat)∨U=0∨V=0 using less˙less˙imp˙un˙less[
      OF ˙ ˙ InfCard˙csucc[OF InfCard˙nat]] by auto moreover
    {
      assume (csucc(nat)-U)∪(csucc(nat)-V)≺csucc(nat) moreover
      have (csucc(nat)-U)∪(csucc(nat)-V)=csucc(nat)-U∩V by auto
      with as(3) have (csucc(nat)-U)∪(csucc(nat)-V)=csucc(nat) by auto
      ultimately have csucc(nat)≺csucc(nat) by auto
      then have False by auto
    }
    ultimately have U=0∨V=0 by auto
  }
 then show (CoCountable csucc(nat))–is hyperconnected″ unfolding IsHConnected˙def
by auto
qed
```

The cocountable topology on $\mathbb{N}^+$ is not anti-hyperconnected.

```
corollary cocountable˙in˙csucc˙nat˙notAntiHConn:
  shows ¬((CoCountable csucc(nat))–is anti-″IsHConnected)
proof
  assume as:(CoCountable csucc(nat))–is anti-″IsHConnected
 have (CoCountable csucc(nat))–is hyperconnected″ using cocountable˙in˙csucc˙nat˙HConn
by auto moreover
  have csucc(nat)≠0 using Ord˙0˙lt˙csucc[OF Ord˙nat] by auto
  then have uni:⋃(CoCountable csucc(nat))=csucc(nat) using union˙cocardinal un-
folding Cocountable˙def by auto
  have ∀A∈(CoCountable csucc(nat)). A⊆⋃(CoCountable csucc(nat)) by fast
  with uni have ∀A∈(CoCountable csucc(nat)). A⊆csucc(nat) by auto
  then have ∀A∈(CoCountable csucc(nat)). csucc(nat)∩A=A by auto
 ultimately have ((CoCountable csucc(nat))–restricted to″csucc(nat))–is hypercon-
nected″
    unfolding RestrictedTo˙def by auto
 with as have (csucc(nat))–is in the spectrum of″IsHConnected unfolding antiProp-
erty˙def
    using uni by auto
  then have csucc(nat)≲1 using HConn˙spectrum by auto
  then have csucc(nat)≺nat using n˙lesspoll˙nat lesspoll˙trans1 by auto
 then show False using lt˙csucc[OF Ord˙nat] lt˙Card˙imp˙lesspoll[OF Card˙csucc[OF
Ord˙nat]]
    lesspoll˙trans by auto
qed
```

The cocountable topology on $\mathbb{N}^+$ is not T_2.

```
theorem cocountable˙in˙csucc˙nat˙noT2:
  shows ¬(CoCountable csucc(nat))–is T2″
proof
  assume (CoCountable csucc(nat))–is T2″
  then have antiHC:(CoCountable csucc(nat))–is anti-″IsHConnected
   using topology0.T2˙imp˙anti˙HConn[OF topology0˙CoCardinal[OF InfCard˙csucc[OF
InfCard˙nat]]]
    unfolding Cocountable˙def by auto
  then show False using cocountable˙in˙csucc˙nat˙notAntiHConn by auto
qed
```

The cocountable topology on $\mathbb{N}^+$ is T_1.

```
theorem cocountable˙in˙csucc˙nat˙T1:
  shows (CoCountable csucc(nat))–is T1″
   using cocardinal˙is˙T1[OF InfCard˙csucc[OF InfCard˙nat]] unfolding Cocount-
able˙def by auto
```

The cocountable topology on $\mathbb{N}^+$ is anti-compact.

```
theorem cocountable˙in˙csucc˙nat˙antiCompact:
  shows (CoCountable csucc(nat))–is anti-compact″
proof-
```

```
 have noE:csucc(nat)≠0 using Ord˙0˙lt˙csucc[OF Ord˙nat] by auto
 {
  fix A assume as:A⊆⋃(CoCountable csucc(nat)) (⋃((CoCountable csucc(nat))–restricted
to″A))–is compact in″((CoCountable csucc(nat))–restricted to″A)
     from as(1) have ass:A⊆csucc(nat) using union˙cocardinal[OF noE] unfolding
Cocountable˙def by auto
   have ((CoCountable csucc(nat))–restricted to″A)=CoCountable (A∩csucc(nat))
using subspace˙cocardinal
      unfolding Cocountable˙def by auto moreover
    from ass have A∩csucc(nat)=A by auto
   ultimately have ((CoCountable csucc(nat))–restricted to″A)=CoCountable A by
auto
    with as(2) have comp:(⋃(CoCountable A))–is compact in″(CoCountable A) by
auto
    {
      assume as2:A≺csucc(nat) moreover
      {
        fix t assume t:t∈A
        have A-–t″⊆A by auto
        then have A-–t″≲A using subset˙imp˙lepoll by auto
        with as2 have A-–t″≺csucc(nat) using lesspoll˙trans1 by auto moreover note
noE
     ultimately have (A-–t″)–is closed in″(CoCountable A) using closed˙sets˙cocardinal[of
csucc(nat)
          A-–t″A] unfolding Cocountable˙def by auto
     then have A-(A-–t″)∈(CoCountable A) unfolding IsClosed˙def using union˙cocardinal[OF
noE, of A]
          unfolding Cocountable˙def by auto moreover
        from t have A-(A-–t″)=–t″ by auto ultimately
        have –t″∈(CoCountable A) by auto
      }
      then have r:∀t∈A. –t″∈(CoCountable A) by auto
      {
        fix U assume U:U∈Pow(A)
        {
          fix t assume t∈U
          with U r have t∈–t″–t″⊆U–t″∈(CoCountable A) by auto
          then have ∃V∈(CoCountable A). t∈V ∧ V⊆U by auto
        }
         then have U∈(CoCountable A) using topology0.open˙neigh˙open[OF topol-
ogy0˙CoCardinal[
          OF InfCard˙csucc[OF InfCard˙nat]]] unfolding Cocountable˙def by auto
      }
      then have Pow(A)⊆(CoCountable A) by auto moreover
      {
        fix B assume B∈(CoCountable A)
        then have B∈Pow(⋃(CoCountable A)) by auto
          then have B∈Pow(A) using union˙cocardinal[OF noE] unfolding Cocount-
able˙def by auto
```

```
    }
    ultimately have p:Pow(A)=(CoCountable A) by auto
    then have (CoCountable A)–is anti-compact″ using pow˙anti˙compact[of A] by
auto moreover
    from p have ⋃(CoCountable A)=⋃Pow(A) by auto
    then have tot:⋃(CoCountable A)=A by auto
    from comp have (⋃((CoCountable A)–restricted to″(⋃(CoCountable A))))–is
compact in″((CoCountable A)–restricted to″(⋃(CoCountable A))) using compact˙imp˙compact˙subspace
      Compact˙is˙card˙nat tot unfolding RestrictedTo˙def by auto
    ultimately have A–is in the spectrum of″(λT. (⋃T)–is compact in″T)
      using comp tot unfolding IsAntiComp˙def antiProperty˙def by auto
  }
  moreover
  {
    assume as1:¬(A≺csucc(nat))
    from ass have A≲csucc(nat) using subset˙imp˙lepoll by auto
    with as1 have A≈csucc(nat) using lepoll˙iff˙leqpoll by auto
    then have csucc(nat)≈A using eqpoll˙sym by auto
    then have nat≺A using lesspoll˙eq˙trans lt˙csucc[OF Ord˙nat]
      lt˙Card˙imp˙lesspoll[OF Card˙csucc[OF Ord˙nat]] by auto
    then have nat≲A using lepoll˙iff˙leqpoll by auto
    then obtain f where f∈inj(nat,A) unfolding lepoll˙def by auto moreover
    then have fun:f:nat→A unfolding inj˙def by auto
    then have f∈surj(nat,range(f)) using fun˙is˙surj by auto
    ultimately have f∈bij(nat,range(f)) unfolding bij˙def inj˙def surj˙def by auto
    then have nat≈range(f) unfolding eqpoll˙def by auto
    then have e:range(f)≈nat using eqpoll˙sym by auto
  then have as2:range(f)≺csucc(nat) using lt˙Card˙imp˙lesspoll[OF Card˙csucc[OF
Ord˙nat]]
      lt˙csucc[OF Ord˙nat] eq˙lesspoll˙trans by auto
   then have range(f)–is closed in″(CoCountable A) using closed˙sets˙cocardinal[of
csucc(nat)
        range(f)A] unfolding Cocountable˙def using func1˙1˙L5B[OF fun] noE by
auto
    then have (A∩range(f))–is compact in″(CoCountable A) using compact˙closed
union˙cocardinal[OF noE, of A]
      comp Compact˙is˙card˙nat unfolding Cocountable˙def by auto
  moreover have int:A∩range(f)=range(f)range(f)∩A=range(f) using func1˙1˙L5B[OF
fun] by auto
    ultimately have range(f)–is compact in″(CoCountable A) by auto
     then have range(f)–is compact in″((CoCountable A)–restricted to″range(f))
using compact˙imp˙compact˙subspace
      Compact˙is˙card˙nat by auto
  moreover have ((CoCountable A)–restricted to″range(f))=CoCountable (range(f)∩A)
      using subspace˙cocardinal unfolding Cocountable˙def by auto
       with int(2) have ((CoCountable A)–restricted to″range(f))=CoCountable
range(f) by auto
   ultimately have comp2:range(f)–is compact in″(CoCountable range(f)) by auto
    {
```

fix t assume t:t∈range(f)
have range(f)-–t″⊆range(f) by auto
then have range(f)-–t″≲range(f) using subset˙imp˙lepoll by auto
with as2 have range(f)-–t″≺csucc(nat) using lesspoll˙trans1 by auto moreover note noE
ultimately have (range(f)-–t″)–is closed in″(CoCountable range(f)) using closed˙sets˙cocardinal[of csucc(nat)
range(f)-–t″range(f)] unfolding Cocountable˙def by auto
then have range(f)-(range(f)-–t″)∈(CoCountable range(f)) unfolding IsClosed˙def using union˙cocardinal[OF noE, of range(f)]
unfolding Cocountable˙def by auto moreover
from t have range(f)-(range(f)-–t″)=–t″ by auto ultimately
have –t″∈(CoCountable range(f)) by auto
}
then have r:∀t∈range(f). –t″∈(CoCountable range(f)) by auto
{
fix U assume U:U∈Pow(range(f))
{
fix t assume t∈U
with U r have t∈–t″–t″⊆U–t″∈(CoCountable range(f)) by auto
then have ∃V∈(CoCountable range(f)). t∈V ∧ V⊆U by auto
}
then have U∈(CoCountable range(f)) using topology0.open˙neigh˙open[OF topology0˙CoCardinal[
OF InfCard˙csucc[OF InfCard˙nat]]] unfolding Cocountable˙def by auto
}
then have Pow(range(f))⊆(CoCountable range(f)) by auto moreover
{
fix B assume B∈(CoCountable range(f))
then have B∈Pow(⋃(CoCountable range(f))) by auto
then have B∈Pow(range(f)) using union˙cocardinal[OF noE] unfolding Cocountable˙def by auto
}
ultimately have p:Pow(range(f))=(CoCountable range(f)) by blast
then have (CoCountable range(f))–is anti-compact″ using pow˙anti˙compact[of range(f)] by auto moreover
from p have ⋃(CoCountable range(f))=⋃Pow(range(f)) by auto
then have tot:⋃(CoCountable range(f))=range(f) by auto
from comp2 have (⋃((CoCountable range(f))–restricted to″(⋃(CoCountable range(f)))))–is compact in″((CoCountable range(f))–restricted to″(⋃(CoCountable range(f)))) using compact˙imp˙compact˙subspace
Compact˙is˙card˙nat tot unfolding RestrictedTo˙def by auto
ultimately have range(f)–is in the spectrum of″(λT. (⋃T)–is compact in″T)
using comp tot unfolding IsAntiComp˙def antiProperty˙def by auto
then have Finite(range(f)) using compact˙spectrum by auto
then have Finite(nat) using e eqpoll˙imp˙Finite˙iff by auto
then have False using nat˙not˙Finite by auto
}
ultimately have A–is in the spectrum of″(λT. (⋃T)–is compact in″T) by auto

```
  }
  then have ∀A∈Pow(⋃(CoCountable csucc(nat))). ((⋃((CoCountable csucc(nat))
–restricted to″ A)) –is compact in″ ((CoCountable csucc(nat)) –restricted to″ A))
    ⟶ (A–is in the spectrum of″(λT. (⋃T)–is compact in″T)) by auto
  then show thesis unfolding IsAntiComp˙def antiProperty˙def by auto
qed
```

In conclusion, the cocountable topology defined on csucc(nat) is KC but not T_2. Also note that is KC but not anti-hyperconnected, hence KC or US spaces need not to be sober.

The cofinite topology on the natural numbers is T_1, but not US.

```
theorem cofinite˙not˙US:
  shows ¬((CoFinite nat)–is US″)
proof
  assume A:(CoFinite nat)–is US″
  let N=id(nat)
  have f:N:nat→nat using id˙type by auto
  then have fun:N:nat→⋃(CoCardinal(nat,nat)) using union˙cocardinal unfolding
Cofinite˙def by auto
  then have dom:domain(N)=nat using func1˙1˙L1 by auto
  with fun have NET:⟨N,Le⟩–is a net on″⋃(CoCardinal(nat,nat)) unfolding Is-
Net˙def
    using Le˙directs˙nat by auto
  have tot:⋃(CoCardinal(nat,nat))=nat using union˙cocardinal by auto
  {
    fix U n assume U:U∈Pow(⋃(CoFinite nat)) n∈Interior(U,(CoFinite nat))
    have Interior(U,(CoFinite nat))∈(CoFinite nat) using topology0.Top˙2˙L2
      topology0˙CoCardinal[OF InfCard˙nat] unfolding Cofinite˙def by auto
    then have nat-Interior(U,(CoFinite nat))≺nat using U(2) unfolding Cofinite˙def
      CoCardinal˙def by auto
    then have Finite(nat-Interior(U,(CoFinite nat))) using lesspoll˙nat˙is˙Finite by
auto moreover
    have nat-U⊆nat-Interior(U,(CoFinite nat)) using topology0.Top˙2˙L1
      topology0˙CoCardinal[OF InfCard˙nat] unfolding Cofinite˙def by auto
    ultimately have fin:Finite(nat-U) using subset˙Finite by auto
    moreover have lin:IsLinOrder(nat,Le) using Le˙directs˙nat(1) by auto
    then have IsLinOrder(nat-U,Le) using ord˙linear˙subset[of nat Le nat-U] by auto
    ultimately have r:nat-U=0 ∨ (∀r∈nat-U. ⟨r,Maximum(Le,nat-U)⟩∈Le) using
linord˙max˙props(3)[of nat-ULenat-U]
      unfolding FinPow˙def by auto
    {
      assume reg:∀s∈nat. ∃r∈nat. ⟨s,r⟩∈Le ∧ Nr∉U
        with r have s:(∀r∈nat-U. ⟨r,Maximum(Le,nat-U)⟩∈Le) nat-U≠0 using ap-
ply˙type[OF f] by auto
      have Maximum(Le,nat-U)∈nat using linord˙max˙props(2)[OF lin ˙ s(2)] fin
        unfolding FinPow˙def by auto
      then have succ(Maximum(Le,nat-U))∈nat using nat˙succI by auto
      with reg have ∃r∈nat. ⟨succ(Maximum(Le,nat-U)),r⟩∈Le ∧ Nr∉U by auto
```

then obtain r where r˙def:r∈nat ⟨succ(Maximum(Le,nat-U)),r⟩∈Le Nr∉U by auto
from r˙def(1,3) have Nr∈nat-U using apply˙type[OF f] by auto
with s(1) have ⟨Nr,Maximum(Le,nat-U)⟩∈Le by auto
then have ⟨r,Maximum(Le,nat-U)⟩∈Le using id˙conv r˙def(1) by auto
then have r¡succ(Maximum(Le,nat-U)) by auto
with r˙def(2) have r¡r using lt˙trans2 by auto
then have False by auto
}
then have ∃s∈nat. ∀r∈nat. ⟨s,r⟩∈Le ⟶ Nr∈U by auto
}
then have ∀n∈nat. ∀U∈Pow(⋃(CoFinite nat)). n∈Interior(U,CoFinite nat) ⟶ (∃s∈nat. ∀r∈nat. ⟨s,r⟩∈Le ⟶ Nr∈U) by auto
with tot have ∀n∈⋃(CoCardinal(nat,nat)). ∀U∈Pow(⋃(CoCardinal(nat,nat))). n∈Interior(U,CoCardinal(nat,nat)) ⟶ (∃s∈nat. ∀r∈nat. ⟨s,r⟩∈Le ⟶ Nr∈U)
unfolding Cofinite˙def by auto
then have ∀n∈⋃(CoCardinal(nat,nat)). (⟨N,Le⟩→$_N$ n –in″(CoCardinal(nat,nat))) unfolding topology0.NetConverges˙def[OF topology0˙CoCardinal[OF InfCard˙nat] NET]
using dom by auto
with tot have ∀n∈nat. (⟨N,Le⟩→$_N$ n –in″(CoFinite nat)) unfolding Cofinite˙def by auto
then have (⟨N,Le⟩→$_N$ 0 –in″(CoFinite nat)) ∧ (⟨N,Le⟩→$_N$ 1 –in″(CoFinite nat)) ∧ 0≠1 by auto
then show False using A unfolding IsUS˙def using fun unfolding Cofinite˙def by auto
qed

To end, we need a space which is US but no KC. This example comes from the one point compactification of a T_2, anti-compact and non discrete space. This T_2, anti-compact and non discrete space comes from a construction over the cardinal $\mathbb{N}^+$ or csucc(nat).

theorem extension˙pow˙top:
shows (Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–is a topology″
proof-
have noE:csucc(nat)≠0 using Ord˙0˙lt˙csucc[OF Ord˙nat] by auto
{
fix M assume M:M⊆(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
let MP=–U∈M. U∈Pow(csucc(nat))″
let MN=–U∈M. U∉Pow(csucc(nat))″
have unM:⋃M=(⋃MP)∪(⋃MN) by auto
have csucc(nat)∉csucc(nat) using mem˙not˙refl by auto
with M have MN:MN=–U∈M. U∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″″ by auto
have unMP:⋃MP∈Pow(csucc(nat)) by auto
then have MN=0⟶⋃M∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)

using unM by auto moreover
{
assume MN≠0
with MN have –U∈M. U∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″″≠0 by auto
then obtain U where U:U∈M U∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″ by blast
then obtain S where S:U=–csucc(nat)″∪S S∈(CoCountable csucc(nat))-–0″ by auto
with U MN have csucc(nat)∈U U∈MN by auto
then have a1:csucc(nat)∈⋃MN by auto
let SC=–S∈(CoCountable csucc(nat)). –csucc(nat)″∪S∈M″
have unSC:⋃SC∈(CoCountable csucc(nat)) using CoCar˙is˙topology[OF InfCard˙csucc[OF InfCard˙nat]]
unfolding IsATopology˙def unfolding Cocountable˙def by blast
{
fix s assume s∈–csucc(nat)″∪⋃SC
then have s=csucc(nat)∨s∈⋃SC by auto
then have s∈⋃MN∨(∃S∈SC. s∈S) using a1 by auto
then have s∈⋃MN∨(∃S∈(CoCountable csucc(nat)). –csucc(nat)″∪S∈M ∧ s∈S) by auto
with MN have s∈⋃MN∨(∃S∈(CoCountable csucc(nat)). –csucc(nat)″∪S∈MN ∧ s∈S) by auto
then have s∈⋃MN by blast
}
then have –csucc(nat)″∪⋃SC⊆⋃MN by blast
moreover
{
fix s assume s∈⋃MN
then obtain U where U:s∈U U∈M U∉Pow(csucc(nat)) by auto
with M have U∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))″ by auto
then obtain S where S:U=–csucc(nat)″∪S S∈(CoCountable csucc(nat)) by auto
with U(1) have s=csucc(nat)∨s∈S by auto
with S U(2) have s=csucc(nat)∨s∈⋃SC by auto
then have s∈–csucc(nat)″∪⋃SC by auto
}
then have ⋃MN⊆–csucc(nat)″∪⋃SC by blast
ultimately have unMN:⋃MN=–csucc(nat)″∪⋃SC by auto
from unSC have b1:csucc(nat)-⋃SC≺csucc(nat)∨⋃SC=0 unfolding Cocountable˙def CoCardinal˙def
by auto
{
assume 0∈SC
then have –csucc(nat)″∈M by auto
then have –csucc(nat)″∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″ using mem˙not˙refl
M by auto
then obtain S where S:S∈(CoCountable csucc(nat))-–0″ –csucc(nat)″=–csucc(nat)″∪S

by auto
{
fix x assume x∈S
then have x∈–csucc(nat)″∪S by auto
with S(2) have x∈–csucc(nat)″ by auto
then have x=csucc(nat) by auto
}
then have S⊆–csucc(nat)″ by auto
with S(1) have S=–csucc(nat)″ by auto
with S(1) have csucc(nat)-–csucc(nat)″≺csucc(nat) unfolding Cocountable˙def CoCardinal˙def
by auto moreover
then have csucc(nat)-–csucc(nat)″=csucc(nat) using mem˙not˙refl[of csucc(nat)] by force
ultimately have False by auto
}
then have 0∉SC by auto moreover
from S U(1) have S∈SC by auto
ultimately have S⊆⋃SC S≠0 by auto
then have noe:⋃SC≠0 by auto
with b1 have csucc(nat)-⋃SC≺csucc(nat) by auto
moreover have csucc(nat)-(⋃SC ∪ ⋃MP)⊆csucc(nat)-⋃SC by auto
then have csucc(nat)-(⋃SC ∪ ⋃MP)≲csucc(nat)-⋃SC using subset˙imp˙lepoll by auto
ultimately have csucc(nat)-(⋃SC ∪ ⋃MP)≺csucc(nat) using lesspoll˙trans1 by auto moreover
have ⋃SC⊆⋃(CoCountable csucc(nat)) using unSC by auto
then have ⋃SC⊆csucc(nat) using union˙cocardinal[OF noE] unfolding Cocountable˙def by auto
ultimately have (⋃SC ∪ ⋃MP)∈(CoCountable csucc(nat))
using unMP unfolding Cocountable˙def CoCardinal˙def by auto
then have –csucc(nat)″∪(⋃SC ∪ ⋃MP)∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
using noe by auto moreover
from unM unMN have ⋃M=(–csucc(nat)″∪⋃SC) ∪ ⋃MP by auto
then have ⋃M=–csucc(nat)″∪(⋃SC ∪ ⋃MP) by auto
ultimately have ⋃M∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) by auto
}
ultimately have ⋃M∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) by auto
}
then have ∀M∈Pow(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″). ⋃M∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) by auto
moreover
{
fix U V assume UV:U∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) V∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)

```
    {
      assume csucc(nat)∉U∨csucc(nat)∉V
      with UV have U∈Pow(csucc(nat))∨V∈Pow(csucc(nat)) by auto
      then have U∩V∈Pow(csucc(nat)) by auto
    then have U∩V∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
by auto
    }
    moreover
    {
      assume csucc(nat)∈U∧csucc(nat)∈V
    then obtain SU SV where S:U=–csucc(nat)″∪SU V=–csucc(nat)″∪SV SU∈(CoCountable
csucc(nat))-–0″
        SV∈(CoCountable csucc(nat))-–0″ using UV mem˙not˙refl by auto
      from S(1,2) have U∩V=–csucc(nat)″∪(SU∩SV) by auto moreover
    from S(3,4) have SU∩SV∈(CoCountable csucc(nat)) using CoCar˙is˙topology[OF
InfCard˙csucc[OF InfCard˙nat]] unfolding IsATopology˙def
        unfolding Cocountable˙def by blast moreover
       from S(3,4) have SU∩SV≠0 using cocountable˙in˙csucc˙nat˙HConn unfolding
IsHConnected˙def
        by auto ultimately
      have U∩V∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″ by auto
    then have U∩V∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
by auto
    }
     ultimately have U∩V∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable
csucc(nat))-–0″″) by auto
  }
  then have ∀U∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″).
∀V∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″). U∩V∈(Pow(csucc(nat))
∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) by auto
  ultimately show thesis unfolding IsATopology˙def by auto
qed
```

This topology is defined over $\mathbb{N}^+ \cup \{\mathbb{N}^+\}$ or csucc(nat)∪–csucc(nat)″.

```
lemma extension˙pow˙union:
  shows ⋃(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)=csucc(nat)∪–csucc(nat)″
proof
  have noE:csucc(nat)≠0 using Ord˙0˙lt˙csucc[OF Ord˙nat] by auto
  have ⋃(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)=⋃(Pow(csucc(nat)))
∪ (⋃—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
    by blast
  also have . . .=csucc(nat) ∪ (⋃—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
by auto
  ultimately have A:⋃(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)=csucc(nat)
∪ (⋃—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) by auto
  have ⋃(CoCountable csucc(nat))∈(CoCountable csucc(nat)) using CoCar˙is˙topology[OF
InfCard˙csucc[OF InfCard˙nat]]
    unfolding IsATopology˙def Cocountable˙def by auto
  then have csucc(nat)∈(CoCountable csucc(nat)) using union˙cocardinal[OF noE]
```

unfolding Cocountable˙def
by auto
with noE have csucc(nat)∈(CoCountable csucc(nat))-–0″ by auto
then have –csucc(nat)″∪csucc(nat)∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″
by auto
then have –csucc(nat)″∪csucc(nat)⊆⋃—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″
by blast
with A show csucc(nat)∪–csucc(nat)″⊆⋃(Pow(csucc(nat)) ∪ —csucc(nat)″∪S.
S∈(CoCountable csucc(nat))-–0″″)
by auto
{
fix x assume x:x∈(⋃—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) x≠csucc(nat)
then obtain U where U:U∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″
x∈U by blast
then obtain S where S:U=–csucc(nat)″∪S S∈(CoCountable csucc(nat))-–0″ by
auto
with U(2) x(2) have x∈S by auto
with S(2) have x∈⋃(CoCountable csucc(nat)) by auto
then have x∈csucc(nat) using union˙cocardinal[OF noE] unfolding Cocount-
able˙def by auto
}
then have (⋃—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)⊆csucc(nat)
∪–csucc(nat)″ by blast
with A show ⋃(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)⊆csucc(nat)∪–csucc(n
by blast
qed

This topology has a discrete open subspace.

lemma extension˙pow˙subspace:
shows (Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted
to″csucc(nat)=Pow(csucc(nat))
and csucc(nat)∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
proof
show csucc(nat)∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
by auto
{
fix x assume x∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted
to″csucc(nat)
then obtain R where x=csucc(nat)∩R R∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S.
S∈(CoCountable csucc(nat))-–0″″) unfolding RestrictedTo˙def
by auto
then have x∈Pow(csucc(nat)) by auto
}
then show (Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted
to″csucc(nat)⊆Pow(csucc(nat)) by auto
{
fix x assume x:x∈Pow(csucc(nat))
then have x=csucc(nat)∩x by auto
with x have x∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted

to″csucc(nat)
unfolding RestrictedTo˙def by auto
}
then show Pow(csucc(nat))⊆(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted to″csucc(nat) by auto
qed

This topology is Hausdorff.

theorem extension˙pow˙T2:
shows (Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–is T_2″
proof-
have noE:csucc(nat)≠0 using Ord˙0˙lt˙csucc[OF Ord˙nat] by auto
{
fix A B assume A∈⋃(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
B∈⋃(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) A≠B
then have AB:A∈csucc(nat)∪–csucc(nat)″ B∈csucc(nat)∪–csucc(nat)″ A≠B using extension˙pow˙union by auto
{
assume A≠csucc(nat) B≠csucc(nat)
then have A∈csucc(nat) B∈csucc(nat) using AB by auto
then have sub:–A″∈Pow(csucc(nat)) –B″∈Pow(csucc(nat)) by auto
then have –A″∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted to″csucc(nat)
–B″∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted to″csucc(nat) using extension˙pow˙subspace(1)
by auto
then obtain RA RB where –A″=csucc(nat)∩RA –B″=csucc(nat)∩RB RA∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
RB∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) unfolding RestrictedTo˙def by auto
then have –A″∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) –B″∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
using extension˙pow˙subspace(2) extension˙pow˙top unfolding IsATopology˙def by auto
moreover
from AB(3) have –A″∩–B″=0 by auto ultimately
have ∃U∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″). ∃V∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″). A∈U∧B∈V∧U∩V=0 by auto
}
moreover
{
assume A=csucc(nat)∨B=csucc(nat)
with AB(3) have disj:(A=csucc(nat)∧B≠csucc(nat))∨(B=csucc(nat)∧A≠csucc(nat)) by auto
{

assume ass:A=csucc(nat)∧B≠csucc(nat)
then have p:B∈csucc(nat) using AB(2) by auto
have –B″≈1 using singleton˙eqpoll˙1 by auto
then have –B″≺nat using eq˙lesspoll˙trans n˙lesspoll˙nat by auto
then have –B″≲nat using lesspoll˙imp˙lepoll by auto
then have –B″≺csucc(nat) using Card˙less˙csucc˙eq˙le[OF Card˙nat] by auto
with p have –B″–is closed in″(CoCountable csucc(nat)) unfolding Cocountable˙def
using closed˙sets˙cocardinal[OF noE] by auto
then have csucc(nat)-–B″∈(CoCountable csucc(nat)) unfolding IsClosed˙def
Cocountable˙def using union˙cocardinal[OF noE] by auto moreover
{
assume csucc(nat)-–B″=0
with p have csucc(nat)=–B″ by auto
then have csucc(nat)≈1 using singleton˙eqpoll˙1 by auto
then have csucc(nat)≲nat using eq˙lesspoll˙trans n˙lesspoll˙nat lesspoll˙imp˙lepoll by auto
then have csucc(nat)≺csucc(nat) using Card˙less˙csucc˙eq˙le[OF Card˙nat] by auto
then have False by auto
}
ultimately
have –csucc(nat)″∪(csucc(nat)-–B″)∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″ by auto
then have U1:–csucc(nat)″∪(csucc(nat)-–B″)∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) by auto
have –B″∈Pow(csucc(nat)) using p by auto
then have –B″∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted to″csucc(nat)
using extension˙pow˙subspace(1) by auto
then obtain R where R∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″ –B″=csucc(nat)∩R
unfolding RestrictedTo˙def by auto
then have U2:–B″∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″ using extension˙pow˙subspace(2)
extension˙pow˙top unfolding IsATopology˙def by auto
have (–csucc(nat)″∪(csucc(nat)-–B″))∩–B″=0 using p mem˙not˙refl[of csucc(nat)] by auto
with U1 U2 have ∃U∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″. ∃V∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″.
A∈U∧B∈V∧U∩V=0 using ass(1) by auto
}
moreover
{
assume ¬(A=csucc(nat)∧B≠csucc(nat))
then have ass:B = csucc(nat) ∧ A ≠ csucc(nat) using disj by auto
then have p:A∈csucc(nat) using AB(1) by auto
have –A″≈1 using singleton˙eqpoll˙1 by auto
then have –A″≺nat using eq˙lesspoll˙trans n˙lesspoll˙nat by auto

then have –A″≲nat using lesspoll˙imp˙lepoll by auto
then have –A″≺csucc(nat) using Card˙less˙csucc˙eq˙le[OF Card˙nat] by auto
with p have –A″–is closed in″(CoCountable csucc(nat)) unfolding Cocountable˙def
using closed˙sets˙cocardinal[OF noE] by auto
then have csucc(nat)-–A″∈(CoCountable csucc(nat)) unfolding IsClosed˙def
Cocountable˙def using union˙cocardinal[OF noE] by auto moreover
{
assume csucc(nat)-–A″=0
with p have csucc(nat)=–A″ by auto
then have csucc(nat)≈1 using singleton˙eqpoll˙1 by auto
then have csucc(nat)≲nat using eq˙lesspoll˙trans n˙lesspoll˙nat lesspoll˙imp˙lepoll by auto
then have csucc(nat)≺csucc(nat) using Card˙less˙csucc˙eq˙le[OF Card˙nat] by auto
then have False by auto
}
ultimately
have –csucc(nat)″∪(csucc(nat)-–A″)∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″ by auto
then have U1:–csucc(nat)″∪(csucc(nat)-–A″)∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) by auto
have –A″∈Pow(csucc(nat)) using p by auto
then have –A″∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted to″csucc(nat)
using extension˙pow˙subspace(1) by auto
then obtain R where R∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″ –A″=csucc(nat)∩R
unfolding RestrictedTo˙def by auto
then have U2:–A″∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″ using extension˙pow˙subspace(2)
extension˙pow˙top unfolding IsATopology˙def by auto
have int:–A″∩(–csucc(nat)″∪(csucc(nat)-–A″))=0 using p mem˙not˙refl[of csucc(nat)] by auto
have A∈–A″ csucc(nat)∈(–csucc(nat)″∪(csucc(nat)-–A″)) by auto
with int U1 have ∃V∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″.
A∈–A″∧csucc(nat)∈V∧–A″∩V=0 by auto
with U2 have ∃U∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″. ∃V∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″.
A∈U∧csucc(nat)∈V∧U∩V=0 using exI[where P=λU. U∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″∧(∃V∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″.
A∈U∧csucc(nat)∈V∧U∩V=0) and x=–A″] unfolding Bex˙def by auto
then have ∃U∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″. ∃V∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″.
A∈U∧B∈V∧U∩V=0 using ass by auto
}
ultimately have ∃U∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable

csucc(nat))-–0″″. ∃V∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″.
A∈U∧B∈V∧U∩V=0 by auto
}
ultimately have ∃U∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable
csucc(nat))-–0″″. ∃V∈Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″.
A∈U∧B∈V∧U∩V=0 by auto
}
then show thesis unfolding isT2˙def by auto
qed

The topology we built is not discrete; i.e., not every set is open.

theorem extension˙pow˙notDiscrete:
shows –csucc(nat)″∉(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
proof
assume –csucc(nat)″∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
then have –csucc(nat)″∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″ using
mem˙not˙refl by auto
then obtain S where S:S∈(CoCountable csucc(nat))-–0″ –csucc(nat)″=–csucc(nat)″∪S
by auto
{
fix x assume x∈S
then have x∈–csucc(nat)″∪S by auto
with S(2) have x∈–csucc(nat)″ by auto
then have x=csucc(nat) by auto
}
then have S⊆–csucc(nat)″ by auto
with S(1) have S=–csucc(nat)″ by auto
with S(1) have csucc(nat)-–csucc(nat)″≺csucc(nat) unfolding Cocountable˙def
CoCardinal˙def
by auto moreover
then have csucc(nat)-–csucc(nat)″=csucc(nat) using mem˙not˙refl[of csucc(nat)]
by force
ultimately show False by auto
qed

The topology we built is anti-compact.

theorem extension˙pow˙antiCompact:
shows (Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–is
anti-compact″
proof-
have noE:csucc(nat)≠0 using Ord˙0˙lt˙csucc[OF Ord˙nat] by auto
{
fix K assume K:K⊆⋃(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable
csucc(nat))-–0″″)
(⋃((Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted
to″K))–is compact in″((Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted
to″K)
from K(1) have sub:K⊆csucc(nat) ∪–csucc(nat)″ using extension˙pow˙union by
auto

have (⋃((Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted to″K))=(csucc(nat) ∪–csucc(nat)″)∩K
using extension˙pow˙union unfolding RestrictedTo˙def by auto moreover
from sub have (csucc(nat) ∪–csucc(nat)″)∩K=K by auto
ultimately have (⋃((Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted to″K))=K by auto
with K(2) have K–is compact in″((Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted to″K) by auto
then have comp:K–is compact in″(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) using
compact˙subspace˙imp˙compact by auto
{
assume ss:K⊆csucc(nat)
then have K–is compact in″((Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted to″csucc(nat))
using compact˙imp˙compact˙subspace comp Compact˙is˙card˙nat by auto
then have K–is compact in″Pow(csucc(nat)) using extension˙pow˙subspace(1) by auto
then have K–is compact in″(Pow(csucc(nat))–restricted to″K) using compact˙imp˙compact˙subspace
Compact˙is˙card˙nat by auto moreover
have ⋃(Pow(csucc(nat))–restricted to″K)=K using ss unfolding RestrictedTo˙def by auto
ultimately have (⋃(Pow(csucc(nat))–restricted to″K))–is compact in″(Pow(csucc(nat))–restricted to″K) by auto
then have K–is in the spectrum of″(λT. (⋃T)–is compact in″T) using pow˙anti˙compact
unfolding IsAntiComp˙def antiProperty˙def using ss by auto
}
moreover
{
assume ¬(K⊆csucc(nat))
with sub have csucc(nat)∈K by auto
with sub have ss:K-–csucc(nat)″⊆csucc(nat) by auto
{
assume prec:K-–csucc(nat)″≺csucc(nat)
then have (K-–csucc(nat)″)–is closed in″(CoCountable csucc(nat))
using closed˙sets˙cocardinal[OF noE] ss unfolding Cocountable˙def by auto
then have csucc(nat)-(K-–csucc(nat)″)∈(CoCountable csucc(nat)) unfolding IsClosed˙def
Cocountable˙def using union˙cocardinal[OF noE] by auto moreover
{
assume csucc(nat)-(K-–csucc(nat)″)=0
with ss have csucc(nat)=(K-–csucc(nat)″) by auto
with prec have False by auto
}
ultimately have –csucc(nat)″ ∪(csucc(nat)-(K-–csucc(nat)″))∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″
by auto

moreover have –csucc(nat)″ ∪(csucc(nat)-(K-–csucc(nat)″))=(–csucc(nat)″ ∪ csucc(nat))-(K-–csucc(nat)″) by blast
ultimately have (–csucc(nat)″ ∪ csucc(nat))-(K-–csucc(nat)″)∈—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″ by auto
then have (–csucc(nat)″ ∪ csucc(nat))-(K-–csucc(nat)″)∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
by auto moreover
have csucc(nat) ∪ –csucc(nat)″=–csucc(nat)″ ∪ csucc(nat) by auto
ultimately have (csucc(nat) ∪ –csucc(nat)″)-(K-–csucc(nat)″)∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
by auto
then have (⋃(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″))-(K-–csucc(nat)″)∈ ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
using extension˙pow˙union by auto
then have (K-–csucc(nat)″)–is closed in″(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
unfolding IsClosed˙def using ss by auto
with comp have (K∩(K-–csucc(nat)″))–is compact in″(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) using compact˙closed
Compact˙is˙card˙nat by auto
moreover have K∩(K-–csucc(nat)″)=(K-–csucc(nat)″) by auto
ultimately have (K-–csucc(nat)″)–is compact in″(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) by auto
with ss have (K-–csucc(nat)″)–is compact in″((Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted to″csucc(nat))
using compact˙imp˙compact˙subspace comp Compact˙is˙card˙nat by auto
then have (K-–csucc(nat)″)–is compact in″(Pow(csucc(nat))) using extension˙pow˙subspace(1) by auto
then have (K-–csucc(nat)″)–is compact in″(Pow(csucc(nat))–restricted to″(K-–csucc(nat)″)) using compact˙imp˙compact˙subspace
Compact˙is˙card˙nat by auto moreover
have ⋃(Pow(csucc(nat))–restricted to″(K-–csucc(nat)″))=(K-–csucc(nat)″) using ss unfolding RestrictedTo˙def by auto
ultimately have (⋃(Pow(csucc(nat))–restricted to″(K-–csucc(nat)″)))–is compact in″(Pow(csucc(nat))–restricted to″(K-–csucc(nat)″)) by auto
then have (K-–csucc(nat)″)–is in the spectrum of″(λT. (⋃T)–is compact in″T) using pow˙anti˙compact
unfolding IsAntiComp˙def antiProperty˙def using ss by auto
then have Finite(K-–csucc(nat)″) using compact˙spectrum by auto moreover
have Finite(–csucc(nat)″) by auto ultimately
have Finite(K) using Diff˙Finite[of –csucc(nat)″ K] by auto
then have K–is in the spectrum of″(λT. (⋃T)–is compact in″T) using compact˙spectrum by auto
}
moreover
{
assume ¬(K-–csucc(nat)″≺csucc(nat))
with ss have K-–csucc(nat)″≈csucc(nat) using lepoll˙iff˙leqpoll subset˙imp˙lepoll[of K-–csucc(nat)″

csucc(nat)] by auto
then have csucc(nat)≈K-–csucc(nat)″ using eqpoll˙sym by auto
then have nat≺K-–csucc(nat)″ using lesspoll˙eq˙trans lt˙csucc[OF Ord˙nat]
lt˙Card˙imp˙lesspoll[OF Card˙csucc[OF Ord˙nat]] by auto
then have nat≲K-–csucc(nat)″ using lepoll˙iff˙leqpoll by auto
then obtain f where f∈inj(nat,K-–csucc(nat)″) unfolding lepoll˙def by auto
moreover
then have fun:f:nat→K-–csucc(nat)″ unfolding inj˙def by auto
then have f∈surj(nat,range(f)) using fun˙is˙surj by auto
ultimately have f∈bij(nat,range(f)) unfolding bij˙def inj˙def surj˙def by auto
then have nat≈range(f) unfolding eqpoll˙def by auto
then have e:range(f)≈nat using eqpoll˙sym by auto
then have as2:range(f)≺csucc(nat) using lt˙Card˙imp˙lesspoll[OF Card˙csucc[OF
Ord˙nat]]
lt˙csucc[OF Ord˙nat] eq˙lesspoll˙trans by auto
then have range(f)–is closed in″(CoCountable csucc(nat)) using closed˙sets˙cocardinal[of
csucc(nat)
range(f)csucc(nat)] unfolding Cocountable˙def using func1˙1˙L5B[OF fun]
ss noE by auto
then have csucc(nat)-(range(f))∈(CoCountable csucc(nat)) unfolding Is-
Closed˙def
Cocountable˙def using union˙cocardinal[OF noE] by auto moreover
{
assume csucc(nat)-(range(f))=0
with ss func1˙1˙L5B[OF fun] have csucc(nat)=(range(f)) by blast
with as2 have False by auto
}
ultimately have –csucc(nat)″ ∪(csucc(nat)-(range(f)))∈—csucc(nat)″∪S.
S∈(CoCountable csucc(nat))-–0″″
by auto
moreover have –csucc(nat)″ ∪(csucc(nat)-(range(f)))=(–csucc(nat)″ ∪ csucc(nat))-(range(f))
using func1˙1˙L5B[OF fun] by blast
ultimately have (–csucc(nat)″ ∪ csucc(nat))-(range(f))∈—csucc(nat)″∪S.
S∈(CoCountable csucc(nat))-–0″″ by auto
then have (–csucc(nat)″ ∪ csucc(nat))-(range(f))∈(Pow(csucc(nat)) ∪ —csucc(nat)″∪S.
S∈(CoCountable csucc(nat))-–0″″)
by auto moreover
have csucc(nat) ∪ –csucc(nat)″=–csucc(nat)″ ∪ csucc(nat) by auto
ultimately have (csucc(nat) ∪ –csucc(nat)″)-(range(f))∈(Pow(csucc(nat)) ∪
—csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
by auto
then have (⋃(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″))-(range(f))∈(Pow(cs
∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
using extension˙pow˙union by auto moreover
have range(f)⊆⋃(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable
csucc(nat))-–0″″) using ss func1˙1˙L5B[OF fun] by auto
ultimately have (range(f))–is closed in″(Pow(csucc(nat)) ∪ —csucc(nat)″∪S.
S∈(CoCountable csucc(nat))-–0″″)
unfolding IsClosed˙def by blast

with comp have (K∩(range(f)))–is compact in″(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) using compact˙closed
Compact˙is˙card˙nat by auto
moreover have K∩(range(f))=(range(f)) using func1˙1˙L5B[OF fun] by auto
ultimately have (range(f))–is compact in″(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″) by auto
with ss func1˙1˙L5B[OF fun] have (range(f))–is compact in″((Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)–restricted to″csucc(nat))
using compact˙imp˙compact˙subspace[of range(f) nat Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″ csucc(nat)] comp Compact˙is˙card˙nat by auto
then have (range(f))–is compact in″(Pow(csucc(nat))) using extension˙pow˙subspace(1) by auto
then have (range(f))–is compact in″(Pow(csucc(nat))–restricted to″(range(f))) using compact˙imp˙compact˙subspace
Compact˙is˙card˙nat by auto moreover
have ⋃(Pow(csucc(nat))–restricted to″(range(f)))=(range(f)) using ss func1˙1˙L5B[OF fun] unfolding RestrictedTo˙def by auto
ultimately have (⋃(Pow(csucc(nat))–restricted to″(range(f))))–is compact in″(Pow(csucc(nat))–restricted to″(range(f))) by auto
then have (range(f))–is in the spectrum of″(λT. (⋃T)–is compact in″T) using pow˙anti˙compact[of csucc(nat)]
unfolding IsAntiComp˙def antiProperty˙def using ss func1˙1˙L5B[OF fun] by auto
then have Finite(range(f)) using compact˙spectrum by auto moreover
then have Finite(nat) using e eqpoll˙imp˙Finite˙iff by auto ultimately
have False using nat˙not˙Finite by auto
}
ultimately have K –is in the spectrum of″(λT. (⋃T) –is compact in″ T) by auto
}
ultimately have K –is in the spectrum of″(λT. (⋃T) –is compact in″ T) by auto
}
then show thesis unfolding IsAntiComp˙def antiProperty˙def by auto
qed

If a topological space is KC, then its one-point compactification is US.

theorem (in topology0) KC˙imp˙OP˙comp˙is˙US:
assumes T–is KC″
shows (–one-point compactification of″T)–is US″
proof-
{
fix N x y assume A:N:nat→⋃(–one-point compactification of″T) ⟨N,Le⟩→$_N$ x–in″(–one-point compactification of″T) ⟨N,Le⟩→$_N$ y–in″(–one-point compactification of″T) x≠y
have dir:Le directs nat using Le˙directs˙nat(2).
from A(1) have dom:domain(N)=nat using func1˙1˙L1 by auto
with dir A(1) have NET:⟨N,Le⟩–is a net on″⋃(–one-point compactification of″T)

unfolding IsNet˙def by auto
have xy:x∈⋃(–one-point compactification of″T) y∈⋃(–one-point compactification of″T)
using A(2,3) topology0.NetConverges˙def[OF ˙ NET] unfolding topology0˙def using op˙comp˙is˙top dom by auto
then have pp:x∈⋃T ∪–⋃T″ y∈⋃T ∪–⋃T″ using op˙compact˙total by auto
from A(2) have comp:∀U∈Pow(⋃–one-point compactification of″T).
x ∈ Interior(U, –one-point compactification of″T) ⟶
(∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ U) using topology0.NetConverges˙def[OF ˙ NET, of x]
unfolding topology0˙def using op˙comp˙is˙top dom op˙compact˙total by auto
from A(3) have op2:∀U∈Pow(⋃–one-point compactification of″T).
y ∈ Interior(U, –one-point compactification of″T) ⟶
(∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ U) using topology0.NetConverges˙def[OF ˙ NET, of y]
unfolding topology0˙def using op˙comp˙is˙top dom op˙compact˙total by auto
{
assume p:x∈⋃T y∈⋃T
{
assume B:∃n∈nat. ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=⋃T
have ⋃T∈(–one-point compactification of″T) using open˙subspace by auto
then have ⋃T=Interior(⋃T,–one-point compactification of″T) using topology0.Top˙2˙L3
unfolding topology0˙def using op˙comp˙is˙top by auto
then have x∈Interior(⋃T,–one-point compactification of″T) using p(1) by auto moreover
have ⋃T∈Pow(⋃(–one-point compactification of″T)) using open˙subspace(1) by auto
ultimately have ∃t∈domain(fst(⟨N, Le⟩)). ∀m∈domain(fst(⟨N, Le⟩)). ⟨t, m⟩ ∈ snd(⟨N, Le⟩) ⟶ fst(⟨N, Le⟩) m ∈ ⋃T using A(2)
using topology0.NetConverges˙def[OF ˙ NET] op˙comp˙is˙top unfolding topology0˙def by blast
then have ∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ ⋃T using dom by auto
then obtain t where t:t∈nat ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ ⋃T by auto
from B obtain n where n:n∈nat ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=⋃T by auto
from t(1) n(1) dir obtain z where z:z∈nat ⟨n,z⟩∈Le ⟨t,z⟩∈Le unfolding IsDirectedSet˙def
by auto
from t(2) z(1,3) have Nz∈⋃T by auto moreover
from n(2) z(1,2) have Nz=⋃T by auto ultimately
have False using mem˙not˙refl by auto
}
then have reg:∀n∈nat. ∃m∈nat. Nm≠⋃T ∧ ⟨n,m⟩∈Le by auto
let NN=–⟨n,N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)⟩. n∈nat″
{
fix x z assume A1:⟨x, z⟩ ∈ NN
{
fix y' assume A2:⟨x,y'⟩∈NN
with A1 have z=y' by auto

}
then have ∀y'. ⟨x,y'⟩∈NN ⟶ z=y' by auto
}
then have ∀x z. ⟨x, z⟩ ∈ NN ⟶ (∀y'. ⟨x,y'⟩∈NN ⟶ z=y') by auto
moreover
{
fix n assume as:n∈nat
with reg obtain m where Nm≠⋃T ∧ ⟨n,m⟩∈Le m∈nat by auto
then have LI:N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)≠⋃T ⟨n,μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨n,m⟩∈Le m]
nat˙into˙Ord[of m] by auto
then have (μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)∈nat by auto
then have N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)∈⋃(–one-point compactification of″T) using apply˙type[OF A(1)] op˙compact˙total by auto
with as have ⟨n,N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)⟩∈nat×⋃(–one-point compactification of″T) by auto
}
then have NN∈Pow(nat×⋃(–one-point compactification of″T)) by auto
ultimately have NFun:NN:nat→⋃(–one-point compactification of″T) unfolding Pi˙def function˙def domain˙def by auto
{
fix n assume as:n∈nat
with reg obtain m where Nm≠⋃T ∧ ⟨n,m⟩∈Le m∈nat by auto
then have LI:N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)≠⋃T ⟨n,μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨n,m⟩∈Le m]
nat˙into˙Ord[of m] by auto
then have NNn≠⋃T using apply˙equality[OF ˙ NFun] by auto
}
then have noy:∀n∈nat. NNn≠⋃T by auto
then have ∀n∈nat. NNn∈⋃T using apply˙type[OF NFun] op˙compact˙total by auto
then have R:NN:nat→⋃T using func1˙1˙L1A[OF NFun] by auto
have dom2:domain(NN)=nat by auto
then have net2:⟨NN,Le⟩–is a net on″⋃T unfolding IsNet˙def using R dir by auto
{
fix U assume U:U⊆⋃T x∈int(U)
have intT:int(U)∈T using Top˙2˙L2 by auto
then have int(U)∈(–one-point compactification of″T) unfolding OPCompactification˙def
by auto
then have Interior(int(U),–one-point compactification of″T)=int(U) using topology0.Top˙2˙L3
unfolding topology0˙def using op˙comp˙is˙top by auto
with U(2) have x∈Interior(int(U),–one-point compactification of″T) by auto
with intT have (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈int(U)) using comp op˙compact˙total by auto
then obtain r where r˙def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U using Top˙2˙L1 by auto

{
fix s assume AA:⟨r,s⟩∈Le
with reg obtain m where Nm≠⋃T ⟨s,m⟩∈Le by auto
then have ⟨s,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨s,m⟩∈Le m]
nat˙into˙Ord by auto
with AA have ⟨r,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using le˙trans by auto
with r˙def(2) have N(μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le)∈U by blast
then have NNs∈U using apply˙equality[OF ˙ NFun] AA by auto
}
then have ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
with r˙def(1) have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
}
then have ∀U∈Pow(⋃T). x ∈ int(U)
⟶ (∃r∈nat. ∀s∈nat. ⟨r, s⟩ ∈ Le ⟶ NN s ∈ U) by auto
then have conx:⟨NN,Le⟩→$_N$ x–in″T using NetConverges˙def[OF net2] p(1) op˙comp˙is˙top
unfolding topology0˙def using xy(1) dom2 by auto
{
fix U assume U:U⊆⋃T y∈int(U)
have intT:int(U)∈T using Top˙2˙L2 by auto
then have int(U)∈(–one-point compactification of″T) unfolding OPCompactification˙def
by auto
then have Interior(int(U),–one-point compactification of″T)=int(U) using topology0.Top˙2˙L3
unfolding topology0˙def using op˙comp˙is˙top by auto
with U(2) have y∈Interior(int(U),–one-point compactification of″T) by auto
with intT have (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈int(U)) using op2 op˙compact˙total by auto
then obtain r where r˙def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U using Top˙2˙L1 by auto
{
fix s assume AA:⟨r,s⟩∈Le
with reg obtain m where Nm≠⋃T ⟨s,m⟩∈Le by auto
then have ⟨s,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨s,m⟩∈Le m]
nat˙into˙Ord by auto
with AA have ⟨r,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using le˙trans by auto
with r˙def(2) have N(μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le)∈U by blast
then have NNs∈U using apply˙equality[OF ˙ NFun] AA by auto
}
then have ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
with r˙def(1) have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
}
then have ∀U∈Pow(⋃T). y ∈ int(U)
⟶ (∃r∈nat. ∀s∈nat. ⟨r, s⟩ ∈ Le ⟶ NN s ∈ U) by auto
then have cony:⟨NN,Le⟩→$_N$ y–in″T using NetConverges˙def[OF net2] p(2) op˙comp˙is˙top

unfolding topology0˙def using xy(2) dom2 by auto
with conx assms have x=y using KC˙imp˙US unfolding IsUS˙def using R by auto
with A(4) have False by auto
}
moreover
{
assume AAA:x∉⋃T∨y∉⋃T
with pp have x=⋃T∨y=⋃T by auto
{
assume x:x=⋃T
with A(4) have y:y∈⋃T using pp(2) by auto
{
assume B:∃n∈nat. ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=⋃T
have ⋃T∈(–one-point compactification of″T) using open˙subspace by auto
then have ⋃T=Interior(⋃T,–one-point compactification of″T) using topology0.Top˙2˙L3
unfolding topology0˙def using op˙comp˙is˙top by auto
then have y∈Interior(⋃T,–one-point compactification of″T) using y by auto
moreover
have ⋃T∈Pow(⋃(–one-point compactification of″T)) using open˙subspace(1) by auto
ultimately have ∃t∈domain(fst(⟨N, Le⟩)). ∀m∈domain(fst(⟨N, Le⟩)). ⟨t, m⟩ ∈ snd(⟨N, Le⟩) ⟶ fst(⟨N, Le⟩) m ∈ ⋃T using A(3)
using topology0.NetConverges˙def[OF ˙ NET] op˙comp˙is˙top unfolding topology0˙def by blast
then have ∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ ⋃T using dom by auto
then obtain t where t:t∈nat ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ ⋃T by auto
from B obtain n where n:n∈nat ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=⋃T by auto
from t(1) n(1) dir obtain z where z:z∈nat ⟨n,z⟩∈Le ⟨t,z⟩∈Le unfolding IsDirectedSet˙def
by auto
from t(2) z(1,3) have Nz∈⋃T by auto moreover
from n(2) z(1,2) have Nz=⋃T by auto ultimately
have False using mem˙not˙refl by auto
}
then have reg:∀n∈nat. ∃m∈nat. Nm≠⋃T ∧ ⟨n,m⟩∈Le by auto
let NN=–⟨n,N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)⟩. n∈nat″
{
fix x z assume A1:⟨x, z⟩ ∈ NN
{
fix y' assume A2:⟨x,y'⟩∈NN
with A1 have z=y' by auto
}
then have ∀y'. ⟨x,y'⟩∈NN ⟶ z=y' by auto
}
then have ∀x z. ⟨x, z⟩ ∈ NN ⟶ (∀y'. ⟨x,y'⟩∈NN ⟶ z=y') by auto
moreover

{
fix n assume as:n∈nat
with reg obtain m where Nm≠⋃T ∧ ⟨n,m⟩∈Le m∈nat by auto
then have LI:N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)≠⋃T ⟨n,μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨n,m⟩∈Le m]
nat˙into˙Ord[of m] by auto
then have (μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)∈nat by auto
then have N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)∈⋃(–one-point compactification of″T) using apply˙type[OF A(1)] op˙compact˙total by auto
with as have ⟨n,N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)⟩∈nat×⋃(–one-point compactification of″T) by auto
}
then have NN∈Pow(nat×⋃(–one-point compactification of″T)) by auto
ultimately have NFun:NN:nat→⋃(–one-point compactification of″T) unfolding Pi˙def function˙def domain˙def by auto
{
fix n assume as:n∈nat
with reg obtain m where Nm≠⋃T ∧ ⟨n,m⟩∈Le m∈nat by auto
then have LI:N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)≠⋃T ⟨n,μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨n,m⟩∈Le m]
nat˙into˙Ord[of m] by auto
then have NNn≠⋃T using apply˙equality[OF ˙ NFun] by auto
}
then have noy:∀n∈nat. NNn≠⋃T by auto
then have ∀n∈nat. NNn∈⋃T using apply˙type[OF NFun] op˙compact˙total by auto
then have R:NN:nat→⋃T using func1˙1˙L1A[OF NFun] by auto
have dom2:domain(NN)=nat by auto
then have net2:⟨NN,Le⟩–is a net on″⋃T unfolding IsNet˙def using R dir by auto
{
fix U assume U:U⊆⋃T y∈int(U)
have intT:int(U)∈T using Top˙2˙L2 by auto
then have int(U)∈(–one-point compactification of″T) unfolding OPCompactification˙def
by auto
then have Interior(int(U),–one-point compactification of″T)=int(U) using topology0.Top˙2˙L3
unfolding topology0˙def using op˙comp˙is˙top by auto
with U(2) have y∈Interior(int(U),–one-point compactification of″T) by auto
with intT have (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈int(U)) using op2 op˙compact˙total by auto
then obtain r where r˙def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U using Top˙2˙L1 by auto
{
fix s assume AA:⟨r,s⟩∈Le
with reg obtain m where Nm≠⋃T ⟨s,m⟩∈Le by auto
then have ⟨s,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨s,m⟩∈Le m]

```
        nat˙into˙Ord by auto
      with AA have ⟨r,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using le˙trans by auto
      with r˙def(2) have N(μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le)∈U by blast
      then have NNs∈U using apply˙equality[OF ˙ NFun] AA by auto
    }
    then have ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
    with r˙def(1) have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
  }
  then have ∀U∈Pow(⋃T). y ∈ int(U)
    ⟶ (∃r∈nat. ∀s∈nat. ⟨r, s⟩ ∈ Le ⟶ NN  s ∈ U) by auto
     then have cony:⟨NN,Le⟩→N y–in″T using NetConverges˙def[OF net2] y
op˙comp˙is˙top
    unfolding topology0˙def using xy(2) dom2 by auto
  let A=–y″∪NNnat
  {
    fix M assume Acov:A⊆⋃M M⊆T
    then have y∈⋃M by auto
    then obtain V where V:y∈V V∈M by auto
    with Acov(2) have VT:V∈T by auto
    then have V=int(V) using Top˙2˙L3 by auto
    with V(1) have y∈int(V) by auto
    with cony obtain r where rr:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈V
      unfolding NetConverges˙def[OF net2, of y] using dom2 VT y by auto
  have NresFun:restrict(NN,–n∈nat. ⟨n,r⟩∈Le″):–n∈nat. ⟨n,r⟩∈Le″→⋃T using
restrict˙fun
      [OF R, of –n∈nat. ⟨n,r⟩∈Le″] by auto
  then have restrict(NN,–n∈nat. ⟨n,r⟩∈Le″)∈surj(–n∈nat. ⟨n,r⟩∈Le″,range(restrict(NN,–n∈nat.
⟨n,r⟩∈Le″)))
      using fun˙is˙surj by auto moreover
    have –n∈nat. ⟨n,r⟩∈Le″⊆nat by auto
    then have –n∈nat. ⟨n,r⟩∈Le″≲nat using subset˙imp˙lepoll by auto
    ultimately have range(restrict(NN,–n∈nat. ⟨n,r⟩∈Le″))≲–n∈nat. ⟨n,r⟩∈Le″
using surj˙fun˙inv˙2 by auto
    moreover
    have –n∈nat. ⟨n,0⟩∈Le″=–0″ by auto
    then have Finite(–n∈nat. ⟨n,0⟩∈Le″) by auto moreover
    {
      fix j assume as:j∈nat Finite(–n∈nat. ⟨n,j⟩∈Le″)
      {
        fix t assume t∈–n∈nat. ⟨n,succ(j)⟩∈Le″
        then have t∈nat ⟨t,succ(j)⟩∈Le by auto
        then have t≤succ(j) by auto
        then have t⊆succ(j) using le˙imp˙subset by auto
        then have t⊆j ∪–j″ using succ˙explained by auto
        then have j∈t∨t⊆j by auto
         then have j∈t∨t≤j using subset˙imp˙le ‹t∈nat› ‹j∈nat› nat˙into˙Ord by
auto
          then have j ∪–j″⊆t∨t≤j using ‹t∈nat› ‹j∈nat› nat˙into˙Ord unfolding
Ord˙def
```

Transset˙def by auto
then have succ(j)⊆t∨t≤j using succ˙explained by auto
with ‹t⊆succ(j)› have t=succ(j)∨t≤j by auto
with ‹t∈nat› ‹j∈nat› have t∈–n∈nat. ⟨n,j⟩∈Le″ ∪ –succ(j)″ by auto
}
then have –n∈nat. ⟨n,succ(j)⟩∈Le″ ⊆–n∈nat. ⟨n,j⟩∈Le″ ∪ –succ(j)″ by auto
moreover have Finite(–n∈nat. ⟨n,j⟩∈Le″ ∪ –succ(j)″) using as(2) Finite˙cons by auto
ultimately have Finite(–n∈nat. ⟨n,succ(j)⟩∈Le″) using subset˙Finite by auto
}
then have ∀j∈nat. Finite(–n∈nat. ⟨n,j⟩∈Le″) ⟶ Finite(–n∈nat. ⟨n,succ(j)⟩∈Le″) by auto
ultimately have Finite(range(restrict(NN, –n ∈ nat . ⟨n, r⟩ ∈ Le″)))
using lepoll˙Finite[of range(restrict(NN, –n ∈ nat . ⟨n, r⟩ ∈ Le″))
–n ∈ nat . ⟨n, r⟩ ∈ Le″] ind˙on˙nat[OF ‹r∈nat›, where P=λt. Finite(–n∈nat. ⟨n,t⟩∈Le″)] by auto
then have Finite((restrict(NN, –n ∈ nat . ⟨n, r⟩ ∈ Le″))–n ∈ nat . ⟨n, r⟩ ∈ Le″) using range˙image˙domain[OF NresFun]
by auto
then have Finite(NN–n ∈ nat . ⟨n, r⟩ ∈ Le″) using restrict˙image by auto
then have (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)–is in the spectrum of″(λT. (⋃T)–is compact in″T) using compact˙spectrum by auto
moreover have ⋃(T–restricted to″NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)=⋃T∩NN–n ∈ nat . ⟨n, r⟩ ∈ Le″
unfolding RestrictedTo˙def by auto moreover
have ⋃T∩NN–n ∈ nat . ⟨n, r⟩ ∈ Le″=NN–n ∈ nat . ⟨n, r⟩ ∈ Le″
using func1˙1˙L6(2)[OF R] by blast
moreover have (T–restricted to″NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)–is a topology″
using Top˙1˙L4 unfolding topology0˙def by auto
ultimately have (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)–is compact in″(T–restricted to″NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)
unfolding Spec˙def by force
then have (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)–is compact in″(T) using compact˙subspace˙imp˙compact by auto
moreover from Acov(1) have (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)⊆⋃M by auto
moreover note Acov(2) ultimately
obtain 𝔑 where 𝔑:𝔑∈FinPow(M) (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)⊆⋃𝔑
unfolding IsCompact˙def by blast
from 𝔑(1) have 𝔑 ∪–V″∈FinPow(M) using V(2) unfolding FinPow˙def by auto moreover
{
fix s assume s:s∈A s∉V
with V(1) have s∈NNnat by auto
then have s∈–NNn. n∈nat″ using func˙imagedef[OF NFun] by auto
then obtain n where n:n∈nat s=NNn by auto
{
assume ⟨r,n⟩∈Le

with rr have NNn∈V by auto
with n(2) s(2) have False by auto
}
then have ⟨r,n⟩∉Le by auto
with rr(1) n(1) have ¬(r≤n) by auto
then have n≤r using Ord˙linear˙le[where thesis=⟨n,r⟩∈Le] nat˙into˙Ord[OF rr(1)]
nat˙into˙Ord[OF n(1)] by auto
with rr(1) n(1) have ⟨n,r⟩∈Le by auto
with n(2) have s∈–NNt. t∈–n∈nat. ⟨n,r⟩∈Le˝˝ by auto moreover
have –n∈nat. ⟨n,r⟩∈Le˝⊆nat by auto
ultimately have s∈NN–n∈nat. ⟨n,r⟩∈Le˝ using func˙imagedef[OF NFun]
by auto
with $\mathfrak{N}$(2) have s∈⋃$\mathfrak{N}$ by auto
}
then have A⊆⋃$\mathfrak{N}$ ∪ V by auto
then have A⊆⋃($\mathfrak{N}$ ∪ –V˝) by auto ultimately
have ∃$\mathfrak{N}$∈FinPow(M). A⊆⋃$\mathfrak{N}$ by auto
}
then have ∀M∈Pow(T). A⊆⋃M ⟶ (∃$\mathfrak{N}$∈FinPow(M). A⊆⋃$\mathfrak{N}$) by auto moreover
have ss:A⊆⋃(T) using func1˙1˙L6(2)[OF R] y by blast ultimately
have A–is compact in˝(T) unfolding IsCompact˙def by auto moreover
with assms have A–is closed in˝(T) unfolding IsKC˙def IsCompact˙def by auto ultimately
have A∈–B∈Pow(⋃T). B–is compact in˝(T)∧B–is closed in˝(T)˝ using ss by auto
then have –⋃T˝∪(⋃T-A)∈(–one-point compactification of˝T) unfolding OPCompactification˙def
by auto
then have –⋃T˝∪(⋃T-A)=Interior(–⋃T˝∪(⋃T-A),–one-point compactification of˝T) using topology0.Top˙2˙L3 op˙comp˙is˙top
unfolding topology0˙def by auto moreover
{
assume x∈A
with A(4) have x∈NNnat by auto
then have x∈–NNn. n∈nat˝ using func˙imagedef[OF NFun] by auto
then obtain n where n∈natNNn=x by auto
with noy x have False by auto
}
with y have x∈–⋃T˝∪(⋃T-A) using x by force ultimately
have x∈Interior(–⋃T˝∪(⋃T-A),–one-point compactification of˝T) –⋃T˝∪(⋃T-A)∈Pow(⋃(–one-point compactification of˝T))
using op˙compact˙total by auto moreover
have (∀U∈Pow(⋃(–one-point compactification of˝T)). x ∈ Interior(U,–one-point compactification of˝T) ⟶ (∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ U))
using A(2) dom topology0.NetConverges˙def[OF ˙ NET] op˙comp˙is˙top unfolding topology0˙def by auto
ultimately have ∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ –⋃T˝∪(⋃T-A)

by blast
then obtain r where r˙def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈–⋃T″∪(⋃T-A)
by auto
{
fix s assume AA:⟨r,s⟩∈Le
with reg obtain m where Nm≠⋃T ⟨s,m⟩∈Le by auto
then have ⟨s,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧
⟨s,m⟩∈Le m]
nat˙into˙Ord by auto
with AA have ⟨r,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using le˙trans by auto
with r˙def(2) have N(μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le)∈–⋃T″∪(⋃T-A) by auto
then have NNs∈–⋃T″∪(⋃T-A) using apply˙equality[OF ˙ NFun] AA by
auto
with noy have NNs∈(⋃T-A) using AA by auto
moreover have NNs∈–NNt. t∈nat″ using AA by auto
then have NNs∈NNnat using func˙imagedef[OF NFun] by auto
then have NNs∈A by auto
ultimately have False by auto
}
moreover have r⊆succ(r) using succ˙explained by auto
then have r≤succ(r) using subset˙imp˙le nat˙into˙Ord ‹r∈nat› nat˙succI
by auto
then have ⟨r,succ(r)⟩∈Le using ‹r∈nat› nat˙succI by auto
ultimately have False by auto
}
then have x≠⋃T by auto
with xy(1) AAA have y∉⋃T x∈⋃T using op˙compact˙total by auto
with xy(2) have y:y=⋃T and x:x∈⋃T using op˙compact˙total by auto
{
assume B:∃n∈nat. ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=⋃T
have ⋃T∈(–one-point compactification of″T) using open˙subspace by auto
then have ⋃T=Interior(⋃T,–one-point compactification of″T) using topology0.Top˙2˙L3
unfolding topology0˙def using op˙comp˙is˙top by auto
then have x∈Interior(⋃T,–one-point compactification of″T) using x by auto
moreover
have ⋃T∈Pow(⋃(–one-point compactification of″T)) using open˙subspace(1)
by auto
ultimately have ∃t∈domain(fst(⟨N, Le⟩)). ∀m∈domain(fst(⟨N, Le⟩)). ⟨t, m⟩
∈ snd(⟨N, Le⟩) ⟶ fst(⟨N, Le⟩) m ∈ ⋃T using A(2)
using topology0.NetConverges˙def[OF ˙ NET] op˙comp˙is˙top unfolding
topology0˙def by blast
then have ∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ ⋃T using dom by auto
then obtain t where t:t∈nat ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ ⋃T by auto
from B obtain n where n:n∈nat ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=⋃T by auto
from t(1) n(1) dir obtain z where z:z∈nat ⟨n,z⟩∈Le ⟨t,z⟩∈Le unfolding
IsDirectedSet˙def
by auto
from t(2) z(1,3) have Nz∈⋃T by auto moreover

from n(2) z(1,2) have Nz=⋃T by auto ultimately
have False using mem˙not˙refl by auto
}
then have reg:∀n∈nat. ∃m∈nat. Nm≠⋃T ∧ ⟨n,m⟩∈Le by auto
let NN=–⟨n,N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)⟩. n∈nat″
{
fix x z assume A1:⟨x, z⟩ ∈ NN
{
fix y' assume A2:⟨x,y'⟩∈NN
with A1 have z=y' by auto
}
then have ∀y'. ⟨x,y'⟩∈NN ⟶ z=y' by auto
}
then have ∀x z. ⟨x, z⟩ ∈ NN ⟶ (∀y'. ⟨x,y'⟩∈NN ⟶ z=y') by auto
moreover
{
fix n assume as:n∈nat
with reg obtain m where Nm≠⋃T ∧ ⟨n,m⟩∈Le m∈nat by auto
then have LI:N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)≠⋃T ⟨n,μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨n,m⟩∈Le m]
nat˙into˙Ord[of m] by auto
then have (μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)∈nat by auto
then have N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)∈⋃(–one-point compactification of″T) using apply˙type[OF A(1)] op˙compact˙total by auto
with as have ⟨n,N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)⟩∈nat×⋃(–one-point compactification of″T) by auto
}
then have NN∈Pow(nat×⋃(–one-point compactification of″T)) by auto
ultimately have NFun:NN:nat→⋃(–one-point compactification of″T) unfolding Pi˙def function˙def domain˙def by auto
{
fix n assume as:n∈nat
with reg obtain m where Nm≠⋃T ∧ ⟨n,m⟩∈Le m∈nat by auto
then have LI:N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)≠⋃T ⟨n,μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨n,m⟩∈Le m]
nat˙into˙Ord[of m] by auto
then have NNn≠⋃T using apply˙equality[OF ˙ NFun] by auto
}
then have noy:∀n∈nat. NNn≠⋃T by auto
then have ∀n∈nat. NNn∈⋃T using apply˙type[OF NFun] op˙compact˙total by auto
then have R:NN:nat→⋃T using func1˙1˙L1A[OF NFun] by auto
have dom2:domain(NN)=nat by auto
then have net2:⟨NN,Le⟩–is a net on″⋃T unfolding IsNet˙def using R dir by auto
{
fix U assume U:U⊆⋃T x∈int(U)
have intT:int(U)∈T using Top˙2˙L2 by auto
then have int(U)∈(–one-point compactification of″T) unfolding OPCompact-

ification˙def
by auto
then have Interior(int(U),–one-point compactification of″T)=int(U) using topology0.Top˙2˙L3
unfolding topology0˙def using op˙comp˙is˙top by auto
with U(2) have x∈Interior(int(U),–one-point compactification of″T) by auto
with intT have (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈int(U)) using comp op˙compact˙total by auto
then obtain r where r˙def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U using Top˙2˙L1 by auto
{
fix s assume AA:⟨r,s⟩∈Le
with reg obtain m where Nm≠⋃T ⟨s,m⟩∈Le by auto
then have ⟨s,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨s,m⟩∈Le m]
nat˙into˙Ord by auto
with AA have ⟨r,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using le˙trans by auto
with r˙def(2) have N(μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le)∈U by blast
then have NNs∈U using apply˙equality[OF ˙ NFun] AA by auto
}
then have ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
with r˙def(1) have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
}
then have ∀U∈Pow(⋃T). x ∈ int(U)
⟶ (∃r∈nat. ∀s∈nat. ⟨r, s⟩ ∈ Le ⟶ NN s ∈ U) by auto
then have cony:⟨NN,Le⟩→$_N$ x–in″T using NetConverges˙def[OF net2] x op˙comp˙is˙top
unfolding topology0˙def using xy(2) dom2 by auto
let A=–x″∪NNnat
{
fix M assume Acov:A⊆⋃M M⊆T
then have x∈⋃M by auto
then obtain V where V:x∈V V∈M by auto
with Acov(2) have VT:V∈T by auto
then have V=int(V) using Top˙2˙L3 by auto
with V(1) have x∈int(V) by auto
with cony VT obtain r where rr:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈V
unfolding NetConverges˙def[OF net2, of x] using dom2 y by auto
have NresFun:restrict(NN,–n∈nat. ⟨n,r⟩∈Le″):–n∈nat. ⟨n,r⟩∈Le″→⋃T using restrict˙fun
[OF R, of –n∈nat. ⟨n,r⟩∈Le″] by auto
then have restrict(NN,–n∈nat. ⟨n,r⟩∈Le″)∈surj(–n∈nat. ⟨n,r⟩∈Le″,range(restrict(NN,–n∈nat. ⟨n,r⟩∈Le″)))
using fun˙is˙surj by auto moreover
have –n∈nat. ⟨n,r⟩∈Le″⊆nat by auto
then have –n∈nat. ⟨n,r⟩∈Le″≲nat using subset˙imp˙lepoll by auto
ultimately have range(restrict(NN,–n∈nat. ⟨n,r⟩∈Le″))≲–n∈nat. ⟨n,r⟩∈Le″ using surj˙fun˙inv˙2 by auto
moreover

have –n∈nat. ⟨n,0⟩∈Le″=–0″ by auto
then have Finite(–n∈nat. ⟨n,0⟩∈Le″) by auto moreover
{
fix j assume as:j∈nat Finite(–n∈nat. ⟨n,j⟩∈Le″)
{
fix t assume t∈–n∈nat. ⟨n,succ(j)⟩∈Le″
then have t∈nat ⟨t,succ(j)⟩∈Le by auto
then have t≤succ(j) by auto
then have t⊆succ(j) using le˙imp˙subset by auto
then have t⊆j ∪–j″ using succ˙explained by auto
then have j∈t∨t⊆j by auto
then have j∈t∨t≤j using subset˙imp˙le ‹t∈nat› ‹j∈nat› nat˙into˙Ord by auto
then have j ∪–j″⊆t∨t≤j using ‹t∈nat› ‹j∈nat› nat˙into˙Ord unfolding Ord˙def
Transset˙def by auto
then have succ(j)⊆t∨t≤j using succ˙explained by auto
with ‹t⊆succ(j)› have t=succ(j)∨t≤j by auto
with ‹t∈nat› ‹j∈nat› have t∈–n∈nat. ⟨n,j⟩∈Le″ ∪ –succ(j)″ by auto
}
then have –n∈nat. ⟨n,succ(j)⟩∈Le″ ⊆–n∈nat. ⟨n,j⟩∈Le″ ∪ –succ(j)″ by auto
moreover have Finite(–n∈nat. ⟨n,j⟩∈Le″ ∪ –succ(j)″) using as(2) Finite˙cons by auto
ultimately have Finite(–n∈nat. ⟨n,succ(j)⟩∈Le″) using subset˙Finite by auto
}
then have ∀j∈nat. Finite(–n∈nat. ⟨n,j⟩∈Le″) ⟶ Finite(–n∈nat. ⟨n,succ(j)⟩∈Le″) by auto
ultimately have Finite(range(restrict(NN, –n ∈ nat . ⟨n, r⟩ ∈ Le″)))
using lepoll˙Finite[of range(restrict(NN, –n ∈ nat . ⟨n, r⟩ ∈ Le″))
–n ∈ nat . ⟨n, r⟩ ∈ Le″] ind˙on˙nat[OF ‹r∈nat›, where P=λt. Finite(–n∈nat. ⟨n,t⟩∈Le″)] by auto
then have Finite((restrict(NN, –n ∈ nat . ⟨n, r⟩ ∈ Le″))–n ∈ nat . ⟨n, r⟩ ∈ Le″) using range˙image˙domain[OF NresFun]
by auto
then have Finite(NN–n ∈ nat . ⟨n, r⟩ ∈ Le″) using restrict˙image by auto
then have (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)–is in the spectrum of″(λT. (⋃T)–is compact in″T) using compact˙spectrum by auto
moreover have ⋃(T–restricted to″NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)=⋃T∩NN–n ∈ nat . ⟨n, r⟩ ∈ Le″
unfolding RestrictedTo˙def by auto moreover
have ⋃T∩NN–n ∈ nat . ⟨n, r⟩ ∈ Le″=NN–n ∈ nat . ⟨n, r⟩ ∈ Le″
using func1˙1˙L6(2)[OF R] by blast
moreover have (T–restricted to″NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)–is a topology″
using Top˙1˙L4 unfolding topology0˙def by auto
ultimately have (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)–is compact in″(T–restricted to″NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)
unfolding Spec˙def by force
then have (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)–is compact in″(T) using compact˙subspace˙imp˙compact by auto
moreover from Acov(1) have (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)⊆⋃M by auto

moreover note Acov(2) ultimately
obtain $\mathfrak{N}$ where $\mathfrak{N}$:$\mathfrak{N}$∈FinPow(M) (NN–n ∈ nat . ⟨n, r⟩ ∈ Le″)⊆⋃$\mathfrak{N}$
unfolding IsCompact˙def by blast
from $\mathfrak{N}$(1) have $\mathfrak{N}$ ∪–V″∈FinPow(M) using V(2) unfolding FinPow˙def by auto moreover
{
fix s assume s:s∈A s∉V
with V(1) have s∈NNnat by auto
then have s∈–NNn. n∈nat″ using func˙imagedef[OF NFun] by auto
then obtain n where n:n∈nat s=NNn by auto
{
assume ⟨r,n⟩∈Le
with rr have NNn∈V by auto
with n(2) s(2) have False by auto
}
then have ⟨r,n⟩∉Le by auto
with rr(1) n(1) have ¬(r≤n) by auto
then have n≤r using Ord˙linear˙le[where thesis=⟨n,r⟩∈Le] nat˙into˙Ord[OF rr(1)]
nat˙into˙Ord[OF n(1)] by auto
with rr(1) n(1) have ⟨n,r⟩∈Le by auto
with n(2) have s∈–NNt. t∈–n∈nat. ⟨n,r⟩∈Le″″ by auto moreover
have –n∈nat. ⟨n,r⟩∈Le″⊆nat by auto
ultimately have s∈NN–n∈nat. ⟨n,r⟩∈Le″ using func˙imagedef[OF NFun]
by auto
with $\mathfrak{N}$(2) have s∈⋃$\mathfrak{N}$ by auto
}
then have A⊆⋃$\mathfrak{N}$ ∪ V by auto
then have A⊆⋃($\mathfrak{N}$ ∪ –V″) by auto ultimately
have ∃$\mathfrak{N}$∈FinPow(M). A⊆⋃$\mathfrak{N}$ by auto
}
then have ∀M∈Pow(T). A⊆⋃M ⟶ (∃$\mathfrak{N}$∈FinPow(M). A⊆⋃$\mathfrak{N}$) by auto moreover
have ss:A⊆⋃(T) using func1˙1˙L6(2)[OF R] x by blast ultimately
have A–is compact in″(T) unfolding IsCompact˙def by auto moreover
with assms have A–is closed in″(T) unfolding IsKC˙def IsCompact˙def by auto ultimately
have A∈–B∈Pow(⋃T). B–is compact in″(T)∧B–is closed in″(T)″ using ss by auto
then have –⋃T″∪(⋃T-A)∈(–one-point compactification of″T) unfolding OP-Compactification˙def
by auto
then have –⋃T″∪(⋃T-A)=Interior(–⋃T″∪(⋃T-A),–one-point compactification of″T) using topology0.Top˙2˙L3 op˙comp˙is˙top
unfolding topology0˙def by auto moreover
{
assume y∈A
with A(4) have y∈NNnat by auto
then have y∈–NNn. n∈nat″ using func˙imagedef[OF NFun] by auto

then obtain n where n∈natNNn=y by auto
with noy y have False by auto
}
with y have y∈–⋃T″∪(⋃T-A) by force ultimately
have y∈Interior(–⋃T″∪(⋃T-A),–one-point compactification of″T) –⋃T″∪(⋃T-A)∈Pow(⋃(–one-point compactification of″T))
using op˙compact˙total by auto moreover
have (∀U∈Pow(⋃(–one-point compactification of″T)). y ∈ Interior(U,–one-point compactification of″T) ⟶ (∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ U))
using A(3) dom topology0.NetConverges˙def[OF ˙ NET] op˙comp˙is˙top unfolding topology0˙def by auto
ultimately have ∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N m ∈ –⋃T″∪(⋃T-A) by blast
then obtain r where r˙def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈–⋃T″∪(⋃T-A) by auto
{
fix s assume AA:⟨r,s⟩∈Le
with reg obtain m where Nm≠⋃T ⟨s,m⟩∈Le by auto
then have ⟨s,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨s,m⟩∈Le m]
nat˙into˙Ord by auto
with AA have ⟨r,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using le˙trans by auto
with r˙def(2) have N(μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le)∈–⋃T″∪(⋃T-A) by auto
then have NNs∈–⋃T″∪(⋃T-A) using apply˙equality[OF ˙ NFun] AA by auto
with noy have NNs∈(⋃T-A) using AA by auto
moreover have NNs∈–NNt. t∈nat″ using AA by auto
then have NNs∈NNnat using func˙imagedef[OF NFun] by auto
then have NNs∈A by auto
ultimately have False by auto
}
moreover have r⊆succ(r) using succ˙explained by auto
then have r≤succ(r) using subset˙imp˙le nat˙into˙Ord ‹r∈nat› nat˙succI
by auto
then have ⟨r,succ(r)⟩∈Le using ‹r∈nat› nat˙succI by auto
ultimately have False by auto
}
ultimately have False by auto
}
then have ∀N x y. N:nat→(⋃–one-point compactification of″T) ∧ (⟨N,Le⟩→$_N$ x–in″(–one-point compactification of″T))
∧ (⟨N,Le⟩→$_N$ y–in″(–one-point compactification of″T)) ⟶ x=y by auto
then show thesis unfolding IsUS˙def by auto
qed

In the one-point compactification of an anti-compact space, ever subspace that contains the infinite point is compact.

theorem (in topology0) anti˙comp˙imp˙OP˙inf˙comp:
assumes T–is anti-compact″ A⊆⋃(–one-point compactification of″T) ⋃T∈A
shows A–is compact in″(–one-point compactification of″T)

proof-
{
fix M assume M:M⊆(–one-point compactification of″T) A⊆⋃M
with assms(3) obtain U where U:⋃T∈U U∈M by auto
with M(1) obtain K where K:K–is compact in″T K–is closed in″T U=–⋃T″∪(⋃T-K)
unfolding OPCompactification˙def using mem˙not˙refl[of ⋃T] by auto
from K(1) have K–is compact in″(T–restricted to″K) using compact˙imp˙compact˙subspace Compact˙is˙card˙nat
by auto
moreover have ⋃(T–restricted to″K)=⋃T∩K unfolding RestrictedTo˙def by auto
with K(1) have ⋃(T–restricted to″K)=K unfolding IsCompact˙def by auto ultimately
have (⋃(T–restricted to″K))–is compact in″(T–restricted to″K) by auto
with assms(1) have K–is in the spectrum of″(λT. (⋃T)–is compact in″T) unfolding IsAntiComp˙def
antiProperty˙def using K(1) unfolding IsCompact˙def by auto
then have finK:Finite(K) using compact˙spectrum by auto
from assms(2) have A-U⊆(⋃T ∪–⋃T″) -U using op˙compact˙total by auto
with K(3) have A-U⊆K by auto
with finK have Finite(A-U) using subset˙Finite by auto
then have (A-U)–is in the spectrum of″(λT. (⋃T)–is compact in″T) using compact˙spectrum by auto moreover
have ⋃((–one-point compactification of″T)–restricted to″(A-U))=A-U unfolding RestrictedTo˙def using assms(2) K(3)
op˙compact˙total by auto moreover
have ((–one-point compactification of″T)–restricted to″(A-U))–is a topology″ using topology0.Top˙1˙L4
op˙comp˙is˙top unfolding topology0˙def by auto
ultimately have (A-U)–is compact in″((–one-point compactification of″T)–restricted to″(A-U))
unfolding Spec˙def by auto
then have (A-U)–is compact in″(–one-point compactification of″T) using compact˙subspace˙imp˙compact by auto
moreover have A-U⊆⋃M using M(2) by auto moreover
note M(1) ultimately obtain N where N:N∈FinPow(M) A-U⊆⋃N unfolding IsCompact˙def by blast
from N(1) U(2) have N ∪–U″∈FinPow(M) unfolding FinPow˙def by auto moreover
from N(2) have A⊆⋃(N ∪–U″) by auto
ultimately have ∃R∈FinPow(M). A⊆⋃R by auto
}
then show thesis using op˙compact˙total assms(2) unfolding IsCompact˙def by auto
qed

As a last result in this section, the one-point compactification of our topology is not a KC space.

theorem extension˙pow˙OP˙not˙KC:

shows ¬((–one-point compactification of″(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″))–is KC″)
proof
have noE:csucc(nat)≠0 using Ord˙0˙lt˙csucc[OF Ord˙nat] by auto
let T=(Pow(csucc(nat)) ∪ —csucc(nat)″∪S. S∈(CoCountable csucc(nat))-–0″″)
assume ass:(–one-point compactification of″T)–is KC″
from extension˙pow˙notDiscrete have –csucc(nat)″ ∉ (Pow(csucc(nat)) ∪ —csucc(nat)″ ∪ S . S ∈ (CoCountable csucc(nat)) - –0″″)
by auto
{
assume csucc(nat)=csucc(nat)∪–csucc(nat)″ moreover
have csucc(nat)∈csucc(nat)∪–csucc(nat)″ by auto
ultimately have csucc(nat)∈csucc(nat) by auto
then have False using mem˙not˙refl by auto
}
then have dist:csucc(nat)≠csucc(nat)∪–csucc(nat)″ by blast
{
assume –csucc(nat)″∈(–one-point compactification of″(Pow(csucc(nat)) ∪ —csucc(nat)″ ∪ S . S ∈ (CoCountable csucc(nat)) - –0″″))
then have –csucc(nat)″∈—⋃T″∪((⋃T)-K). K∈–B∈Pow(⋃T). B–is compact in″T ∧ B–is closed in″T″″
unfolding OPCompactification˙def using extension˙pow˙notDiscrete by auto
then obtain K where –csucc(nat)″=–⋃T″∪((⋃T)-K) by auto moreover
have ⋃T∈–⋃T″∪((⋃T)-K) by auto
ultimately have ⋃T∈–csucc(nat)″ by auto
with dist have False using extension˙pow˙union by auto
}
then have –csucc(nat)″∉(–one-point compactification of″T) by auto moreover
have ⋃(–one-point compactification of″T)-(⋃(–one-point compactification of″T)-–csucc(nat)″)=–csucc(nat)″ using extension˙pow˙union
topology0.op˙compact˙total unfolding topology0˙def using extension˙pow˙top by auto ultimately
have di:⋃(–one-point compactification of″T)-(⋃(–one-point compactification of″T)-–csucc(nat)″)∉(–one-poin compactification of″T) by auto
{
assume (⋃(–one-point compactification of″T)-–csucc(nat)″)–is closed in″(–one-point compactification of″T)
then have ⋃(–one-point compactification of″T)-(⋃(–one-point compactification of″T)-–csucc(nat)″)∈(–one-point compactification of″T) unfolding IsClosed˙def by auto
with di have False by auto
}
then have n:¬((⋃(–one-point compactification of″T)-–csucc(nat)″)–is closed in″(–one-point compactification of″T)) by auto moreover
from dist have ⋃T∈(⋃(–one-point compactification of″T)-–csucc(nat)″) using topology0.op˙compact˙total unfolding topology0˙def using extension˙pow˙top extension˙pow˙union by auto
then have (⋃(–one-point compactification of″T)-–csucc(nat)″)–is compact in″(–one-point compactification of″T) using topology0.anti˙comp˙imp˙OP˙inf˙comp[of T

(⋃(–one-point compactification of″T)-–csucc(nat)″)] unfolding topology0˙def using extension˙pow˙antiCompact extension˙pow˙top by auto
with ass have (⋃(–one-point compactification of″T)-–csucc(nat)″)–is closed in″(–one-point compactification of″T) unfolding IsKC˙def by auto
with n show False by auto
qed

In conclusion, $US \not\Rightarrow KC$.

68.8 Other types of properties

In this section we will define new properties that aren't defined as anti-properties and that are not separation axioms. In some cases we will consider their anti-properties.

68.9 Definitions

A space is called perfect if it has no isolated points. This definition may vary in the literature to similar, but not equivalent definitions.

definition
IsPerf (˙ –is perfect″) where
T–is perfect″ ≡ ∀x∈⋃T. –x″∉T

An anti-perfect space is called scattered.

definition
IsScatt (˙ –is scattered″) where
T–is scattered″ ≡ T–is anti-″IsPerf

A topological space with two disjoint dense subspaces is called resolvable.

definition
IsRes (˙ –is resolvable″) where
T–is resolvable″ ≡ ∃U∈Pow(⋃T). ∃V∈Pow(⋃T). Closure(U,T)=⋃T ∧ Closure(V,T)=⋃T ∧ U∩V=0

A topological space where every dense subset is open is called submaximal.

definition
IsSubMax (˙ –is submaximal″) where
T–is submaximal″ ≡ ∀U∈Pow(⋃T). Closure(U,T)=⋃T ⟶ U∈T

A subset of a topological space is nowhere-dense if the interior of its closure is empty.

definition
IsNowhereDense (˙ –is nowhere dense in″ ˙) where
A–is nowhere dense in″T ≡ A⊆⋃T ∧ Interior(Closure(A,T),T)=0

A topological space is then a Luzin space if every nowhere-dense subset is countable.

definition
IsLuzin (˙ –is luzin″) where
T–is luzin″ ≡ ∀A∈Pow(⋃T). (A–is nowhere dense in″T) ⟶ A≲nat

An also useful property is local-connexion.

definition
IsLocConn (˙–is locally-connected″) where
T–is locally-connected″ ≡ T–is locally″(λT. λB. ((T–restricted to″B)–is connected″))

An SI-space is an anti-resolvable perfect space.

definition
IsAntiRes (˙–is anti-resolvable″) where
T–is anti-resolvable″ ≡ T–is anti-″IsRes

definition
IsSI (˙–is Strongly Irresolvable″) where
T–is Strongly Irresolvable″ ≡ (T–is anti-resolvable″) ∧ (T–is perfect″)

68.10 First examples

Firstly, we need to compute the spectrum of the being perfect.

lemma spectrum˙perfect:
shows (A–is in the spectrum of″IsPerf) ⟷ A=0
proof
assume A–is in the spectrum of″IsPerf
then have Pow(A)–is perfect″ unfolding Spec˙def using Pow˙is˙top by auto
then have ∀b∈A. –b″∉Pow(A) unfolding IsPerf˙def by auto
then show A=0 by auto
next
assume A:A=0
{
fix T assume T:T–is a topology″ ⋃T≈A
with T(2) A have ⋃T≈0 by auto
then have ⋃T=0 using eqpoll˙0˙is˙0 by auto
then have T–is perfect″ unfolding IsPerf˙def by auto
}
then show A–is in the spectrum of″IsPerf unfolding Spec˙def by auto
qed

The discrete space is clearly scattered:

lemma pow˙is˙scattered:
shows Pow(A)–is scattered″
proof-
{
fix B assume B:B⊆⋃Pow(A) (Pow(A)–restricted to″B)–is perfect″
from B(1) have Pow(A)–restricted to″B=Pow(B) unfolding RestrictedTo˙def by blast

```
    with B(2) have Pow(B)–is perfect″ by auto
    then have ∀b∈B. –b″∉Pow(B) unfolding IsPerf˙def by auto
    then have B=0 by auto
  }
  then show thesis using spectrum˙perfect unfolding IsScatt˙def antiProperty˙def by
auto
qed
```

The trivial topology is perfect, if it is defined over a set with more than one point.

```
lemma trivial˙is˙perfect:
  assumes ∃x y. x∈X ∧ y∈X ∧ x≠y
  shows –0,X″–is perfect″
proof-
  {
    fix r assume –r″∈–0,X″
    then have X=–r″ by auto
    with assms have False by auto
  }
  then show thesis unfolding IsPerf˙def by auto
qed
```

The trivial topology is resolvable, if it is defined over a set with more than one point.

```
lemma trivial˙is˙resolvable:
  assumes ∃x y. x∈X ∧ y∈X ∧ x≠y
  shows –0,X″–is resolvable″
proof-
  from assms obtain x y where xy:x∈X y∈X x≠y by auto
  {
    fix A assume A:A–is closed in″–0,X″ A⊆X
    then have X-A∈–0,X″ unfolding IsClosed˙def by auto
    then have X-A=0∨X-A=X by auto
    with A(2) have A=X∨X-A=X by auto moreover
    {
      assume X-A=X
      then have X-(X-A)=0 by auto
      with A(2) have A=0 by auto
    }
    ultimately have A=X∨A=0 by auto
    then have A=0∨A=X by auto
  }
  then have cl:∀A∈Pow(X). A–is closed in″–0,X″ ⟶ A=0∨A=X by auto
  from xy(3) have –x″∩–y″=0 by auto moreover
  {
    have –X″–is a partition of″X using indiscrete˙partition xy(1) by auto
    then have top:topology0(PTopology X –X″) using topology0˙ptopology by auto
    have X≠0 using xy(1) by auto
    then have (PTopology X –X″)=–0,X″ using indiscrete˙ptopology[of X] by auto
```

```
    with top have top0:topology0(–0,X″) by auto
    then have x∈Closure(–x″,–0,X″) using topology0.cl˙contains˙set xy(1) by auto
moreover
    have Closure(–x″,–0,X″) –is closed in″–0,X″ using topology0.cl˙is˙closed top0
xy(1) by auto
    moreover note cl
    moreover have Closure(–x″,–0,X″)⊆X using topology0.Top˙3˙L11(1) top0 xy(1)
by auto
    ultimately have Closure(–x″,–0,X″)=X by auto
  }
  moreover
  {
    have –X″–is a partition of″X using indiscrete˙partition xy(1) by auto
    then have top:topology0(PTopology X –X″) using topology0˙ptopology by auto
    have X≠0 using xy(1) by auto
    then have (PTopology X –X″)=–0,X″ using indiscrete˙ptopology[of X] by auto
    with top have top0:topology0(–0,X″) by auto
    then have y∈Closure(–y″,–0,X″) using topology0.cl˙contains˙set xy(2) by auto
moreover
    have Closure(–y″,–0,X″) –is closed in″–0,X″ using topology0.cl˙is˙closed top0
xy(2) by auto
    moreover note cl
    moreover have Closure(–y″,–0,X″)⊆X using topology0.Top˙3˙L11(1) top0 xy(2)
by auto
    ultimately have Closure(–y″,–0,X″)=X by auto
  }
  ultimately show thesis using xy(1,2) unfolding IsRes˙def by auto
qed
```

The spectrum of Luzin spaces is the class of countable sets, so there are lots of examples of Luzin spaces.

```
lemma spectrum˙Luzin:
  shows (A–is in the spectrum of″IsLuzin) ⟷ A≲nat
proof
  assume A:A–is in the spectrum of″IsLuzin
  {
    assume A=0
    then have A≲nat using empty˙lepollI by auto
  }
  moreover
  {
    assume A≠0
    then obtain x where x:x∈A by auto
    {
      fix M assume M⊆–0,–x″,A″
      then have ⋃M∈–0,–x″,A″ using x by blast
    }
    moreover
    {
```

fix U V assume U∈–0,–x″,A″ V∈–0,–x″,A″
then have U∩V∈–0,–x″,A″ by auto
}
ultimately have top:–0,–x″,A″–is a topology″ unfolding IsATopology˙def by auto
moreover have tot:⋃–0,–x″,A″=A using x by auto
moreover note A ultimately have luz:–0,–x″,A″–is luzin″ unfolding Spec˙def by auto
moreover have –x″∈–0,–x″,A″ by auto
then have ((⋃–0,–x″,A″)-–x″)–is closed in″–0,–x″,A″ using topology0.Top˙3˙L9
unfolding topology0˙def using top by blast
then have (A-–x″)–is closed in″–0,–x″,A″ using tot by auto
then have Closure(A-–x″,–0,–x″,A″)=A-–x″ using tot top topology0.Top˙3˙L8[of –0,–x″,A″]
unfolding topology0˙def by auto
then have B:Interior(Closure(A-–x″,–0,–x″,A″),–0,–x″,A″)=Interior(A-–x″,–0,–x″,A″) by auto
then have C:Interior(Closure(A-–x″,–0,–x″,A″),–0,–x″,A″)⊆A-–x″ using top topology0.Top˙2˙L1
unfolding topology0˙def by auto
then have D:Interior(Closure(A-–x″,–0,–x″,A″),–0,–x″,A″)∈–0,–x″,A″ using topology0.Top˙2˙L2
unfolding topology0˙def using top by auto
from x have ¬(A⊆A-–x″) by auto
with C D have Interior(Closure(A-–x″,–0,–x″,A″),–0,–x″,A″)=0 by auto
then have (A-–x″)–is nowhere dense in″–0,–x″,A″ unfolding IsNowhereDense˙def using tot
by auto
with luz have A-–x″≲nat unfolding IsLuzin˙def using tot by auto
then have U1:A-–x″≺csucc(nat) using Card˙less˙csucc˙eq˙le[OF Card˙nat] by auto
have –x″≈1 using singleton˙eqpoll˙1 by auto
then have –x″≺nat using n˙lesspoll˙nat eq˙lesspoll˙trans by auto
then have –x″≲nat using lesspoll˙imp˙lepoll by auto
then have U2:–x″≺csucc(nat) using Card˙less˙csucc˙eq˙le[OF Card˙nat] by auto
with U1 have U:(A-–x″)∪–x″≺csucc(nat) using less˙less˙imp˙un˙less[OF ˙ ˙ InfCard˙csucc[OF InfCard˙nat]]
by auto
have (A-–x″)∪–x″=A using x by auto
with U have A≺csucc(nat) by auto
then have A≲nat using Card˙less˙csucc˙eq˙le[OF Card˙nat] by auto
}
ultimately
show A≲nat by auto
next
assume A:A≲nat
{
fix T assume T:T–is a topology″ ⋃T≈A
{
fix B assume B⊆⋃T B–is nowhere dense in″T
then have B≲⋃T using subset˙imp˙lepoll by auto

```
    with T(2) have B≲A using lepoll˙eq˙trans by auto
    with A have B≲nat using lepoll˙trans by blast
  }
  then have ∀B∈Pow(⋃T). (B–is nowhere dense in″T) ⟶ B≲nat by auto
  then have T–is luzin″ unfolding IsLuzin˙def by auto
 }
 then show A–is in the spectrum of″IsLuzin unfolding Spec˙def by auto
qed
```

68.11 Structural results

Every resolvable space is also perfect.

```
theorem (in topology0) resolvable˙imp˙perfect:
 assumes T–is resolvable″
 shows T–is perfect″
proof-
 {
  assume ¬(T–is perfect″)
  then obtain x where x:x∈⋃T –x″∈T unfolding IsPerf˙def by auto
  then have cl:(⋃T-–x″)–is closed in″T using Top˙3˙L9 by auto
  from assms obtain U V where UV:U⊆⋃T V⊆⋃T cl(U)=⋃T cl(V)=⋃T U∩V=0
unfolding IsRes˙def by auto
  {
   fix W assume x∉W W⊆⋃T
   then have W⊆⋃T-–x″ by auto
   then have cl(W)⊆⋃T-–x″ using cl Top˙3˙L13 by auto
   with x(1) have ¬(⋃T⊆cl(W)) by auto
   then have ¬(cl(W)=⋃T) by auto
  }
  with UV have False by auto
 }
 then show thesis by auto
qed
```

The spectrum of being resolvable follows:

```
corollary spectrum˙resolvable:
 shows (A–is in the spectrum of″IsRes) ⟷ A=0
proof
 assume A:A–is in the spectrum of″IsRes
 have ∀T. T–is a topology″ ⟶ IsRes(T) ⟶ IsPerf(T) using topology0.resolvable˙imp˙perfect

  unfolding topology0˙def by auto
 with A have A–is in the spectrum of″IsPerf using P˙imp˙Q˙spec˙inv[of IsRes IsPerf]
by auto
 then show A=0 using spectrum˙perfect by auto
next
 assume A:A=0
 {
  fix T assume T:T–is a topology″ ⋃T≈A
```

```
    with T(2) A have ⋃T≈0 by auto
    then have ⋃T=0 using eqpoll˙0˙is˙0 by auto
    then have Closure(0,T)=⋃T using topology0.Top˙3˙L2 T(1)
      topology0.Top˙3˙L8 unfolding topology0˙def by auto
    then have T–is resolvable″ unfolding IsRes˙def by auto
  }
  then show A–is in the spectrum of″IsRes unfolding Spec˙def by auto
qed
```

The cofinite space over $\mathbb{N}$ is a T_1, perfect and luzin space.

```
theorem cofinite˙nat˙perfect:
  shows (CoFinite nat)–is perfect″
proof-
  {
    fix x assume x:x∈⋃(CoFinite nat) –x″∈(CoFinite nat)
    then have xn:x∈nat using union˙cocardinal unfolding Cofinite˙def by auto
    with x(2) have nat-–x″≺nat unfolding Cofinite˙def CoCardinal˙def by auto
    moreover have Finite(–x″) by auto
    then have –x″≺nat unfolding Finite˙def using n˙lesspoll˙nat eq˙lesspoll˙trans by
auto
   ultimately have (nat-–x″)∪–x″≺nat using less˙less˙imp˙un˙less[OF ˙˙ InfCard˙nat]
by auto
    moreover have (nat-–x″)∪–x″=nat using xn by auto
    ultimately have False by auto
  }
  then show thesis unfolding IsPerf˙def by auto
qed

theorem cofinite˙nat˙luzin:
  shows (CoFinite nat)–is luzin″
proof-
  have nat–is in the spectrum of″IsLuzin using spectrum˙Luzin by auto moreover
  have ⋃(CoFinite nat)=nat using union˙cocardinal unfolding Cofinite˙def by auto
   moreover have (CoFinite nat)–is a topology″ unfolding Cofinite˙def using Co-
Car˙is˙topology[OF InfCard˙nat]
    by auto
  ultimately show thesis unfolding Spec˙def by auto
qed
```

The cocountable topology on $\mathbb{N}^+$ or csucc(nat) is also T_1, perfect and luzin; but defined on a set not in the spectrum.

```
theorem cocountable˙csucc˙nat˙perfect:
  shows (CoCountable csucc(nat))–is perfect″
proof-
  have noE:csucc(nat)≠0 using lt˙csucc[OF Ord˙nat] by auto
  {
    fix x assume x:x∈⋃(CoCountable csucc(nat)) –x″∈(CoCountable csucc(nat))
   then have xn:x∈csucc(nat) using union˙cocardinal noE unfolding Cocountable˙def
by auto
```

```
    with x(2) have csucc(nat)-–x″≺csucc(nat) unfolding Cocountable˙def CoCardi-
nal˙def by auto
    moreover have Finite(–x″) by auto
    then have –x″≺nat unfolding Finite˙def using n˙lesspoll˙nat eq˙lesspoll˙trans by
auto
    then have –x″≲nat using lesspoll˙imp˙lepoll by auto
    then have –x″≺csucc(nat) using Card˙less˙csucc˙eq˙le[OF Card˙nat] by auto
    ultimately have (csucc(nat)-–x″)∪–x″≺csucc(nat) using less˙less˙imp˙un˙less[OF
˙ ˙ InfCard˙csucc[OF InfCard˙nat]] by auto
    moreover have (csucc(nat)-–x″)∪–x″=csucc(nat) using xn by auto
    ultimately have False by auto
  }
  then show thesis unfolding IsPerf˙def by auto
qed

theorem cocountable˙csucc˙nat˙luzin:
  shows (CoCountable csucc(nat))–is luzin″
proof-
  have noE:csucc(nat)≠0 using lt˙csucc[OF Ord˙nat] by auto
  {
   fix B assume B:B∈Pow(⋃(CoCountable csucc(nat))) B–is nowhere dense in″(CoCountable
csucc(nat)) ¬(B≲nat)
     from B(1) have B⊆csucc(nat) using union˙cocardinal noE unfolding Cocount-
able˙def by auto moreover
     from B(3) have ¬(B≺csucc(nat)) using Card˙less˙csucc˙eq˙le[OF Card˙nat] by
auto ultimately
   have Closure(B,CoCountable csucc(nat))=csucc(nat) using closure˙set˙cocardinal
noE unfolding Cocountable˙def by auto
   then have Interior(Closure(B,CoCountable csucc(nat)),CoCountable csucc(nat))=Interior(csucc(nat),CoCoun
csucc(nat)) by auto
     with B(2) have 0=Interior(csucc(nat),CoCountable csucc(nat)) unfolding Is-
NowhereDense˙def by auto moreover
    have csucc(nat)-csucc(nat)=0 by auto
   then have csucc(nat)-csucc(nat)≺csucc(nat) using empty˙lepollI Card˙less˙csucc˙eq˙le[OF
Card˙nat] by auto
     then have Interior(csucc(nat),CoCountable csucc(nat))=csucc(nat) using inte-
rior˙set˙cocardinal noE unfolding Cocountable˙def by auto
    ultimately have False using noE by auto
  }
 then have ∀B∈Pow(⋃(CoCountable csucc(nat))). (B–is nowhere dense in″(CoCountable
csucc(nat))) ⟶ B≲nat by auto
  then show thesis unfolding IsLuzin˙def by auto
qed
```

The existence of T_2, uncountable, perfect and luzin spaces is unprovable in ZFC. It is related to the CH and Martin's axiom.

```
end
```

69 Metric spaces

theory MetricSpace˙ZF imports Topology˙ZF˙1 OrderedGroup˙ZF Lattice˙ZF
begin

A metric space is a set on which a distance between points is defined as a function $d : X \times X \to [0, \infty)$. With this definition each metric space is a topological space which is paracompact and Hausdorff (T_2), hence normal (in fact even perfectly normal).

69.1 Pseudometric - definition and basic properties

A metric on X is usually defined as a function $d : X \times X \to [0, \infty)$ that satisfies the conditions $d(x, x) = 0$, $d(x, y) = 0 \Rightarrow x = y$ (identity of indiscernibles), $d(x, y) = d(y, x)$ (symmetry) and $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality) for all $x, y \in X$. Here we are going to be a bit more general and define metric and pseudo-metric as a function valued in an ordered group.

First we define a pseudo-metric, which has the axioms of a metric, but without the second part of the identity of indiscernibles. In our definition IsApseudoMetric is a predicate on five sets: the function d, the set X on which the metric is defined, the group carrier G, the group operation A and the order r on G.

definition
IsApseudoMetric(d,X,G,A,r) ≡ d:X×X → Nonnegative(G,A,r)
∧ (∀x∈X. d⟨x,x⟩ = TheNeutralElement(G,A))
∧ (∀x∈X.∀y∈X. d⟨x,y⟩ = d⟨y,x⟩)
∧ (∀x∈X.∀y∈X.∀z∈X. ⟨d⟨x,z⟩, A⟨d⟨x,y⟩,d⟨y,z⟩⟩⟩ ∈ r)

We add the full axiom of identity of indiscernibles to the definition of a pseudometric to get the definition of metric.

definition
IsAmetric(d,X,G,A,r) ≡
IsApseudoMetric(d,X,G,A,r) ∧ (∀x∈X.∀y∈X. d⟨x,y⟩ = TheNeutralElement(G,A)
⟶ x=y)

A disk is defined as set of points located less than the radius from the center.

definition Disk(X,d,r,c,R) ≡ –x∈X. ⟨d⟨c,x⟩,R⟩ ∈ StrictVersion(r)″

Next we define notation for metric spaces. We will use additive notation for the group operation but we do not assume that the group is abelian. Since for many theorems it is sufficient to assume the pseudometric axioms we will assume in this context that the sets d, X, G, A, r form a pseudometric raher than a metric.

locale pmetric˙space =

fixes d and X and G and A and r

assumes ordGroupAssum: IsAnOrdGroup(G,A,r)

assumes pmetricAssum: IsApseudoMetric(d,X,G,A,r)

fixes zero (**0**)
defines zero˙def [simp]: **0** ≡ TheNeutralElement(G,A)

fixes grop (infixl + 80)
defines grop˙def [simp]: x+y ≡ A⟨x,y⟩

fixes grinv (- ˙ 89)
defines grinv˙def [simp]: (-x) ≡ GroupInv(G,A)(x)

fixes lesseq (infix ≤ 68)
defines lesseq˙def [simp]: x ≤ y ≡ ⟨ x,y⟩ ∈ r

fixes sless (infix ¡ 68)
defines sless˙def [simp]: x ¡ y ≡ x≤y ∧ x≠y

fixes nonnegative (G^+)
defines nonnegative˙def [simp]: G^+ ≡ Nonnegative(G,A,r)

fixes positive (G_+)
defines positive˙def [simp]: G_+ ≡ PositiveSet(G,A,r)

fixes setinv (- ˙ 72)
defines setninv˙def [simp]: -C ≡ GroupInv(G,A)(C)

fixes abs
defines abs˙def [simp]: abs(x) ≡ AbsoluteValue(G,A,r)(x)

fixes oddext (˙ °)
defines oddext˙def [simp]: f° ≡ OddExtension(G,A,r,f)

fixes disk
defines disk˙def [simp]: disk(c,R) ≡ Disk(X,d,r,c,R)

The theorems proven the in the group3 locale are valid in the pmetric˙space locale.

sublocale
pmetric˙space ¡ group3 G A r zero grop grinv lesseq sless nonnegative positive setinv abs oddext
using ordGroupAssum unfolding group3˙def by auto

The theorems proven the in the group0 locale are valid in the pmetric˙space locale.

sublocale pmetric˙space ¡ group0 G A zero grop grinv

using ordGroupAssum unfolding group0˙def IsAnOrdGroup˙def by auto

The next lemma shows the definition of the pseudometric in the notation used in the metric˙space context.

lemma (in pmetric˙space) pmetric˙properties: shows
d: X×X → G^+
∀x∈X. d⟨x,x⟩ = **0**
∀x∈X.∀y∈X. d⟨x,y⟩ = d⟨y,x⟩
∀x∈X.∀y∈X.∀z∈X. d⟨x,z⟩ ≤ d⟨x,y⟩ + d⟨y,z⟩
using pmetricAssum unfolding IsApseudoMetric˙def by auto

The definition of the disk in the notation used in the pmetric˙space context:

lemma (in pmetric˙space) disk˙definition: shows disk(c,R) = –x∈X. d⟨c,x⟩ ¡ R″
proof -
have disk(c,R) = Disk(X,d,r,c,R) by simp
then have disk(c,R) = –x∈X. ⟨d⟨c,x⟩,R⟩ ∈ StrictVersion(r)″ unfolding Disk˙def by simp
moreover have ∀x∈X. ⟨d⟨c,x⟩,R⟩ ∈ StrictVersion(r) ⟷ d⟨c,x⟩ ¡ R
using def˙of˙strict˙ver by simp
ultimately show thesis by auto
qed

If the radius is positive then the center is in disk.

lemma (in pmetric˙space) center˙in˙disk: assumes c∈X and R∈G_+ shows c ∈ disk(c,R)
using pmetricAssum assms IsApseudoMetric˙def PositiveSet˙def disk˙definition by simp

A technical lemma that allows us to shorten some proofs:

lemma (in pmetric˙space) radius˙in˙group: assumes c∈X and x ∈ disk(c,R)
shows R∈G **0**¡R R∈G_+ ((-d⟨c,x⟩) + R) ∈ G_+
proof -
from assms(2) have x∈X and d⟨c,x⟩ ¡ R using disk˙definition by auto
with assms(1) show **0**¡R using pmetric˙properties(1) apply˙funtype
OrderedGroup˙ZF˙1˙L2 group˙strict˙ord˙transit by blast
then show R∈G and R∈G_+ using OrderedGroup˙ZF˙1˙L2A less˙are˙members by auto
from assms(1) ‹x∈X› have d⟨c,x⟩ ∈ G
using pmetric˙properties(1) apply˙funtype OrderedGroup˙ZF˙1˙L4E by auto
then have (-d⟨c,x⟩) ∈ G using inverse˙in˙group by simp
with ‹R∈G› ‹d⟨c,x⟩ ¡ R› have (-d⟨c,x⟩) + d⟨c,x⟩ ¡ (-d⟨c,x⟩) + R
using group˙strict˙ord˙transl˙inv(2) by simp
with ‹d⟨c,x⟩ ∈ G› show ((-d⟨c,x⟩) + R) ∈ G_+
using group0˙2˙L6 OrderedGroup˙ZF˙1˙L2A less˙are˙members by auto
qed

If a point x is inside a disk B and $m \leq R - d(c, x)$ then the disk centered at the point x and with radius m is contained in the disk B.

lemma (in pmetric˙space) disk˙in˙disk:

```
  assumes c∈X  and x ∈ disk(c,R) and m ≤ (-d⟨c,x⟩) + R
  shows disk(x,m) ⊆ disk(c,R)
proof
  fix y assume y ∈ disk(x,m)
  then have d⟨x,y⟩ ¡ m using disk˙definition by simp
   from assms(1,2) ‹y ∈ disk(x,m)› have R∈G x∈X y∈X using radius˙in˙group(1)
disk˙definition
    by auto
  with assms(1) have d⟨c,y⟩ ≤ d⟨c,x⟩ + d⟨x,y⟩ using pmetric˙properties(4) by simp
  from assms(1) ‹x∈X› have d⟨c,x⟩ ∈ G
    using pmetric˙properties(1) apply˙funtype OrderedGroup˙ZF˙1˙L4E by auto
  with ‹d⟨x,y⟩ ¡ m› assms(3) have d⟨c,x⟩ + d⟨x,y⟩ ¡ d⟨c,x⟩ + ((-d⟨c,x⟩) + R)
    using OrderedGroup˙ZF˙1˙L4A group˙strict˙ord˙transl˙inv(2) by blast
  with ‹d⟨c,x⟩ ∈ G› ‹R∈G› ‹d⟨c,y⟩ ≤ d⟨c,x⟩ + d⟨x,y⟩› ‹y∈X› show y ∈ disk(c,R)
    using inv˙cancel˙two(4) group˙strict˙ord˙transit disk˙definition by simp
qed
```

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

```
lemma (in pmetric˙space) disks˙form˙base:
  assumes IsMeetSemilattice(G+,r ∩ G+×G+)
  defines B ≡ ⋃c∈X. –disk(c,R). R∈G+″
  shows B –satisfies the base condition″
proof -
  { fix U V assume U∈B V∈B
    fix x assume x∈U∩V
    have ∃W∈B. x∈W ∧ W⊆U∩V
    proof -
      from assms(2) ‹U∈B› ‹V∈B› obtain cU  cV RU  RV
       where cU ∈ X RU ∈ G+ cV ∈ X RV ∈ G+ U = disk(cU,RU) V = disk(cV,RV)
        by auto
      with ‹x∈U∩V› have x ∈ disk(cU,RU) and x ∈ disk(cV,RV) by auto
      then have x∈X d⟨cU,x⟩ ¡ RU d⟨cV,x⟩ ¡ RV using disk˙definition by auto
      let mU = (- d⟨cU,x⟩) + RU
      let mV = (- d⟨cV,x⟩) + RV
        from ‹cU∈X› ‹x∈disk(cU,RU)› ‹cV∈X› ‹x∈disk(cV,RV)› have mU∈G+ and
mV∈G+
        using radius˙in˙group(4) by auto
      let m = Meet(G+,r ∩ G+×G+)⟨mU,mV⟩
      let W = disk(x,m)
      from assms(1) ‹mU ∈ G+› ‹mV ∈ G+› have  ⟨m,mU⟩ ∈ r ∩ G+×G+
        using meet˙val(3) by blast
      moreover from assms(1) ‹mU ∈ G+› ‹mV ∈ G+› have  ⟨m,mV⟩ ∈ r ∩ G+×G+
        using meet˙val(4) by blast
      moreover from assms(1) ‹mU ∈ G+› ‹mV ∈ G+› have m ∈ G+
```

using meet˙val(1) by simp
ultimately have m ∈ G_+ m ≤ m_U m ≤ m_V by auto
with ‹c_U ∈ X› ‹x ∈ disk(c_U,R_U)› ‹c_V ∈ X› ‹x ∈ disk(c_V,R_V)› ‹U = disk(c_U,R_U)› ‹V = disk(c_V,R_V)›
have W ⊆ U∩V using disk˙in˙disk by blast
moreover from assms(2) ‹x∈X› ‹m ∈ G_+› have W ∈ B and x∈W using center˙in˙disk
by auto
ultimately show thesis by auto
qed
} then show thesis unfolding SatisfiesBaseCondition˙def by auto
qed

Unions of disks form a topology, hence (pseudo)metric spaces are topological spaces. We have to add the assumption that the positive set is not empty. This is necessary to show that we can cover the space with disks and it does not look like it follows from anything we have assumed so far.

theorem (in pmetric˙space) pmetric˙is˙top:
assumes IsMeetSemilattice(G_+,r ∩ G_+×G_+) G_+≠0
defines B ≡ ⋃c∈X. –disk(c,R). R∈G_+″
defines T ≡ –⋃A. A ∈ Pow(B)″
shows T –is a topology″ B –is a base for″ T ⋃T = X
proof -
from assms(1,3,4) show T –is a topology″ B –is a base for″ T
using disks˙form˙base Top˙1˙2˙T1 by auto
then have ⋃T = ⋃B using Top˙1˙2˙L5 by simp
moreover have ⋃B = X
proof
from assms(3) show ⋃B ⊆ X using disk˙definition by auto
{ fix x assume x∈X
from assms(2) obtain R where R∈G_+ by auto
with assms(3) ‹x∈X› have x ∈ ⋃B using center˙in˙disk by auto
} thus X ⊆ ⋃B by auto
qed
ultimately show ⋃T = X by simp
qed

end

70 Uniform spaces

theory UniformSpace˙ZF imports Topology˙ZF˙4a
begin

This theory defines uniform spaces and proves their basic properties.

70.1 Definition and motivation

Just like a topological space constitutes the minimal setting in which one can speak of continuous functions, the notion of uniform spaces (commonly attributed to Andr Weil) captures the minimal setting in which one can speak of uniformly continuous functions. In some sense this is a generalization of the notion of metric (or metrizable) spaces and topological groups.

There are several definitions of uniform spaces. The fact that these definitions are equivalent is far from obvious (some people call such phenomenon cryptomorphism). We will use the definition of the uniform structure (or "uniformity") based on entourages. This was the original definition by Weil and it seems to be the most commonly used. A uniformity consists of entourages that are binary relations between points of space X that satisfy a certain collection of conditions, specified below.

```
definition
  IsUniformity (˙ –is a uniformity on″ ˙ 90) where
    Φ –is a uniformity on″ X ≡(Φ –is a filter on″ (X×X))
    ∧ (∀U∈Φ. id(X) ⊆ U ∧ (∃V∈Φ. V O V ⊆ U) ∧ converse(U) ∈ Φ)
```

If Φ is a uniformity on X, then the every element V of Φ is a certain relation on X (a subset of $X \times X$) and is called an "entourage". For an $x \in X$ we call $V\{x\}$ a neighborhood of x. The first useful fact we will show is that neighborhoods are non-empty.

```
lemma neigh˙not˙empty:
  assumes Φ –is a uniformity on″ X V∈Φ and x∈X
  shows V–x″ ≠ 0 and x ∈ V–x″
proof -
  from assms(1,2) have id(X) ⊆ V using IsUniformity˙def IsFilter˙def
    by auto
  with ‹x∈X› show x∈V–x″ and V–x″ ≠ 0 by auto
qed
```

Uniformity Φ defines a natural topology on its space X via the neighborhood system that assigns the collection $\{V(\{x\}) : V \in \Phi\}$ to every point $x \in X$. In the next lemma we show that if we define a function this way the values of that function are what they should be. This is only a technical fact which is useful to shorten the remaining proofs, usually treated as obvious in standard mathematics.

```
lemma neigh˙filt˙fun:
  assumes Φ –is a uniformity on″ X
  defines ℳ ≡ –⟨x,–V–x″.V∈Φ″⟩.x∈X″
  shows ℳ:X→Pow(Pow(X)) and ∀x∈X. ℳ(x) = –V–x″.V∈Φ″
proof -
  from assms have ∀x∈X. –V–x″.V∈Φ″ ∈ Pow(Pow(X))
    using IsUniformity˙def IsFilter˙def image˙subset by auto
```

```
  with assms show 𝓜:X→Pow(Pow(X)) using ZF˙fun˙from˙total by simp
  with assms show ∀x∈X. 𝓜(x) = –V–x″.V∈Φ″ using ZF˙fun˙from˙tot˙val
    by simp
qed
```

In the next lemma we show that the collection defined in lemma neigh˙filt˙fun is a filter on X. The proof is kind of long, but it just checks that all filter conditions hold.

```
lemma filter˙from˙uniformity:
  assumes Φ –is a uniformity on″ X and x∈X
  defines 𝓜 ≡ –⟨x,–V–x″.V∈Φ″⟩.x∈X″
  shows 𝓜(x) –is a filter on″ X
proof -
  from assms have PhiFilter: Φ –is a filter on″ (X×X) and
    𝓜:X→Pow(Pow(X)) and 𝓜(x) = –V–x″.V∈Φ″
    using IsUniformity˙def neigh˙filt˙fun by auto
  have 0 ∉ 𝓜(x)
  proof -
    from assms ‹x∈X› have 0 ∉ –V–x″.V∈Φ″ using neigh˙not˙empty by blast
    with ‹𝓜(x) = –V–x″.V∈Φ″› show 0 ∉ 𝓜(x) by simp
  qed
  moreover have X ∈ 𝓜(x)
  proof -
    note ‹𝓜(x) = –V–x″.V∈Φ″›
    moreover from assms have X×X ∈ Φ unfolding IsUniformity˙def IsFilter˙def
      by blast
    hence (X×X)–x″ ∈ –V–x″.V∈Φ″ by auto
    moreover from ‹x∈X› have (X×X)–x″ = X by auto
    ultimately show X ∈ 𝓜(x) by simp
  qed
   moreover from ‹𝓜:X→Pow(Pow(X))› ‹x∈X› have 𝓜(x) ⊆ Pow(X) using ap-
ply˙funtype
    by blast
  moreover have LargerIn: ∀B ∈ 𝓜(x). ∀C ∈ Pow(X). B⊆C ⟶ C ∈ 𝓜(x)
  proof -
  { fix B assume B ∈ 𝓜(x)
    fix C assume C ∈ Pow(X) and B⊆C
    from ‹𝓜(x) = –V–x″.V∈Φ″› ‹B ∈ 𝓜(x)› obtain U where
        U∈Φ and B = U–x″ by auto
    let V = U ∪ C×C
    from assms ‹U∈Φ› ‹C ∈ Pow(X)› have V ∈ Pow(X×X) and U⊆V
      using IsUniformity˙def IsFilter˙def by auto
    with ‹U∈Φ› PhiFilter have V∈Φ using IsFilter˙def by simp
    moreover from assms ‹U∈Φ› ‹x∈X› ‹B = U–x″› ‹B⊆C› have C = V–x″
      using neigh˙not˙empty image˙greater˙rel by simp
    ultimately have C ∈ –V–x″.V∈Φ″ by auto
    with ‹𝓜(x) = –V–x″.V∈Φ″› have C ∈ 𝓜(x) by simp
  } thus thesis by blast
  qed
```

```
  moreover have ∀A ∈ 𝓜(x).∀B ∈ 𝓜(x). A∩B ∈ 𝓜(x)
  proof -
  { fix A B assume A ∈ 𝓜(x) and B ∈ 𝓜(x)
    with ‹𝓜(x) = –V–x″.V∈Φ″› obtain V_A V_B where
      A = V_A–x″ B = V_B–x″ and  V_A ∈ Φ V_B ∈ Φ
      by auto
    let C = V_A–x″ ∩ V_B–x″
      from assms ‹V_A ∈ Φ› ‹V_B ∈ Φ› have V_A∩V_B ∈ Φ using IsUniformity˙def
IsFilter˙def
      by simp
    with ‹𝓜(x) = –V–x″.V∈Φ″› have (V_A∩V_B)–x″ ∈ 𝓜(x) by auto
      moreover from PhiFilter ‹V_A ∈ Φ› ‹V_B ∈ Φ› have C ∈ Pow(X) unfolding
IsFilter˙def
            by auto
    moreover have (V_A∩V_B)–x″ ⊆ C using image˙Int˙subset˙left by simp
    moreover note LargerIn
    ultimately have C ∈ 𝓜(x) by simp
    with ‹A = V_A–x″› ‹B = V_B–x″› have A∩B ∈ 𝓜(x) by blast
  } thus thesis by simp
  qed
  ultimately show thesis unfolding IsFilter˙def by simp
qed
```

The function defined in the premises of lemma neigh˙filt˙fun (or filter˙from˙uniformity) is a neighborhood system. The proof uses the existence of the "half-the-size" neighborhood condition (∃V∈Φ. V O V ⊆ U) of the uniformity definition, but not the converse(U) ∈ Φ part.

```
theorem neigh˙from˙uniformity:
  assumes Φ –is a uniformity on″ X
  shows –⟨x,–V–x″.V∈Φ″⟩.x∈X″ –is a neighborhood system on″ X
proof -
  let 𝓜 = –⟨x,–V–x″.V∈Φ″⟩.x∈X″
  from assms have 𝓜:X→Pow(Pow(X)) and Mval: ∀x∈X. 𝓜(x) = –V–x″.V∈Φ″
    using IsUniformity˙def neigh˙filt˙fun by auto
 moreover from assms have ∀x∈X. (𝓜(x) –is a filter on″ X) using filter˙from˙uniformity
    by simp
  moreover
  { fix x assume x∈X
    have ∀N∈𝓜(x). x∈N ∧ (∃U∈𝓜(x).∀y∈U.(N∈𝓜(y)))
    proof -
      { fix N assume N∈𝓜(x)
        have x∈N and ∃U∈𝓜(x).∀y∈U.(N ∈ 𝓜(y))
        proof -
          from ‹𝓜:X→Pow(Pow(X))› Mval ‹x∈X› ‹N∈𝓜(x)›
          obtain U where U∈Φ and N = U–x″ by auto
          with assms ‹x∈X› show x∈N using neigh˙not˙empty by simp
          from assms ‹U∈Φ› obtain V where V∈Φ  and  V O V ⊆ U
            unfolding IsUniformity˙def by auto
          let W = V–x″
```

```
        from ‹V∈Φ› Mval ‹x∈X› have W ∈ 𝓜(x) by auto
        moreover have ∀y∈W. N ∈ 𝓜(y)
        proof -
          { fix y assume y∈W
            with ‹𝓜:X→Pow(Pow(X))› ‹x∈X› ‹W ∈ 𝓜(x)› have y∈X
              using apply˙funtype by blast
            with assms have 𝓜(y) –is a filter on″ X using filter˙from˙uniformity
              by simp
            moreover from assms ‹y∈X› ‹V∈Φ› have V–y″ ∈ 𝓜(y)
              using neigh˙filt˙fun by auto
                moreover from ‹𝓜:X→Pow(Pow(X))› ‹x∈X› ‹N ∈ 𝓜(x)› have N ∈
Pow(X)
              using apply˙funtype by blast
            moreover from ‹V O V ⊆ U› ‹y∈W› have
              V–y″ ⊆ (V O V)–x″ and (V O V)–x″ ⊆ U–x″
              by auto
            with ‹N = U–x″›  have V–y″ ⊆ N by blast
            ultimately have N ∈ 𝓜(y) unfolding IsFilter˙def by simp
          } thus thesis by simp
        qed
        ultimately show ∃U∈𝓜(x).∀y∈U.(N ∈ 𝓜(y)) by auto
      qed
    } thus thesis by simp
  qed
}
ultimately show thesis unfolding IsNeighSystem˙def by simp
qed
```

When we have a uniformity Φ on X we can define a topology on X in a (relatively) natural way. We will call that topology the UniformTopology(Φ). The definition may be a bit cryptic but it just combines the construction of a neighborhood system from uniformity as in the assumptions of lemma filter˙from˙uniformity and the construction of topology from a neighborhood system from theorem topology˙from˙neighs. We could probably reformulate the definition to skip the X parameter because if Φ is a uniformity on X then X can be recovered from (is determined by) Φ.

```
definition
  UniformTopology(Φ,X) ≡ –U ∈ Pow(X). ∀x∈U. U ∈ –⟨t,–V–t″.V∈Φ″⟩.t∈X″(x)″
```

The collection of sets constructed in the UniformTopology definition is indeed a topology on X.

```
theorem uniform˙top˙is˙top:
  assumes Φ –is a uniformity on″ X
  shows
  UniformTopology(Φ,X) –is a topology″ and ⋃ UniformTopology(Φ,X) = X
  using assms  neigh˙from˙uniformity UniformTopology˙def topology˙from˙neighs
  by auto
```

end

71 More on uniform spaces

theory UniformSpace˙ZF˙1 imports func˙ZF˙1 UniformSpace˙ZF Topology˙ZF˙2
begin

This theory defines the maps to study in uniform spaces and proves their basic properties.

71.1 Uniformly continuous functions

Just as the the most general setting for continuity of functions is that of topological spaces, uniform spaces are the most general setting for the study of uniform continuity.

A map between 2 uniformities is uniformly continuous if it preserves the entourages:

```
definition
  IsUniformlyCont (˙ –is uniformly continuous between˝ ˙ –and˝ ˙ 90) where
  f:X-¿Y ==¿ Φ –is a uniformity on˝X  ==¿ Γ –is a uniformity on˝Y  ==¿
 f –is uniformly continuous between˝ Φ –and˝ Γ ≡ ∀V∈Γ. (ProdFunction(f,f)-V)∈Φ
```

Any uniformly continuous function is continuous when considering the topologies on the uniformities.

```
lemma uniformly˙cont˙is˙cont:
  assumes f:X-¿Y Φ –is a uniformity on˝X  Γ –is a uniformity on˝Y
    f –is uniformly continuous between˝ Φ –and˝ Γ
  shows IsContinuous(UniformTopology(Φ,X),UniformTopology(Γ,Y),f)
proof -
  {  fix U assume op: U ∈ UniformTopology(Γ,Y)
    have f-(U) ∈ UniformTopology(Φ,X)
    proof -
      from assms(1) have f-(U) ⊆ X using func1˙1˙L3 by simp
      moreover
      { fix x xa assume as:⟨x,xa⟩ ∈ f xa ∈ U
        with assms(1) have x:x∈X unfolding Pi˙def by auto
          from as(2) op have U:U ∈ –⟨t,–V–t˝.V∈Γ˝⟩.t∈Y˝(xa) unfolding Uniform-
Topology˙def by auto
        from as(1) assms(1) have xa:xa ∈ Y unfolding Pi˙def by auto
          have –⟨t,–V–t˝.V∈Γ˝⟩.t∈Y˝ ∈ Pi(Y,%t. —V–t˝.V∈Γ˝˝) unfolding Pi˙def
function˙def
          by auto
        with U xa have U ∈–V–xa˝.V∈Γ˝ using apply˙equality by auto
        then obtain V where V:U = V–xa˝ V∈Γ by auto
         with assms have ent:(ProdFunction(f,f)-(V))∈Φ using IsUniformlyCont˙def
by simp
```

have ∀t. t ∈ (ProdFunction(f,f)-V)–x″ ¡-¿ ⟨x,t⟩ ∈ ProdFunction(f,f)-(V)
using image˙def by auto
with assms(1) x have ∀t. t: (ProdFunction(f,f)-V)–x″ ⟷ (t∈X ∧ ⟨fx,ft⟩ ∈ V)
using prodFunVimage by auto
with assms(1) as(1) have ∀t. t ∈ (ProdFunction(f,f)-V)–x″ ⟷ (t∈X ∧ ⟨xa,ft⟩: V)
using apply˙equality by auto
with V(1) have ∀t. t ∈ (ProdFunction(f,f)-V)–x″ ⟷ (t∈X ∧ f(t) ∈ U) by auto
with assms(1) U have ∀t. t ∈ (ProdFunction(f,f)-V)–x″ ⟷ t ∈ f-U
using func1˙1˙L15 by simp
hence f-U = (ProdFunction(f,f)-V)–x″ by blast
with ent have f-(U) ∈ –V–x″ . V ∈ Φ″ by auto
moreover
have –⟨t,–V–t″.V∈Φ″⟩.t∈X″ ∈ Pi(X,%t. —V–t″.V∈Φ″″) unfolding Pi˙def function˙def
by auto
ultimately have f-(U) ∈ –⟨t, –V –t″ . V ∈ Φ″⟩ . t ∈ X″(x) using x apply˙equality

by auto
}
ultimately show f-(U) ∈ UniformTopology(Φ,X) unfolding UniformTopology˙def by blast
qed
} then show thesis unfolding IsContinuous˙def by simp
qed

end

72 Topological groups - introduction

theory TopologicalGroup˙ZF imports Topology˙ZF˙3 Group˙ZF˙1 Semigroup˙ZF

begin

This theory is about the first subject of algebraic topology: topological groups.

72.1 Topological group: definition and notation

Topological group is a group that is a topological space at the same time. This means that a topological group is a triple of sets, say (G, f, T) such that T is a topology on G, f is a group operation on G and both f and the operation of taking inverse in G are continuous. Since IsarMathLib defines topology without using the carrier, (see Topology˙ZF), in our setup we just use $\bigcup T$ instead of G and say that the pair of sets $(\bigcup T, f)$ is a group. This way our definition of being a topological group is a statement about two

sets: the topology T and the group operation f on $G = \bigcup T$. Since the domain of the group operation is $G \times G$, the pair of topologies in which f is supposed to be continuous is T and the product topology on $G \times G$ (which we will call τ below).

This way we arrive at the following definition of a predicate that states that pair of sets is a topological group.

definition
IsAtopologicalGroup(T,f) ≡ (T –is a topology″) ∧ IsAgroup(⋃T,f) ∧
IsContinuous(ProductTopology(T,T),T,f) ∧
IsContinuous(T,T,GroupInv(⋃T,f))

We will inherit notation from the topology0 locale. That locale assumes that T is a topology. For convenience we will denote $G = \bigcup T$ and τ to be the product topology on $G \times G$. To that we add some notation specific to groups. We will use additive notation for the group operation, even though we don't assume that the group is abelian. The notation $g + A$ will mean the left translation of the set A by element g, i.e. $g + A = \{g + a | a \in A\}$. The group operation G induces a natural operation on the subsets of G defined as $\langle A, B\rangle \mapsto \{x + y | x \in A, y \in B\}$. Such operation has been considered in func˙ZF and called f "lifted to subsets of" G. We will denote the value of such operation on sets A, B as $A + B$. The set of neigboorhoods of zero (denoted $\mathcal{N}_0$) is the collection of (not necessarily open) sets whose interior contains the neutral element of the group.

locale topgroup = topology0 +

fixes G
defines G˙def [simp]: G ≡ ⋃T

fixes prodtop (τ)
defines prodtop˙def [simp]: τ ≡ ProductTopology(T,T)

fixes f

assumes Ggroup: IsAgroup(G,f)

assumes fcon: IsContinuous(τ,T,f)

assumes inv˙cont: IsContinuous(T,T,GroupInv(G,f))

fixes grop (infixl + 90)
defines grop˙def [simp]: x+y ≡ f⟨x,y⟩

fixes grinv (- ˙ 89)
defines grinv˙def [simp]: (-x) ≡ GroupInv(G,f)(x)

fixes grsub (infixl - 90)

defines grsub˙def [simp]: x-y ≡ x+(-y)

fixes setinv (- ˙ 72)
defines setninv˙def [simp]: -A ≡ GroupInv(G,f)(A)

fixes ltrans (infix + 73)
defines ltrans˙def [simp]: x + A ≡ LeftTranslation(G,f,x)(A)

fixes rtrans (infix + 73)
defines rtrans˙def [simp]: A + x ≡ RightTranslation(G,f,x)(A)

fixes setadd (infixl + 71)
defines setadd˙def [simp]: A+B ≡ (f –lifted to subsets of˝ G)⟨A,B⟩

fixes gzero (**0**)
defines gzero˙def [simp]: **0** ≡ TheNeutralElement(G,f)

fixes zerohoods ($\mathcal{N}_0$)
defines zerohoods˙def [simp]: $\mathcal{N}_0$ ≡ –A ∈ Pow(G). **0** ∈ int(A)˝

fixes listsum (∑ ˙ 70)
defines listsum˙def[simp]: ∑k ≡ Fold1(f,k)

The first lemma states that we indeeed talk about topological group in the context of topgroup locale.

lemma (in topgroup) topGroup: shows IsAtopologicalGroup(T,f)
using topSpaceAssum Ggroup fcon inv˙cont IsAtopologicalGroup˙def
by simp

If a pair of sets (T, f) forms a topological group, then all theorems proven in the topgroup context are valid as applied to (T, f).

lemma topGroupLocale: assumes IsAtopologicalGroup(T,f)
shows topgroup(T,f)
using assms IsAtopologicalGroup˙def topgroup˙def
topgroup˙axioms.intro topology0˙def by simp

We can use the group0 locale in the context of topgroup.

lemma (in topgroup) group0˙valid˙in˙tgroup: shows group0(G,f)
using Ggroup group0˙def by simp

We can use the group0 locale in the context of topgroup.

sublocale topgroup ¡ group0 G f gzero grop grinv
unfolding group0˙def gzero˙def grop˙def grinv˙def using Ggroup by auto

We can use semigr0 locale in the context of topgroup.

lemma (in topgroup) semigr0˙valid˙in˙tgroup: shows semigr0(G,f)
using Ggroup IsAgroup˙def IsAmonoid˙def semigr0˙def by simp

We can use the prod˙top˙spaces0 locale in the context of topgroup.

lemma (in topgroup) prod˙top˙spaces0˙valid: shows prod˙top˙spaces0(T,T,T)
  using topSpaceAssum prod˙top˙spaces0˙def by simp

Negative of a group element is in group.

lemma (in topgroup) neg˙in˙tgroup: assumes g∈G shows (-g) ∈ G
  using assms inverse˙in˙group by simp

Sum of two group elements is in the group.

lemma (in topgroup) group˙op˙closed˙add: assumes $x_1 \in G$ $x_2 \in G$
  shows $x_1+x_2 \in G$
  using assms group˙op˙closed by simp

Zero is in the group.

lemma (in topgroup) zero˙in˙tgroup: shows **0**∈G
  using group0˙2˙L2 by simp

Another lemma about canceling with two group elements written in additive notation

lemma (in topgroup) inv˙cancel˙two˙add:
  assumes $x_1 \in G$ $x_2 \in G$
  shows
    $x_1+(-x_2)+x_2 = x_1$
    $x_1+x_2+(-x_2) = x_1$
    $(-x_1)+(x_1+x_2) = x_2$
    $x_1+((-x_1)+x_2) = x_2$
  using assms inv˙cancel˙two by auto

Useful identities proven in the Group˙ZF theory, rewritten here in additive notation. Note since the group operation notation is left associative we don't really need the first set of parentheses in some cases.

lemma (in topgroup) cancel˙middle˙add: assumes $x_1 \in G$ $x_2 \in G$ $x_3 \in G$
  shows
    $(x_1+(-x_2))+(x_2+(-x_3)) = x_1+ (-x_3)$
    $((-x_1)+x_2)+((-x_2)+x_3) = (-x_1)+ x_3$
    $(- (x_1+x_2)) + (x_1+x_3) = (-x_2)+x_3$
    $(x_1+x_2) + (-(x_3+x_2)) =x_1+ (-x_3)$
    $(-x_1) + (x_1+x_2+x_3) + (-x_3) = x_2$
proof -
  from assms have $x_1+ (-x_3) = (x_1+(-x_2))+(x_2+(-x_3))$
    using group0˙2˙L14A(1) by blast
  thus $(x_1+(-x_2))+(x_2+(-x_3)) = x_1+ (-x_3)$ by simp
  from assms have $(-x_1)+ x_3 = ((-x_1)+x_2)+((-x_2)+x_3)$
    using group0˙2˙L14A(2) by blast
  thus $((-x_1)+x_2)+((-x_2)+x_3) = (-x_1)+ x_3$ by simp
  from assms show $(- (x_1+x_2)) + (x_1+x_3) = (-x_2)+x_3$
    using cancel˙middle(1) by simp

from assms show $(x_1+x_2) + (-(x_3+x_2)) = x_1 + (-x_3)$
using cancel˙middle(2) by simp
from assms show $(-x_1) + (x_1+x_2+x_3) + (-x_3) = x_2$
using cancel˙middle(3) by simp
qed

We can cancel an element on the right from both sides of an equation.

lemma (in topgroup) cancel˙right˙add:
assumes $x_1 \in G$ $x_2 \in G$ $x_3 \in G$ $x_1+x_2 = x_3+x_2$
shows $x_1 = x_3$
using assms cancel˙right by simp

We can cancel an element on the left from both sides of an equation.

lemma (in topgroup) cancel˙left˙add:
assumes $x_1 \in G$ $x_2 \in G$ $x_3 \in G$ $x_1+x_2 = x_1+x_3$
shows $x_2 = x_3$
using assms cancel˙left by simp

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

lemma (in topgroup) put˙on˙the˙other˙side:
assumes $x_1 \in G$ $x_2 \in G$ $x_3 = x_1+x_2$
shows $x_3+(-x_2) = x_1$ and $(-x_1)+x_3 = x_2$
using assms group0˙2˙L18 by auto

A simple equation from lemma simple˙equation0 in Group˙ZF in additive notation

lemma (in topgroup) simple˙equation0˙add:
assumes $x_1 \in G$ $x_2 \in G$ $x_3 \in G$ $x_1+(-x_2) = (-x_3)$
shows $x_3 = x_2 + (-x_1)$
using assms simple˙equation0 by blast

A simple equation from lemma simple˙equation1 in Group˙ZF in additive notation

lemma (in topgroup) simple˙equation1˙add:
assumes $x_1 \in G$ $x_2 \in G$ $x_3 \in G$ $(-x_1)+x_2 = (-x_3)$
shows $x_3 = (-x_2) + x_1$
using assms simple˙equation1 by blast

The set comprehension form of negative of a set. The proof uses the ginv˙image lemma from Group˙ZF theory which states the same thing in multiplicative notation.

lemma (in topgroup) ginv˙image˙add: assumes $V \subseteq G$
shows $(-V) \subseteq G$ and $(-V) = -x.\ x \in V''$
using assms ginv˙image by auto

The additive notation version of ginv˙image˙el lemma from Group˙ZF theory

lemma (in topgroup) ginv˙image˙el˙add: assumes $V \subseteq G$ $x \in (-V)$

shows (-x) ∈ V
using assms ginv˙image˙el by simp

Of course the product topology is a topology (on $G \times G$).

lemma (in topgroup) prod˙top˙on˙G:
shows τ –is a topology″ and $\bigcup \tau$ = G×G
using topSpaceAssum Top˙1˙4˙T1 by auto

Let's recall that f is a binary operation on G in this context.

lemma (in topgroup) topgroup˙f˙binop: shows f : G×G → G
using Ggroup group0˙def group0.group˙oper˙assocA by simp

A subgroup of a topological group is a topological group with relative topology and restricted operation. Relative topology is the same as T –restricted to″ H which is defined to be $\{V \cap H : V \in T\}$ in ZF1 theory.

lemma (in topgroup) top˙subgroup: assumes A1: IsAsubgroup(H,f)
shows IsAtopologicalGroup(T –restricted to″ H,restrict(f,H×H))
proof -
let τ_0 = T –restricted to″ H
let f_H = restrict(f,H×H)
have $\bigcup \tau_0$ = G ∩ H using union˙restrict by simp
also from A1 have ... = H
using group0˙3˙L2 by blast
finally have $\bigcup \tau_0$ = H by simp
have τ_0 –is a topology″ using Top˙1˙L4 by simp
moreover from A1 ‹$\bigcup \tau_0$ = H› have IsAgroup($\bigcup \tau_0$,f_H)
using IsAsubgroup˙def by simp
moreover have IsContinuous(ProductTopology(τ_0,τ_0),τ_0,f_H)
proof -
have two˙top˙spaces0(τ, T,f)
using topSpaceAssum prod˙top˙on˙G topgroup˙f˙binop prod˙top˙on˙G two˙top˙spaces0˙def by simp
moreover
from A1 have H ⊆ G using group0˙3˙L2 by simp
then have H×H ⊆ $\bigcup \tau$ using prod˙top˙on˙G by auto
moreover have IsContinuous(τ,T,f) using fcon by simp
ultimately have
IsContinuous(τ –restricted to″ H×H, T –restricted to″ f_H(H×H),f_H) using two˙top˙spaces0.restr˙restr˙image˙cont
by simp
moreover have
ProductTopology(τ_0,τ_0) = τ –restricted to″ H×H using topSpaceAssum prod˙top˙restr˙comm
by simp
moreover from A1 have f_H(H×H) = H using image˙subgr˙op
by simp
ultimately show thesis by simp
qed

```
  moreover have IsContinuous(τ0,τ0,GroupInv(⋃τ0,fH))
  proof -
    let g = restrict(GroupInv(G,f),H)
    have GroupInv(G,f) : G → G
      using Ggroup group0˙2˙T2 by simp
    then have two˙top˙spaces0(T,T,GroupInv(G,f))
      using topSpaceAssum two˙top˙spaces0˙def by simp
    moreover from A1 have H ⊆ ⋃T using group0˙3˙L2 by simp
    ultimately have
      IsContinuous(τ0,T –restricted to″ g(H),g)
      using inv˙cont two˙top˙spaces0.restr˙restr˙image˙cont
      by simp
    moreover from A1 have g(H) = H using restr˙inv˙onto by simp
    moreover
    from A1 have GroupInv(H,fH) = g using group0˙3˙T1 by simp
    with ‹⋃τ0 = H› have g = GroupInv(⋃τ0,fH) by simp
    ultimately show thesis by simp
  qed
  ultimately show thesis unfolding IsAtopologicalGroup˙def by simp
qed
```

72.2 Interval arithmetic, translations and inverse of set

In this section we list some properties of operations of translating a set and reflecting it around the neutral element of the group. Many of the results are proven in other theories, here we just collect them and rewrite in notation specific to the topgroup context.

Different ways of looking at adding sets.

```
lemma (in topgroup) interval˙add: assumes A⊆G B⊆G shows
  A+B ⊆ G
  A+B = f(A×B)
  A+B = (⋃x∈A. x+B)
  A+B = –x+y. ⟨x,y⟩ ∈ A×B″
proof -
  from assms show A+B ⊆ G and A+B = f(A×B) and A+B = –x+y. ⟨x,y⟩ ∈ A×B″
    using topgroup˙f˙binop lift˙subsets˙explained by auto
  from assms show A+B = (⋃x∈A. x+B) using image˙ltrans˙union by simp
qed
```

If the neutral element is in a set, then it is in the sum of the sets.

```
lemma (in topgroup) interval˙add˙zero: assumes A⊆G 0∈A
  shows 0 ∈ A+A
proof -
  from assms have 0+0 ∈ A+A using interval˙add(4) by auto
  then show 0 ∈ A+A using group0˙2˙L2 by auto
qed
```

Some lemmas from Group˙ZF˙1 about images of set by translations written in additive notation

lemma (in topgroup) lrtrans˙image: assumes V⊆G x∈G
shows
x+V = −x+v. v∈V″
V+x = −v+x. v∈V″
using assms ltrans˙image rtrans˙image by auto

Right and left translations of a set are subsets of the group. This is of course typically applied to the subsets of the group, but formally we don't need to assume that.

lemma (in topgroup) lrtrans˙in˙group˙add: assumes x∈G
shows x+V ⊆ G and V+x ⊆G
using assms lrtrans˙in˙group by auto

A corollary from interval˙add

corollary (in topgroup) elements˙in˙set˙sum: assumes A⊆G B⊆G
t ∈ A+B shows ∃s∈A. ∃q∈B. t=s+q
using assms interval˙add(4) by auto

A corollary from lrtrans˙image

corollary (in topgroup) elements˙in˙ltrans:
assumes B⊆G g∈G t ∈ g+B
shows ∃q∈B. t=g+q
using assms lrtrans˙image(1) by simp

Another corollary of lrtrans˙image

corollary (in topgroup) elements˙in˙rtrans:
assumes B⊆G g∈G t ∈ B+g shows ∃q∈B. t=q+g
using assms lrtrans˙image(2) by simp

Another corollary from interval˙add

corollary (in topgroup) elements˙in˙set˙sum˙inv:
assumes A⊆G B⊆G t=s+q s∈A q∈B
shows t ∈ A+B
using assms interval˙add by auto

Another corollary of lrtrans˙image

corollary (in topgroup) elements˙in˙ltrans˙inv: assumes B⊆G g∈G q∈B t=g+q
shows t ∈ g+B
using assms lrtrans˙image(1) by auto

Another corollary of rtrans˙image˙add

lemma (in topgroup) elements˙in˙rtrans˙inv:
assumes B⊆G g∈G q∈B t=q+g
shows t ∈ B+g
using assms lrtrans˙image(2) by auto

Right and left translations are continuous.

```
lemma (in topgroup) trans˙cont: assumes g∈G shows
  IsContinuous(T,T,RightTranslation(G,f,g)) and
  IsContinuous(T,T,LeftTranslation(G,f,g))
using assms trans˙eq˙section topgroup˙f˙binop fcon prod˙top˙spaces0˙valid
  prod˙top˙spaces0.fix˙1st˙var˙cont prod˙top˙spaces0.fix˙2nd˙var˙cont
  by auto
```

Left and right translations of an open set are open.

```
lemma (in topgroup) open˙tr˙open: assumes g∈G and V∈T
  shows g+V ∈ T and  V+g ∈ T
 using assms neg˙in˙tgroup trans˙cont IsContinuous˙def trans˙image˙vimage by auto
```

Right and left translations are homeomorphisms.

```
lemma (in topgroup) tr˙homeo: assumes g∈G shows
  IsAhomeomorphism(T,T,RightTranslation(G,f,g)) and
  IsAhomeomorphism(T,T,LeftTranslation(G,f,g))
  using assms trans˙bij trans˙cont open˙tr˙open bij˙cont˙open˙homeo
  by auto
```

Left translations preserve interior.

```
lemma (in topgroup) ltrans˙interior: assumes A1: g∈G and A2: A⊆G
  shows g + int(A) = int(g+A)
proof -
  from assms have A ⊆ ⋃T and IsAhomeomorphism(T,T,LeftTranslation(G,f,g))
using tr˙homeo
    by auto
  then show thesis using int˙top˙invariant by simp
qed
```

Right translations preserve interior.

```
lemma (in topgroup) rtrans˙interior: assumes A1: g∈G and A2: A⊆G
  shows int(A) + g = int(A+g)
proof -
  from assms have A ⊆ ⋃T and IsAhomeomorphism(T,T,RightTranslation(G,f,g))
using tr˙homeo
    by auto
  then show thesis using int˙top˙invariant by simp
qed
```

Translating by an inverse and then by an element cancels out.

```
lemma (in topgroup) trans˙inverse˙elem: assumes g∈G and A⊆G
  shows g+((-g)+A) = A
 using assms neg˙in˙tgroup trans˙comp˙image group0˙2˙L6 trans˙neutral image˙id˙same

  by simp
```

Inverse of an open set is open.

```
lemma (in topgroup) open˙inv˙open: assumes V∈T shows (-V) ∈ T
  using assms inv˙image˙vimage inv˙cont IsContinuous˙def by simp
```

Inverse is a homeomorphism.

```
lemma (in topgroup) inv˙homeo: shows IsAhomeomorphism(T,T,GroupInv(G,f))
  using group˙inv˙bij inv˙cont open˙inv˙open bij˙cont˙open˙homeo by simp
```

Taking negative preserves interior.

```
lemma (in topgroup) int˙inv˙inv˙int: assumes A ⊆ G
  shows int(-A) = -(int(A))
  using assms inv˙homeo int˙top˙invariant by simp
```

72.3 Neighborhoods of zero

Zero neighborhoods are (not necessarily open) sets whose interior contains the neutral element of the group. In the topgroup locale the collection of neighboorhoods of zero is denoted $\mathcal{N}_0$.

The whole space is a neighborhood of zero.

```
lemma (in topgroup) zneigh˙not˙empty: shows G ∈ 𝒩0
  using topSpaceAssum IsATopology˙def Top˙2˙L3 zero˙in˙tgroup
  by simp
```

Any element that belongs to a subset of the group belongs to that subset with the interior of a neighborhood of zero added.

```
lemma (in topgroup) elem˙in˙int˙sad: assumes A⊆G g∈A H ∈ 𝒩0
  shows g ∈ A+int(H)
proof -
  from assms(3) have 0 ∈ int(H) and int(H) ⊆ G using Top˙2˙L2 by auto
  with assms(1,2) have g+0 ∈ A+int(H) using elements˙in˙set˙sum˙inv
    by simp
  with assms(1,2) show thesis using group0˙2˙L2 by auto
qed
```

Any element belongs to the interior of any neighboorhood of zero left translated by that element.

```
lemma (in topgroup) elem˙in˙int˙ltrans:
  assumes g∈G and H ∈ 𝒩0
  shows g ∈ int(g+H) and g ∈ int(g+H) + int(H)
proof -
  from assms(2) have 0 ∈ int(H) and int(H) ⊆ G using Top˙2˙L2 by auto
  with assms(1) have g ∈ g + int(H) using neut˙trans˙elem by simp
  with assms show g ∈ int(g+H) using ltrans˙interior by simp
  from assms(1) have int(g+H) ⊆ G using lrtrans˙in˙group˙add(1) Top˙2˙L1
    by blast
  with ‹g ∈ int(g+H)› assms(2) show g ∈ int(g+H) + int(H)
    using elem˙in˙int˙sad by simp
```

qed

Any element belongs to the interior of any neighboorhood of zero right translated by that element.

```
lemma (in topgroup) elem˙in˙int˙rtrans:
  assumes A1: g∈G and A2: H ∈ 𝒩0
  shows g ∈ int(H+g) and g ∈ int(H+g) + int(H)
proof -
  from A2 have 𝟎 ∈ int(H) and int(H) ⊆ G using Top˙2˙L2 by auto
  with A1 have g ∈ int(H) + g using neut˙trans˙elem by simp
  with assms show g ∈ int(H+g) using rtrans˙interior by simp
  from assms(1) have int(H+g) ⊆ G using lrtrans˙in˙group˙add(2) Top˙2˙L1
    by blast
  with ‹g ∈ int(H+g)› assms(2) show g ∈ int(H+g) + int(H)
    using elem˙in˙int˙sad by simp
qed
```

Negative of a neighborhood of zero is a neighborhood of zero.

```
lemma (in topgroup) neg˙neigh˙neigh: assumes H ∈ 𝒩0
  shows (-H) ∈ 𝒩0
proof -
  from assms have int(H) ⊆ G and 𝟎 ∈ int(H) using Top˙2˙L1 by auto
  with assms have 𝟎 ∈ int(-H) using neut˙inv˙neut int˙inv˙inv˙int by simp
  moreover
  have GroupInv(G,f):G→G using Ggroup group0˙2˙T2 by simp
  then have (-H) ⊆ G using func1˙1˙L6 by simp
  ultimately show thesis by simp
qed
```

Left translating an open set by a negative of a point that belongs to it makes it a neighboorhood of zero.

```
lemma (in topgroup) open˙trans˙neigh: assumes A1: U∈T and g∈U
  shows (-g)+U ∈ 𝒩0
proof -
  let H = (-g)+U
  from assms have g∈G by auto
  then have (-g) ∈ G using neg˙in˙tgroup by simp
  with A1 have H∈T using open˙tr˙open by simp
  hence H ⊆ G by auto
  moreover have 𝟎 ∈ int(H)
  proof -
    from assms have U⊆G and g∈U by auto
    with ‹H∈T› show 𝟎 ∈ int(H) using elem˙trans˙neut Top˙2˙L3 by auto
  qed
  ultimately show thesis by simp
qed
```

Right translating an open set by a negative of a point that belongs to it makes it a neighboorhood of zero.

```
lemma (in topgroup) open˙trans˙neigh˙2: assumes A1: U∈T and g∈U
  shows U+(-g) ∈ 𝒩0
proof -
  let H = U+(-g)
  from assms have g∈G by auto
  then have (-g) ∈ G using neg˙in˙tgroup by simp
  with A1 have H∈T using open˙tr˙open by simp
  hence H ⊆ G by auto
  moreover have 0 ∈ int(H)
  proof -
    from assms have U⊆G and g∈U by auto
    with ‹H∈T› show 0 ∈ int(H) using elem˙trans˙neut Top˙2˙L3  by auto
  qed
  ultimately show thesis by simp
qed
```

Right and left translating an neighboorhood of zero by a point and its negative makes it back a neighboorhood of zero.

```
lemma (in topgroup) lrtrans˙neigh: assumes W∈𝒩0 and x∈G
  shows x+(W+(-x)) ∈ 𝒩0 and (x+W)+(-x) ∈ 𝒩0
proof -
  from assms(2) have x+(W+(-x)) ⊆ G using lrtrans˙in˙group˙add(1) by simp
  moreover have 0 ∈ int(x+(W+(-x)))
  proof -
    from assms(2) have int(W+(-x)) ⊆ G
      using neg˙in˙tgroup lrtrans˙in˙group˙add(2) Top˙2˙L1 by blast
    with assms(2) have (x+int((W+(-x)))) = –x+y. y∈int(W+(-x))″
      using lrtrans˙image(1) by simp
    moreover from assms have (-x) ∈ int(W+(-x))
      using neg˙in˙tgroup elem˙in˙int˙rtrans(1) by simp
    ultimately have x+(-x) ∈ x+int(W+(-x)) by auto
    with assms show thesis using group0˙2˙L6 neg˙in˙tgroup lrtrans˙in˙group˙add(2)
ltrans˙interior
      by simp
  qed
  ultimately show x+(W+(-x)) ∈ 𝒩0 by simp
  from assms(2) have (x+W)+(-x) ⊆ G using lrtrans˙in˙group˙add(2) neg˙in˙tgroup

    by simp
  moreover have 0 ∈ int((x+W)+(-x))
  proof -
    from assms(2) have int((x+W)) ⊆ G using lrtrans˙in˙group˙add(1) Top˙2˙L1 by
blast
    with assms(2) have int(x+W) + (-x) = –y+(-x).y∈int(x+W)″
      using neg˙in˙tgroup  lrtrans˙image(2) by simp
    moreover from assms have x ∈ int(x+W) using elem˙in˙int˙ltrans(1) by simp
    ultimately have x+(-x) ∈ int(x+W) + (-x) by auto
    with assms(2) have 0 ∈ int(x+W) + (-x) using group0˙2˙L6 by simp
    with assms show thesis using group0˙2˙L6 neg˙in˙tgroup lrtrans˙in˙group˙add(1)
```

```
rtrans˙interior
      by auto
  qed
  ultimately show (x+W)+(-x) ∈ 𝒩0 by simp
qed
```

If A is a subset of B translated by $-x$ then its translation by x is a subset of B.

```
lemma (in topgroup) trans˙subset:
  assumes A ⊆ ((-x)+B)x∈G B⊆G
  shows x+A ⊆ B
proof-
  from assms(1) have x+A ⊆ (x+ ((-x)+B)) by auto
  with assms(2,3) show x+A ⊆ B
    using neg˙in˙tgroup trans˙comp˙image group0˙2˙L6 trans˙neutral image˙id˙same
by simp
qed
```

Every neighborhood of zero has a symmetric subset that is a neighborhood of zero.

```
theorem (in topgroup) exists˙sym˙zerohood:
  assumes U∈𝒩0
  shows ∃V∈𝒩0. (V⊆U ∧ (-V)=V)
proof
  let V = U∩(-U)
  have U⊆G using assms unfolding zerohoods˙def by auto
  then have V⊆G by auto
  have invg: GroupInv(G, f) ∈ G → G using group0˙2˙T2 Ggroup by auto
  have invb:GroupInv(G, f) ∈bij(G,G) using group˙inv˙bij(2) by auto
  have (-V)=GroupInv(G,f)-V unfolding setninv˙def using inv˙image˙vimage
    by auto
 also have ...=(GroupInv(G,f)-U)∩(GroupInv(G,f)-(-U)) using invim˙inter˙inter˙invim
invg
    by auto
  also have ...=(-U)∩(GroupInv(G,f)-(GroupInv(G,f)U))
    unfolding setninv˙def using inv˙image˙vimage by auto
  also from ‹U⊆G› have ...=(-U)∩U using inj˙vimage˙image invb unfolding bij˙def
    by auto
  finally have (-V)=V by auto
  then show V ⊆ U ∧ (- V) = V by auto
  from assms have (-U)∈𝒩0 using neg˙neigh˙neigh by auto
  with assms have 0 ∈ int(U)∩int(-U) unfolding zerohoods˙def by auto
  moreover have int(U)∩int(-U) = int(V) using int˙inter˙int by simp
  ultimately have 0 ∈ int(V) by (rule set˙mem˙eq)
  with ‹V⊆G› show V∈𝒩0 using zerohoods˙def by auto
qed
```

We can say even more than in exists˙sym˙zerohood: every neighborhood of zero U has a symmetric subset that is a neighborhood of zero and its set

double is contained in U.

```
theorem (in topgroup) exists˙procls˙zerohood:
  assumes U∈𝒩0
  shows ∃V∈𝒩0. (V⊆U∧ (V+V)⊆U ∧ (-V)=V)
proof-
  have int(U)∈T using Top˙2˙L2 by auto
  then have f-(int(U))∈τ using fcon IsContinuous˙def by auto
  moreover have fne:f ⟨0, 0⟩ = 0 using group0˙2˙L2 by auto
  moreover
  have 0∈int(U) using assms unfolding zerohoods˙def by auto
  then have f - −0″⊆f-(int(U)) using func1˙1˙L8 vimage˙def by auto
  then have GroupInv(G,f)⊆f-(int(U)) using group0˙2˙T3 by auto
  then have ⟨0,0⟩∈f-(int(U)) using fne zero˙in˙tgroup unfolding GroupInv˙def
    by auto
  ultimately obtain W V where
   wop:W∈T and vop:V∈T and cartsub:W×V⊆f-(int(U)) and zerhood:⟨0,0⟩∈W×V

    using prod˙top˙point˙neighb topSpaceAssum
    unfolding prodtop˙def by force
  then have 0∈W and 0∈V by auto
  then have 0∈W∩V by auto
  have sub:W∩V⊆G using wop vop G˙def by auto
  have assoc:f∈G×G→G using group˙oper˙assocA by auto
  {
    fix t s assume t∈W∩V and s∈W∩V
    then have t∈W and s∈V by auto
    then have ⟨t,s⟩∈W×V by auto
    then have ⟨t,s⟩∈f-(int(U)) using cartsub by auto
    then have f⟨t,s⟩∈int(U) using func1˙1˙L15 assoc by auto
  } hence −f⟨t,s⟩. ⟨t,s⟩∈(W∩V)×(W∩V)″⊆int(U) by auto
  then have (W∩V)+(W∩V)⊆int(U)
    unfolding setadd˙def using lift˙subsets˙explained(4) assoc sub
    by auto
  then have (W∩V)+(W∩V)⊆U using Top˙2˙L1 by auto
  from topSpaceAssum have W∩V∈T using vop wop unfolding IsATopology˙def by
auto
  then have int(W∩V)=W∩V using Top˙2˙L3 by auto
  with sub ‹0∈W∩V› have W∩V∈𝒩0 unfolding zerohoods˙def by auto
  then obtain Q where Q∈𝒩0 and Q⊆W∩V and (-Q)=Q using exists˙sym˙zerohood
by blast
  then have Q×Q⊆(W∩V)×(W∩V) by auto
  moreover from ‹Q⊆W∩V› have W∩V⊆G and Q⊆G using vop wop unfolding G˙def
by auto
  ultimately have Q+Q⊆(W∩V)+(W∩V) using interval˙add(2) func1˙1˙L8 by auto
  with ‹(W∩V)+(W∩V)⊆U› have Q+Q⊆U by auto
  from ‹Q∈𝒩0› have 0∈Q unfolding zerohoods˙def using Top˙2˙L1 by auto
  with ‹Q+Q⊆U› ‹Q⊆G› have 0+Q⊆U using interval˙add(3) by auto
  with ‹Q⊆G› have Q⊆U unfolding ltrans˙def gzero˙def using trans˙neutral(2) im-
age˙id˙same
```

```
    by auto
  with ‹Q∈𝒩0› ‹Q+Q⊆U› ‹(-Q)=Q› show thesis by auto
qed
```

72.4 Closure in topological groups

This section is devoted to a characterization of closure in topological groups.

Closure of a set is contained in the sum of the set and any neighboorhood of zero.

```
lemma (in topgroup) cl˙contains˙zneigh:
  assumes A1: A⊆G and A2: H ∈ 𝒩0
  shows cl(A) ⊆ A+H
proof
  fix x assume x ∈ cl(A)
  from A1 have cl(A) ⊆ G using Top˙3˙L11 by simp
  with ‹x ∈ cl(A)› have x∈G by auto
  have int(H) ⊆ G using Top˙2˙L2 by auto
  let V = int(x + (-H))
  have V = x + (-int(H))
  proof -
    from A2 ‹x∈G› have V = x + int(-H)
      using neg˙neigh˙neigh ltrans˙interior by simp
    with A2 show thesis  using int˙inv˙inv˙int by simp
  qed
  have A∩V ≠ 0
  proof -
    from A2 ‹x∈G› ‹x ∈ cl(A)› have V∈T and x ∈ cl(A) ∩ V
      using neg˙neigh˙neigh elem˙in˙int˙ltrans(1) Top˙2˙L2 by auto
    with A1 show A∩V ≠ 0 using cl˙inter˙neigh by simp
  qed
  then obtain y where y∈A and y∈V by auto
  with ‹V = x + (-int(H))› ‹int(H) ⊆ G› ‹x∈G› have x ∈ y+int(H)
    using ltrans˙inv˙in by simp
  with ‹y∈A› have x ∈ (⋃y∈A. y+H) using Top˙2˙L1 func1˙1˙L8 by auto
  with assms show x ∈ A+H using interval˙add(3) by simp
qed
```

The next theorem provides a characterization of closure in topological groups in terms of neighborhoods of zero.

```
theorem (in topgroup) cl˙topgroup:
  assumes A⊆G shows cl(A) = (⋂H∈𝒩0. A+H)
proof
  from assms show cl(A) ⊆ (⋂H∈𝒩0. A+H)
    using zneigh˙not˙empty cl˙contains˙zneigh by auto
next
  { fix x assume x ∈ (⋂H∈𝒩0. A+H)
    then have x ∈ A+G using zneigh˙not˙empty by auto
    with assms have x∈G using interval˙add by blast
```

```
    have ∀U∈T. x∈U ⟶ U∩A ≠ 0
    proof -
      { fix U assume U∈T and x∈U
        let H = -((-x)+U)
        from ‹U∈T› and ‹x∈U› have (-x)+U ⊆ G and H ∈ 𝒩0
          using open˙trans˙neigh neg˙neigh˙neigh by auto
        with ‹x ∈ (⋂H∈𝒩0. A+H)› have x ∈ A+H by auto
        with assms ‹H ∈ 𝒩0› obtain y where y∈A and x ∈ y+H
          using interval˙add(3) by auto
        have y∈U
        proof -
          from assms ‹y∈A› have y∈G by auto
          with ‹(-x)+U ⊆ G› and ‹x ∈ y+H› have y ∈ x+((-x)+U)
            using ltrans˙inv˙in by simp
          with ‹U∈T› ‹x∈G› show y∈U
                using neg˙in˙tgroup trans˙comp˙image group0˙2˙L6 trans˙neutral im-
age˙id˙same
              by auto
        qed
        with ‹y∈A› have U∩A ≠ 0 by auto
      } thus thesis by simp
    qed
    with assms ‹x∈G› have x ∈ cl(A) using inter˙neigh˙cl by simp
  } thus (⋂H∈𝒩0. A+H) ⊆ cl(A) by auto
qed
```

72.5 Sums of sequences of elements and subsets

In this section we consider properties of the function $G^n \to G, x = (x_0, x_1, ..., x_{n-1}) \mapsto \sum_{i=0}^{n-1} x_i$. We will model the cartesian product G^n by the space of sequences $n \to G$, where $n = \{0, 1, ..., n-1\}$ is a natural number. This space is equipped with a natural product topology defined in Topology˙ZF˙3.

Let's recall first that the sum of elements of a group is an element of the group.

```
lemma (in topgroup) sum˙list˙in˙group:
  assumes n ∈ nat and x: succ(n)→G
  shows (∑x) ∈ G
proof -
  from assms have semigr0(G,f) and n ∈ nat x: succ(n)→G
    using semigr0˙valid˙in˙tgroup by auto
  then have Fold1(f,x) ∈ G by (rule semigr0.prod˙type)
  thus (∑x) ∈ G by simp
qed
```

In this context x+y is the same as the value of the group operation on the elements x and y. Normally we shouldn't need to state this a s separate lemma.

lemma (in topgroup) grop˙def1: shows f⟨x,y⟩ = x+y by simp

Another theorem from Semigroup˙ZF theory that is useful to have in the additive notation.

```
lemma (in topgroup) shorter˙set˙add:
  assumes n ∈ nat and x: succ(succ(n))→G
  shows (∑x) = (∑Init(x)) + (x(succ(n)))
proof -
  from assms have semigr0(G,f) and n ∈ nat x: succ(succ(n))→G
    using semigr0˙valid˙in˙tgroup by auto
  then have Fold1(f,x) = f⟨Fold1(f,Init(x)),x(succ(n))⟩
    by (rule semigr0.shorter˙seq)
  thus thesis by simp
qed
```

Sum is a continuous function in the product topology.

```
theorem (in topgroup) sum˙continuous: assumes n ∈ nat
  shows IsContinuous(SeqProductTopology(succ(n),T),T,−⟨x,∑x⟩.x∈succ(n)→G″)
  proof -
    note ‹n ∈ nat›
   moreover have IsContinuous(SeqProductTopology(succ(0),T),T,−⟨x,∑x⟩.x∈succ(0)→G″)
    proof -
      have −⟨x,∑x⟩.x∈succ(0)→G″ = −⟨x,x(0)⟩. x∈1→G″
        using semigr0˙valid˙in˙tgroup semigr0.prod˙of˙1elem by simp
      moreover have
      IsAhomeomorphism(SeqProductTopology(1,T),T,−⟨x,x(0)⟩. x∈1→⋃T″) using
topSpaceAssum singleton˙prod˙top1
          by simp
      ultimately show thesis using IsAhomeomorphism˙def by simp
    qed
    moreover have ∀k∈nat.
      IsContinuous(SeqProductTopology(succ(k),T),T,−⟨x,∑x⟩.x∈succ(k)→G″)
      ⟶
    IsContinuous(SeqProductTopology(succ(succ(k)),T),T,−⟨x,∑x⟩.x∈succ(succ(k))→G″)
      proof -
        { fix k assume k ∈ nat
          let s = −⟨x,∑x⟩.x∈succ(k)→G″
          let g = −⟨p,⟨s(fst(p)),snd(p)⟩⟩. p ∈ (succ(k)→G)×G″
          let h = −⟨x,⟨Init(x),x(succ(k))⟩⟩. x ∈ succ(succ(k))→G″
          let φ = SeqProductTopology(succ(k),T)
          let ψ = SeqProductTopology(succ(succ(k)),T)
          assume IsContinuous(φ,T,s)
          from ‹k ∈ nat› have s: (succ(k)→G) → G
            using sum˙list˙in˙group ZF˙fun˙from˙total by simp
          have h: (succ(succ(k))→G)→(succ(k)→G)×G
          proof -
            { fix x assume x ∈ succ(succ(k))→G
              with ‹k ∈ nat› have Init(x) ∈ (succ(k)→G)
                using init˙props by simp
```

```
    with ‹k ∈ nat› ‹x : succ(succ(k))→G›
      have ⟨Init(x),x(succ(k))⟩ ∈ (succ(k)→G)×G using apply˙funtype
        by blast
  } then show thesis using ZF˙fun˙from˙total by simp
qed
moreover have g:((succ(k)→G)×G)→(G×G)
proof -
  { fix p assume p ∈ (succ(k)→G)×G
    hence fst(p): succ(k)→G and snd(p) ∈ G by auto
    with ‹s: (succ(k)→G) → G› have ⟨s(fst(p)),snd(p)⟩ ∈ G×G
      using apply˙funtype by blast
  } then show g:((succ(k)→G)×G)→(G×G) using ZF˙fun˙from˙total
    by simp
qed
moreover have f : G×G → G using topgroup˙f˙binop by simp
ultimately have f O g O h :(succ(succ(k))→G)→G using comp˙fun
  by blast
from ‹k ∈ nat› have IsContinuous(ψ,ProductTopology(φ,T),h)
  using topSpaceAssum finite˙top˙prod˙homeo IsAhomeomorphism˙def
  by simp
moreover have IsContinuous(ProductTopology(φ,T),τ,g)
proof -
  from topSpaceAssum have
     T –is a topology″ φ –is a topology″ ⋃φ = succ(k)→G
     using seq˙prod˙top˙is˙top by auto
  moreover from ‹⋃φ = succ(k)→G› ‹s: (succ(k)→G) → G›
    have s:⋃φ→⋃T by simp
  moreover note ‹IsContinuous(φ,T,s)›
  moreover from ‹⋃φ = succ(k)→G›
    have g = –⟨p,⟨s(fst(p)),snd(p)⟩⟩. p ∈ ⋃φ×⋃T″
    by simp
ultimately have IsContinuous(ProductTopology(φ,T),ProductTopology(T,T),g)
    using cart˙prod˙cont1 by blast
  thus thesis by simp
qed
moreover have IsContinuous(τ,T,f) using fcon by simp
moreover have –⟨x,∑x⟩.x∈succ(succ(k))→G″ = f O g O h
proof -
  let d = –⟨x,∑x⟩.x∈succ(succ(k))→G″
  from ‹k∈nat› have ∀x∈succ(succ(k))→G. (∑x) ∈ G
    using sum˙list˙in˙group by blast
  then have d:(succ(succ(k))→G)→G
    using sum˙list˙in˙group ZF˙fun˙from˙total by simp
  moreover note ‹f O g O h :(succ(succ(k))→G)→G›
  moreover have ∀x∈succ(succ(k))→G. d(x) = (f O g O h)(x)
  proof
    fix x assume x∈succ(succ(k))→G
    then have I: h(x) = ⟨Init(x),x(succ(k))⟩
      using ZF˙fun˙from˙tot˙val1 by simp
```

```
            moreover from ‹k∈nat› ‹x∈succ(succ(k))→G›
              have Init(x): succ(k)→G
              using init˙props by simp
            moreover from ‹k∈nat› ‹x:succ(succ(k))→G›
              have II: x(succ(k)) ∈ G
              using apply˙funtype by blast
            ultimately have h(x) ∈ (succ(k)→G)×G by simp
            then have g(h(x)) = ⟨s(fst(h(x))),snd(h(x))⟩
              using ZF˙fun˙from˙tot˙val1 by simp
            with I have g(h(x)) = ⟨s(Init(x)),x(succ(k))⟩
              by simp
            with ‹Init(x): succ(k)→G› have g(h(x)) = ⟨∑Init(x),x(succ(k))⟩
              using ZF˙fun˙from˙tot˙val1 by simp
            with ‹k ∈ nat› ‹x: succ(succ(k))→G›
              have f(g(h(x))) = (∑x)
              using shorter˙set˙add by simp
            with ‹x ∈ succ(succ(k))→G› have f(g(h(x))) = d(x)
              using ZF˙fun˙from˙tot˙val1 by simp
            moreover from
              ‹h: (succ(succ(k))→G)→(succ(k)→G)×G›
              ‹g:((succ(k)→G)×G)→(G×G)›
              ‹f:(G×G)→G› ‹x∈succ(succ(k))→G›
              have (f O g O h)(x) = f(g(h(x))) by (rule func1˙1˙L18)
            ultimately show d(x) = (f O g O h)(x) by simp
          qed
          ultimately show −⟨x,∑x⟩.x∈succ(succ(k))→G″ = f O g O h
            using func˙eq by simp
        qed
        moreover note ‹IsContinuous(τ,T,f)›
        ultimately have IsContinuous(ψ,T,−⟨x,∑x⟩.x∈succ(succ(k))→G″)
          using comp˙cont3 by simp
      } thus thesis by simp
    qed
  ultimately show thesis by (rule ind˙on˙nat)
 qed
end
```

73 Topological groups 1

theory TopologicalGroup˙ZF˙1 imports TopologicalGroup˙ZF Topology˙ZF˙properties˙2 begin

This theory deals with some topological properties of topological groups.

73.1 Separation properties of topological groups

The topological groups have very specific properties. For instance, G is T_0 iff it is T_3.

```
theorem(in topgroup) cl˙point:
  assumes x∈G
  shows cl(–x″) = (⋂H∈𝒩0. x+H)
proof-
  {
    have c:cl(–x″) = (⋂H∈𝒩0. –x″+H) using cl˙topgroup assms by auto
    {
      fix H
      assume H∈𝒩0
      then have –x″+H=x+ H using interval˙add(3) assms
        by auto
      with ‹H∈𝒩0› have –x″+H∈–x+H. H∈𝒩0″ by auto
    }
    then have —x″+H. H∈𝒩0″⊆–x+H. H∈𝒩0″ by auto
    moreover
    {
      fix H
      assume H∈𝒩0
      then have –x″+H=x+ H using interval˙add(3) assms
        by auto
      with ‹H∈𝒩0› have x+ H∈—x″+H. H∈𝒩0″ by auto
    }
    then have –x+H. H∈𝒩0″⊆—x″+H. H∈𝒩0″ by auto
    ultimately have —x″+H. H∈𝒩0″=–x+H. H∈𝒩0″ by auto
    then have (⋂H∈𝒩0. –x″+H) = (⋂H∈𝒩0. x+H) by auto
    with c show cl(–x″)=(⋂H∈𝒩0. x+H) by auto
  }
qed
```

We prove the equivalence between T_0 and T_1 first.

```
theorem (in topgroup) neu˙closed˙imp˙T1:
  assumes –0″–is closed in″T
  shows T–is T1″
proof-
  {
    fix x z assume xG:x∈G and zG:z∈G and dis:x≠z
    then have clx:cl(–x″)=(⋂H∈𝒩0. x+H) using cl˙point by auto
    {
      fix y
      assume y∈cl(–x″)
      with clx have y∈(⋂H∈𝒩0. x+H) by auto
      then have t:∀H∈𝒩0. y∈x+H by auto
      from ‹y∈cl(–x″)› xG have yG:y∈G using Top˙3˙L11(1) G˙def by auto
      {
        fix H
        assume HNeig:H∈𝒩0
        with t have y∈x+H by auto
       then obtain n where y=x+n and n∈H unfolding ltrans˙def grop˙def LeftTrans-
lation˙def by auto
```

with HNeig have nG:n∈G unfolding zerohoods˙def by auto
from ‹y=x+n› and ‹n∈H› have (-x)+y∈H using group0.group0˙2˙L18(2) group0˙valid˙in˙tgroup xG nG yG unfolding grinv˙def grop˙def
by auto
}
then have el:(-x)+y∈(⋂$\mathcal{N}_0$) using zneigh˙not˙empty by auto
have cl(–**0**″)=(⋂H∈$\mathcal{N}_0$. **0**+H) using cl˙point zero˙in˙tgroup by auto
moreover
{
fix H assume H∈$\mathcal{N}_0$
then have H⊆G unfolding zerohoods˙def by auto
then have **0**+H=H using image˙id˙same group0.trans˙neutral(2) group0˙valid˙in˙tgroup unfolding gzero˙def ltrans˙def
by auto
with ‹H∈$\mathcal{N}_0$› have **0**+H∈$\mathcal{N}_0$ H∈–**0**+H. H∈$\mathcal{N}_0$″ by auto
}
then have –**0**+H. H∈$\mathcal{N}_0$″=$\mathcal{N}_0$ by blast
ultimately have cl(–**0**″)=(⋂$\mathcal{N}_0$) by auto
with el have (-x)+y∈cl(–**0**″) by auto
then have (-x)+y∈–**0**″ using assms Top˙3˙L8 G˙def zero˙in˙tgroup by auto
then have (-x)+y=**0** by auto
then have y=-(-x) using group0.group0˙2˙L9(2) group0˙valid˙in˙tgroup neg˙in˙tgroup xG yG unfolding grop˙def grinv˙def by auto
then have y=x using group0.group˙inv˙of˙inv group0˙valid˙in˙tgroup xG unfolding grinv˙def by auto
}
then have cl(–x″)⊆–x″ by auto
then have cl(–x″)=–x″ using xG cl˙contains˙set G˙def by blast
then have –x″–is closed in″T using Top˙3˙L8 xG G˙def by auto
then have (⋃T)-–x″∈T using IsClosed˙def by auto moreover
from dis zG G˙def have z∈((⋃T)-–x″) ∧ x∉((⋃T)-–x″) by auto
ultimately have ∃V∈T. z∈V∧x∉V by(safe,auto)
}
then show T–is T_1″ using isT1˙def by auto
qed

theorem (in topgroup) T0˙imp˙neu˙closed:
assumes T–is T_0″
shows –**0**″–is closed in″T
proof-
{
fix x assume x∈cl(–**0**″) and x≠**0**
have cl(–**0**″)=(⋂H∈$\mathcal{N}_0$. **0**+H) using cl˙point zero˙in˙tgroup by auto
moreover
{
fix H assume H∈$\mathcal{N}_0$
then have H⊆G unfolding zerohoods˙def by auto
then have **0**+H=H using image˙id˙same group0.trans˙neutral(2) group0˙valid˙in˙tgroup unfolding gzero˙def ltrans˙def

by auto
with ‹H∈$\mathcal{N}_0$› have **0**+H∈$\mathcal{N}_0$ H∈–**0**+H. H∈$\mathcal{N}_0$″ by auto
}
then have –**0**+H. H∈$\mathcal{N}_0$″=$\mathcal{N}_0$ by blast
ultimately have cl(–**0**″)=(⋂$\mathcal{N}_0$) by auto
from ‹x≠**0**› and ‹x∈cl(–**0**″)› obtain U where U∈T and (x∉U∧**0**∈U)∨(**0**∉U∧x∈U)
using assms Top˙3˙L11(1)
zero˙in˙tgroup unfolding isT0˙def G˙def by blast moreover
{
assume **0**∈U
with ‹U∈T› have U∈$\mathcal{N}_0$ using zerohoods˙def G˙def Top˙2˙L3 by auto
with ‹x∈cl(–**0**″)› and ‹cl(–**0**″)=(⋂$\mathcal{N}_0$)› have x∈U by auto
}
ultimately have **0**∉U and x∈U by auto
with ‹U∈T› ‹x∈cl(–**0**″)› have False using cl˙inter˙neigh zero˙in˙tgroup unfolding
G˙def by blast
}
then have cl(–**0**″)⊆–**0**″ by auto
then have cl(–**0**″)=–**0**″ using zero˙in˙tgroup cl˙contains˙set G˙def by blast
then show thesis using Top˙3˙L8 zero˙in˙tgroup unfolding G˙def by auto
qed

73.2 Existence of nice neighbourhoods.

lemma (in topgroup) exist˙basehoods˙closed:
assumes U∈$\mathcal{N}_0$
shows ∃V∈$\mathcal{N}_0$. cl(V)⊆U
proof-
from assms obtain V where V∈$\mathcal{N}_0$ V⊆U (V+V)⊆U (-V)=V using exists˙procls˙zerohood
by blast
have inv˙fun:GroupInv(G,f)∈G→G using group0˙2˙T2 Ggroup by auto
have f˙fun:f∈G×G→G using group0.group˙oper˙assocA group0˙valid˙in˙tgroup by
auto
{
fix x assume x∈cl(V)
with ‹V∈$\mathcal{N}_0$› have x∈⋃T V⊆⋃T using Top˙3˙L11(1) unfolding zerohoods˙def
G˙def by blast+
with ‹V∈$\mathcal{N}_0$› have x∈int(x+V) using elem˙in˙int˙ltrans G˙def by auto
with ‹V⊆⋃T›‹x∈cl(V)› have int(x+V)∩V≠0 using cl˙inter˙neigh Top˙2˙L2 by
blast
then have (x+V)∩V≠0 using Top˙2˙L1 by blast
then obtain q where q∈(x+V) and q∈V by blast
with ‹V⊆⋃T›‹x∈⋃T› obtain v where q=x+v v∈V unfolding ltrans˙def grop˙def
using group0.ltrans˙image
group0˙valid˙in˙tgroup unfolding G˙def by auto
from ‹V⊆⋃T› ‹v∈V›‹q∈V› have v∈⋃T q∈⋃T by auto
with ‹q=x+v›‹x∈⋃T› have q-v=x using group0.group0˙2˙L18(1) group0˙valid˙in˙tgroup
unfolding G˙def
unfolding grsub˙def grinv˙def grop˙def by auto moreover

from ‹v∈V› have (-v)∈(-V) unfolding setninv˙def grinv˙def using func˙imagedef inv˙fun ‹V⊆⋃T› G˙def by auto
then have (-v)∈V using ‹(-V)=V› by auto
with ‹q∈V› have ⟨q,-v⟩∈V×V by auto
then have f⟨q,-v⟩∈V+V using lift˙subset˙suff f˙fun ‹V⊆⋃T› unfolding setadd˙def by auto
with ‹V+V⊆U› have q-v∈U unfolding grsub˙def grop˙def by auto
with ‹q-v=x› have x∈U by auto
}
then have cl(V)⊆U by auto
with ‹V∈$\mathcal{N}_0$› show thesis by auto
qed

73.3 Rest of separation axioms

theorem(in topgroup) T1˙imp˙T2:
assumes T–is T_1˝
shows T–is T_2˝
proof-
{
fix x y assume ass:x∈⋃T y∈⋃T x≠y
{
assume (-y)+x=**0**
with ass(1,2) have y=x using group0.group0˙2˙L11[where a=y and b=x] group0˙valid˙in˙tgroup by auto
with ass(3) have False by auto
}
then have (-y)+x≠**0** by auto
then have **0**≠(-y)+x by auto
from ‹y∈⋃T› have (-y)∈⋃T using neg˙in˙tgroup G˙def by auto
with ‹x∈⋃T› have (-y)+x∈⋃T using group0.group˙op˙closed[where a=-y and b=x] group0˙valid˙in˙tgroup unfolding
G˙def by auto
with assms ‹**0**≠(-y)+x› obtain U where U∈T and (-y)+x∉U and **0**∈U unfolding isT1˙def using zero˙in˙tgroup
by auto
then have U∈$\mathcal{N}_0$ unfolding zerohoods˙def G˙def using Top˙2˙L3 by auto
then obtain Q where Q∈$\mathcal{N}_0$ Q⊆U (Q+Q)⊆U (-Q)=Q using exists˙procls˙zerohood by blast
with ‹(-y)+x∉U› have (-y)+x∉Q by auto
from ‹Q∈$\mathcal{N}_0$› have Q⊆G unfolding zerohoods˙def by auto
{
assume x∈y+Q
with ‹Q⊆G› ‹y∈⋃T› obtain u where u∈Q and x=y+u unfolding ltrans˙def grop˙def using group0.ltrans˙image group0˙valid˙in˙tgroup
unfolding G˙def by auto
with ‹Q⊆G› have u∈⋃T unfolding G˙def by auto
with ‹x=y+u› ‹y∈⋃T› ‹x∈⋃T› ‹Q⊆G› have (-y)+x=u using group0.group0˙2˙L18(2) group0˙valid˙in˙tgroup unfolding G˙def

```
      unfolding grsub˙def grinv˙def grop˙def by auto
    with ‹u∈Q› have (-y)+x∈Q by auto
    then have False using ‹(-y)+x∉Q› by auto
  }
  then have x∉y+Q by auto moreover
  {
    assume y∈x+Q
     with ‹Q⊆G› ‹x∈⋃T› obtain u where u∈Q and y=x+u unfolding ltrans˙def
grop˙def using group0.ltrans˙image group0˙valid˙in˙tgroup
      unfolding G˙def by auto
    with ‹Q⊆G› have u∈⋃T unfolding G˙def by auto
  with ‹y=x+u› ‹y∈⋃T› ‹x∈⋃T› ‹Q⊆G› have (-x)+y=u using group0.group0˙2˙L18(2)
group0˙valid˙in˙tgroup unfolding G˙def
      unfolding grsub˙def grinv˙def grop˙def by auto
  with ‹u∈Q› have (-y)+x=-u using group0.group˙inv˙of˙two[OF group0˙valid˙in˙tgroup
group0.inverse˙in˙group[OF group0˙valid˙in˙tgroup,of x],of y]
    using ‹x∈⋃T› ‹y∈⋃T› using group0.group˙inv˙of˙inv[OF group0˙valid˙in˙tgroup]
unfolding G˙def grinv˙def grop˙def by auto
      moreover from ‹u∈Q› have (-u)∈(-Q) unfolding setninv˙def grinv˙def using
func˙imagedef[OF group0˙2˙T2[OF Ggroup] ‹Q⊆G›] by auto
     ultimately have (-y)+x∈Q using ‹(-y)+x∉Q› ‹(-Q)=Q› unfolding setninv˙def
grinv˙def by auto
    then have False using ‹(-y)+x∉Q› by auto
  }
  then have y∉x+Q by auto moreover
  {
    fix t
    assume t∈(x+Q)∩(y+Q)
    then have t∈(x+Q) t∈(y+Q) by auto
    with ‹Q⊆G› ‹x∈⋃T› ‹y∈⋃T› obtain u v where u∈Q v∈Q and t=x+u t=y+v
unfolding ltrans˙def grop˙def using group0.ltrans˙image[OF group0˙valid˙in˙tgroup]
      unfolding G˙def by auto
    then have x+u=y+v by auto
     moreover from ‹u∈Q› ‹v∈Q› ‹Q⊆G› have u∈⋃T v∈⋃T unfolding G˙def by
auto
    moreover note ‹x∈⋃T› ‹y∈⋃T›
  ultimately have (-y)+(x+u)=v using group0.group0˙2˙L18(2)[OF group0˙valid˙in˙tgroup,
of y v x+u] group0.group˙op˙closed[OF group0˙valid˙in˙tgroup, of x u] unfolding
G˙def
      unfolding grsub˙def grinv˙def grop˙def by auto
  then have ((-y)+x)+u=v using group0.group˙oper˙assoc[OF group0˙valid˙in˙tgroup]
    unfolding grop˙def using ‹x∈⋃T› ‹y∈⋃T› ‹u∈⋃T› using group0.inverse˙in˙group[OF
group0˙valid˙in˙tgroup] unfolding G˙def
      by auto
  then have ((-y)+x)=v-u using group0.group0˙2˙L18(1)[OF group0˙valid˙in˙tgroup,of
(-y)+x u v]
        using ‹(-y)+x∈⋃T› ‹u∈⋃T› ‹v∈⋃T› unfolding G˙def grsub˙def grinv˙def
grop˙def by force
    moreover
```

```
    from ‹u∈Q› have (-u)∈(-Q) unfolding setninv˙def grinv˙def using func˙imagedef[OF
group0˙2˙T2[OF Ggroup] ‹Q⊆G›] by auto
      then have (-u)∈Q using ‹(-Q)=Q› by auto
      with ‹v∈Q› have ⟨v,-u⟩∈Q×Q by auto
     then have f⟨v,-u⟩∈Q+Q using lift˙subset˙suff[OF group0.group˙oper˙assocA[OF
group0˙valid˙in˙tgroup] ‹Q⊆G› ‹Q⊆G›]
        unfolding setadd˙def by auto
      with ‹Q+Q⊆U› have v-u∈U unfolding grsub˙def grop˙def by auto
      ultimately have (-y)+x∈U by auto
      with ‹(-y)+x∉U› have False by auto
    }
    then have (x+Q)∩(y+Q)=0 by auto
    moreover have x∈int(x+Q)y∈int(y+Q) using elem˙in˙int˙ltrans ‹Q∈𝒩0›
      ‹x∈⋃T› ‹y∈⋃T› unfolding G˙def by auto moreover
    have int(x+Q)⊆(x+Q)int(y+Q)⊆(y+Q) using Top˙2˙L1 by auto
    moreover have int(x+Q)∈T int(y+Q)∈T using Top˙2˙L2 by auto
    ultimately have int(x+Q)∈T ∧ int(y+Q)∈T ∧ x ∈ int(x+Q) ∧ y ∈ int(y+Q) ∧
int(x+Q) ∩ int(y+Q) = 0
      by blast
    then have ∃U∈T. ∃V∈T. x∈U∧y∈V∧U∩V=0 by auto
  }
  then show thesis using isT2˙def by auto
qed
```

Here follow some auxiliary lemmas.

```
lemma (in topgroup) trans˙closure:
  assumes x∈G A⊆G
  shows cl(x+A)=x+cl(A)
proof-
  have ⋃T-(⋃T-(x+A))=(x+A) unfolding ltrans˙def using group0.group0˙5˙L1(2)[OF
group0˙valid˙in˙tgroup assms(1)]
     unfolding image˙def range˙def domain˙def converse˙def Pi˙def by auto
   then have cl(x+A)=⋃T-int(⋃T-(x+A)) using Top˙3˙L11(2)[of ⋃T-(x+A)] by
auto moreover
   have x+G=G using surj˙image˙eq group0.trans˙bij(2)[OF group0˙valid˙in˙tgroup
assms(1)] bij˙def by auto
   then have ⋃T-(x+A)=x+(⋃T-A) using inj˙image˙dif[of LeftTranslation(G, f,
x)GG, OF ˙ assms(2)]
     unfolding ltrans˙def G˙def using group0.trans˙bij(2)[OF group0˙valid˙in˙tgroup
assms(1)] bij˙def by auto
  then have int(⋃T-(x+A))=int(x+(⋃T-A)) by auto
   then have int(⋃T-(x+A))=x+int(⋃T-A) using ltrans˙interior[OF assms(1),of
⋃T-A] unfolding G˙def by force
  have ⋃T-int(⋃T-A)=cl(⋃T-(⋃T-A)) using Top˙3˙L11(2)[of ⋃T-A] by force
  have ⋃T-(⋃T-A)=A using assms(2) G˙def by auto
  with ‹⋃T-int(⋃T-A)=cl(⋃T-(⋃T-A))› have ⋃T-int(⋃T-A)=cl(A) by auto
  have ⋃T-(⋃T-int(⋃T-A))=int(⋃T-A) using Top˙2˙L2 by auto
  with ‹⋃T-int(⋃T-A)=cl(A)› have int(⋃T-A)=⋃T-cl(A) by auto
   with ‹int(⋃T-(x+A))=x+int(⋃T-A)› have int(⋃T-(x+A))=x+(⋃T-cl(A)) by
```

```
auto
  with ‹x+G=G› have int(⋃T-(x+A))=⋃T-(x+cl(A)) using inj˙image˙dif[of Left-
Translation(G, f, x)GGcl(A)]
   unfolding ltrans˙def using group0.trans˙bij(2)[OF group0˙valid˙in˙tgroup assms(1)]
Top˙3˙L11(1) assms(2) unfolding bij˙def G˙def
    by auto
  then have ⋃T-int(⋃T-(x+A))=⋃T-(⋃T-(x+cl(A))) by auto
 then have ⋃T-int(⋃T-(x+A))=x+cl(A) unfolding ltrans˙def using group0.group0˙5˙L1(2)[OF
group0˙valid˙in˙tgroup assms(1)]
    unfolding image˙def range˙def domain˙def converse˙def Pi˙def by auto
  with ‹cl(x+A)=⋃T-int(⋃T-(x+A))› show thesis by auto
qed

lemma (in topgroup) trans˙interior2: assumes A1: g∈G and A2: A⊆G
  shows int(A)+g = int(A+g)
proof -
  from assms have A ⊆ ⋃T and IsAhomeomorphism(T,T,RightTranslation(G,f,g))
    using tr˙homeo by auto
  then show thesis using int˙top˙invariant by simp
qed

lemma (in topgroup) trans˙closure2:
  assumes x∈G A⊆G
  shows cl(A+x)=cl(A)+x
proof-
 have ⋃T-(⋃T-(A+x))=(A+x) unfolding ltrans˙def using group0.group0˙5˙L1(1)[OF
group0˙valid˙in˙tgroup assms(1)]
    unfolding image˙def range˙def domain˙def converse˙def Pi˙def by auto
   then have cl(A+x)=⋃T-int(⋃T-(A+x)) using Top˙3˙L11(2)[of ⋃T-(A+x)] by
auto moreover
  have G+x=G using surj˙image˙eq group0.trans˙bij(1)[OF group0˙valid˙in˙tgroup
assms(1)] bij˙def by auto
  then have ⋃T-(A+x)=(⋃T-A)+x using inj˙image˙dif[of RightTranslation(G, f,
x)GG, OF ˙ assms(2)]
    unfolding rtrans˙def G˙def using group0.trans˙bij(1)[OF group0˙valid˙in˙tgroup
assms(1)] bij˙def by auto
  then have int(⋃T-(A+x))=int((⋃T-A)+x) by auto
   then have int(⋃T-(A+x))=int(⋃T-A)+x using trans˙interior2[OF assms(1),of
⋃T-A] unfolding G˙def by force
  have ⋃T-int(⋃T-A)=cl(⋃T-(⋃T-A)) using Top˙3˙L11(2)[of ⋃T-A] by force
  have ⋃T-(⋃T-A)=A using assms(2) G˙def by auto
  with ‹⋃T-int(⋃T-A)=cl(⋃T-(⋃T-A))› have ⋃T-int(⋃T-A)=cl(A) by auto
  have ⋃T-(⋃T-int(⋃T-A))=int(⋃T-A) using Top˙2˙L2 by auto
  with ‹⋃T-int(⋃T-A)=cl(A)› have int(⋃T-A)=⋃T-cl(A) by auto
   with ‹int(⋃T-(A+x))=int(⋃T-A)+x› have int(⋃T-(A+x))=(⋃T-cl(A))+x by
auto
  with ‹G+x=G› have int(⋃T-(A+x))=⋃T-(cl(A)+x) using inj˙image˙dif[of Right-
Translation(G, f, x)GGcl(A)]
   unfolding rtrans˙def using group0.trans˙bij(1)[OF group0˙valid˙in˙tgroup assms(1)]
```

```
Top˙3˙L11(1) assms(2) unfolding bij˙def G˙def
    by auto
  then have ⋃T-int(⋃T-(A+x))=⋃T-(⋃T-(cl(A)+x)) by auto
 then have ⋃T-int(⋃T-(A+x))=cl(A)+x unfolding ltrans˙def using group0.group0˙5˙L1(1)[OF
group0˙valid˙in˙tgroup assms(1)]
    unfolding image˙def range˙def domain˙def converse˙def Pi˙def by auto
  with ‹cl(A+x)=⋃T-int(⋃T-(A+x))› show thesis by auto
qed

lemma (in topgroup) trans˙subset:
  assumes A⊆((-x)+B)x∈GA⊆GB⊆G
  shows x+A⊆B
proof-
  {
   fix t assume t∈x+A
     with ‹x∈G› ‹A⊆G› obtain u where u∈A t=x+u unfolding ltrans˙def grop˙def
using group0.ltrans˙image[OF group0˙valid˙in˙tgroup]
      unfolding G˙def by auto
      with ‹x∈G› ‹A⊆G› ‹u∈A› have (-x)+t=u using group0.group0˙2˙L18(2)[OF
group0˙valid˙in˙tgroup, of xut]
       group0.group˙op˙closed[OF group0˙valid˙in˙tgroup,of x u] unfolding grop˙def
grinv˙def by auto
    with ‹u∈A› have (-x)+t∈A by auto
    with ‹A⊆(-x)+B› have (-x)+t∈(-x)+B by auto
     with ‹B⊆G› obtain v where (-x)+t=(-x)+v v∈B unfolding ltrans˙def grop˙def
using neg˙in˙tgroup[OF ‹x∈G›] group0.ltrans˙image[OF group0˙valid˙in˙tgroup]
      unfolding G˙def by auto
   have LeftTranslation(G,f,-x)∈inj(G,G) using group0.trans˙bij(2)[OF group0˙valid˙in˙tgroup
neg˙in˙tgroup[OF ‹x∈G›]] bij˙def by auto
   then have eq:∀A∈G. ∀B∈G. LeftTranslation(G,f,-x)A=LeftTranslation(G,f,-x)B
⟶ A=B unfolding inj˙def by auto
    {
      fix A B assume A∈GB∈G
      assume f⟨-x,A⟩=f⟨-x,B⟩
    then have LeftTranslation(G,f,-x)A=LeftTranslation(G,f,-x)B using group0.group0˙5˙L2(2)[OF
group0˙valid˙in˙tgroup neg˙in˙tgroup[OF ‹x∈G›]]
        ‹A∈G›‹B∈G› by auto
      with eq ‹A∈G›‹B∈G› have A=B by auto
    }
    then have eq1:∀A∈G. ∀B∈G. f⟨-x,A⟩=f⟨-x,B⟩ ⟶ A=B by auto
    from ‹A⊆G› ‹u∈A› have u∈G by auto
     with ‹v∈B› ‹B⊆G› ‹t=x+u› have t∈G v∈G using group0.group˙op˙closed[OF
group0˙valid˙in˙tgroup ‹x∈G›,of u] unfolding grop˙def
      by auto
    with eq1 ‹(-x)+t=(-x)+v› have t=v unfolding grop˙def by auto
    with ‹v∈B› have t∈B by auto
  }
  then show thesis by auto
qed
```

Every topological group is regular, and hence T_3. The proof is in the next section, since it uses local properties.

73.4 Local properties

In a topological group, all local properties depend only on the neighbourhoods of the neutral element; when considering topological properties. The next result of regularity, will use this idea, since translations preserve closed sets.

```
lemma (in topgroup) local˙iff˙neutral:
  assumes ∀U∈T∩𝒩0. ∃N∈𝒩0. N⊆U∧ P(N,T) ∀N∈Pow(G). ∀x∈G. P(N,T) ⟶
P(x+N,T)
  shows T–is locally″P
proof-
  {
    fix x U assume x∈⋃TU∈Tx∈U
    then have (-x)+U∈T∩𝒩0 using open˙tr˙open(1) open˙trans˙neigh neg˙in˙tgroup
unfolding G˙def
      by auto
    with assms(1) obtain N where N⊆((-x)+U)P(N,T)N∈𝒩0 by auto
    note ‹x∈⋃T›‹N⊆((-x)+U)› moreover
    from ‹U∈T› have U⊆⋃T by auto moreover
    from ‹N∈𝒩0› have N⊆G unfolding zerohoods˙def by auto
    ultimately have (x+N)⊆U using trans˙subset unfolding G˙def by auto moreover
     from ‹N⊆G›‹x∈⋃T› assms(2) ‹P(N,T)› have P((x+N),T) unfolding G˙def by
auto moreover
    from ‹N∈𝒩0›‹x∈⋃T› have x∈int(x+N) using elem˙in˙int˙ltrans unfolding G˙def
by auto
    ultimately have ∃N∈Pow(U). x∈int(N)∧P(N,T) by auto
  }
  then show thesis unfolding IsLocally˙def[OF topSpaceAssum] by auto
qed

lemma (in topgroup) trans˙closed:
  assumes A–is closed in″Tx∈G
  shows (x+A)–is closed in″T
proof-
  from assms(1) have cl(A)=A using Top˙3˙L8 unfolding IsClosed˙def by auto
  then have x+cl(A)=x+A by auto
  then have cl(x+A)=x+A using trans˙closure assms unfolding IsClosed˙def by auto
  moreover have x+A⊆G unfolding ltrans˙def using group0.group0˙5˙L1(2)[OF group0˙valid˙in˙tgroup
‹x∈G›]
      unfolding image˙def range˙def domain˙def converse˙def Pi˙def by auto
  ultimately show thesis using Top˙3˙L8 unfolding G˙def by auto
qed
```

As it is written in the previous section, every topological group is regular.

```
theorem (in topgroup) topgroup˙reg:
```

shows T–is regular″
proof-
{
fix U assume U∈T∩$\mathcal{N}_0$
then obtain V where cl(V)⊆UV∈$\mathcal{N}_0$ using exist˙basehoods˙closed by blast
then have V⊆cl(V) using cl˙contains˙set unfolding zerohoods˙def G˙def by auto
then have int(V)⊆int(cl(V)) using interior˙mono by auto
with ‹V∈$\mathcal{N}_0$› have cl(V)∈$\mathcal{N}_0$ unfolding zerohoods˙def G˙def using Top˙3˙L11(1) by auto
from ‹V∈$\mathcal{N}_0$› have cl(V)–is closed in″T using cl˙is˙closed unfolding zerohoods˙def G˙def by auto
with ‹cl(V)∈$\mathcal{N}_0$›‹cl(V)⊆U› have ∃N∈$\mathcal{N}_0$. N⊆U∧N–is closed in″T by auto
}
then have ∀U∈T∩$\mathcal{N}_0$. ∃N∈$\mathcal{N}_0$. N⊆U∧N–is closed in″T by auto moreover
have ∀N∈Pow(G).(∀x∈G. (N–is closed in″T⟶(x+N)–is closed in″T)) using trans˙closed by auto
ultimately have T–is locally-closed″ using local˙iff˙neutral unfolding IsLocally-Closed˙def by auto
then show T–is regular″ using regular˙locally˙closed by auto
qed

The promised corollary follows:

corollary (in topgroup) T2˙imp˙T3:
assumes T–is T_2″
shows T–is T_3″ using T2˙is˙T1 topgroup˙reg isT3˙def assms by auto

end

74 Topological groups - uniformity

theory TopologicalGroup˙Uniformity˙ZF imports TopologicalGroup˙ZF UniformSpace˙ZF˙1

begin

Each topological group is a uniform space. This theory is about the unifomities that are naturally defined by a topological group structure.

74.1 Natural uniformities in topological groups: definitions and notation

There are two basic uniformities that can be defined on a topological group.

Definition of left uniformity

definition (in topgroup) leftUniformity
where leftUniformity ≡ –V∈Pow(G×G).∃U∈ $\mathcal{N}_0$. –⟨s,t⟩∈G×G. (-s)+t ∈U″ ⊆ V″

Definition of right uniformity

definition (in topgroup) rightUniformity
where rightUniformity ≡ –V∈Pow(G×G).∃ U∈ $\mathcal{N}_0$. –⟨s,t⟩∈G×G. s+(-t) ∈U″⊆ V″

Right and left uniformities are indeed uniformities.

lemma (in topgroup) side˙uniformities:
shows leftUniformity –is a uniformity on″ G and rightUniformity –is a uniformity on″ G
proof-
{
assume 0 ∈ leftUniformity
then obtain U where U:U∈$\mathcal{N}_0$ –⟨s,t⟩∈G×G. (-s)+t ∈U″⊆0 unfolding leftUniformity˙def
by auto
have ⟨**0**,**0**⟩:G×G using zero˙in˙tgroup by auto
moreover have (-**0**)+**0** = **0**
using group0˙valid˙in˙tgroup group0.group˙inv˙of˙one group0.group0˙2˙L2 zero˙in˙tgroup

by auto
moreover have **0**∈int(U) using U(1) by auto
then have **0**∈U using Top˙2˙L1 by auto
ultimately have ⟨**0**,**0**⟩∈–⟨s,t⟩∈G×G. (-s)+t ∈U″ by auto
with U(2) have ⟨**0**,**0**⟩∈0 by blast
hence False by auto
}
hence 0∉leftUniformity by auto
moreover have leftUniformity ⊆ Pow(G×G) unfolding leftUniformity˙def by auto

moreover
{
have G×G∈Pow(G×G) by auto moreover
have –⟨s,t⟩:G×G. (-s)+t:G″ ⊆ G×G by auto moreover
note zneigh˙not˙empty
ultimately have G×G∈leftUniformity unfolding leftUniformity˙def by auto
}
moreover
{
fix A B assume as:A∈leftUniformity B∈leftUniformity
from as(1) obtain AU where AU:AU∈ $\mathcal{N}_0$ –⟨s,t⟩∈G×G. (-s)+t ∈AU″⊆ A A∈Pow(G×G)
unfolding leftUniformity˙def by auto
from as(2) obtain BU where BU:BU∈ $\mathcal{N}_0$ –⟨s,t⟩∈G×G. (-s)+t ∈BU″⊆ B B∈Pow(G×G)
unfolding leftUniformity˙def by auto
from AU(1) BU(1) have **0**∈int(AU)∩int(BU) by auto
moreover from AU BU have op:int(AU)∩int(BU)∈T using Top˙2˙L2 topSpaceAssum IsATopology˙def
by auto
moreover
have int(AU)∩int(BU) ⊆ AU∩BU using Top˙2˙L1 by auto

with op have int(AU)∩int(BU)⊆int(AU∩BU) using Top˙2˙L5 by auto
moreover note AU(1) BU(1)
ultimately have AU∩BU: $\mathcal{N}_0$ unfolding zerohoods˙def by auto
moreover have –⟨s,t⟩∈G×G. (-s)+t ∈AU∩BU″⊆–⟨s,t⟩∈G×G. (-s)+t ∈AU″ by auto
with AU(2) BU(2) have –⟨s,t⟩∈G×G. (-s)+t ∈AU∩BU″⊆A∩B by auto
ultimately have A∩B ∈ –V∈Pow(G×G).∃U∈ $\mathcal{N}_0$. –⟨s,t⟩∈G×G. (-s)+t ∈U″ ⊆ V″
using AU(3) BU(3) by blast
then have A∩B∈leftUniformity unfolding leftUniformity˙def by simp
}
hence ∀A∈leftUniformity. ∀B∈leftUniformity. A ∩ B ∈ leftUniformity by auto
moreover
{
fix B C assume as:B∈leftUniformity C∈Pow(G × G) B ⊆ C
from as(1) obtain BU where BU:BU∈$\mathcal{N}_0$ –⟨s,t⟩∈G×G. (-s)+t ∈ BU″⊆B
unfolding leftUniformity˙def by blast
from as(3) BU(2) have –⟨s,t⟩∈G×G. (-s)+t ∈ BU″⊆C by auto
with as(2) BU(1) have C ∈ –V∈Pow(G×G).∃U∈ $\mathcal{N}_0$. –⟨s,t⟩∈G×G. (-s)+t ∈U″ ⊆ V″
by auto
then have C ∈ leftUniformity unfolding leftUniformity˙def by auto
}
then have ∀B∈leftUniformity. ∀C∈Pow(G×G). B⊆C ⟶ C∈leftUniformity by auto
ultimately have leftFilter:leftUniformity –is a filter on″ (G×G) unfolding IsFilter˙def
by auto
{
assume 0∈rightUniformity
then obtain U where U:U∈$\mathcal{N}_0$ –⟨s,t⟩∈G×G. s+(-t) ∈U″⊆0 unfolding rightUniformity˙def
by auto
have ⟨**0**,**0**⟩:G×G using zero˙in˙tgroup by auto
moreover have **0**+(-**0**) = **0**
using group0˙valid˙in˙tgroup group0.group˙inv˙of˙one group0.group0˙2˙L2 zero˙in˙tgroup

by auto
moreover
have **0**∈int(U) using U(1) by auto
then have **0**∈U using Top˙2˙L1 by auto
ultimately have ⟨**0**,**0**⟩∈–⟨s,t⟩∈G×G. s+(-t) ∈U″ by auto
with U(2) have ⟨**0**,**0**⟩∈0 by blast
hence False by auto
}
then have 0∉rightUniformity by auto
moreover have rightUniformity ⊆ Pow(G×G) unfolding rightUniformity˙def by auto
moreover

{
have G×G∈Pow(G×G) by auto
moreover have −⟨s,t⟩:G×G. (-s)+t:G″ ⊆ G×G by auto
moreover note zneigh˙not˙empty
ultimately have G×G ∈ rightUniformity unfolding rightUniformity˙def by auto
}
moreover
{
fix A B assume as:A∈rightUniformity B∈rightUniformity
from as(1) obtain AU where AU:AU∈ $\mathcal{N}_0$ −⟨s,t⟩∈G×G. s+(-t) ∈AU″⊆ A A∈Pow(G×G)
unfolding rightUniformity˙def by auto
from as(2) obtain BU where BU:BU∈ $\mathcal{N}_0$ −⟨s,t⟩∈G×G. s+(-t) ∈BU″⊆ B B∈Pow(G×G)
unfolding rightUniformity˙def by auto
from AU(1) BU(1) have **0**∈int(AU)∩int(BU) by auto
moreover from AU BU have op:int(AU)∩int(BU)∈T
using Top˙2˙L2 topSpaceAssum IsATopology˙def by auto
moreover
have int(AU)∩int(BU) ⊆ AU∩BU using Top˙2˙L1 by auto
with op have int(AU)∩int(BU)⊆int(AU∩BU) using Top˙2˙L5 by auto
moreover note AU(1) BU(1)
ultimately have AU∩BU: $\mathcal{N}_0$ unfolding zerohoods˙def by auto
moreover have −⟨s,t⟩∈G×G. s+(-t) ∈AU∩BU″⊆−⟨s,t⟩∈G×G. s+(-t) ∈AU″ by auto
with AU(2) BU(2) have −⟨s,t⟩∈G×G. s+(-t) ∈AU∩BU″⊆A∩B by auto
ultimately have A∩B ∈ −V∈Pow(G×G).∃ U∈ $\mathcal{N}_0$. −⟨s,t⟩∈G×G. s+(-t) ∈U″⊆ V″
using AU(3) BU(3) by blast
then have A∩B ∈ rightUniformity unfolding rightUniformity˙def by simp
}
hence ∀ A∈rightUniformity. ∀ B∈rightUniformity. A∩B ∈ rightUniformity by auto
moreover
{
fix B C assume as:B∈rightUniformity C∈Pow(G × G) B ⊆ C
from as(1) obtain BU where BU:BU∈$\mathcal{N}_0$ −⟨s,t⟩∈G×G. s+(-t) ∈ BU″⊆B
unfolding rightUniformity˙def by blast
from as(3) BU(2) have −⟨s,t⟩∈G×G. s+(-t) ∈ BU″⊆C by auto
then have C ∈ rightUniformity using as(2) BU(1) unfolding rightUniformity˙def by auto
}
then have ∀ B∈rightUniformity. ∀ C∈Pow(G×G). B⊆C ⟶ C∈rightUniformity by auto
ultimately have rightFilter:rightUniformity −is a filter on″ (G×G) unfolding IsFilter˙def
by auto
{
fix U assume as:U∈leftUniformity
from as obtain V where V:V∈$\mathcal{N}_0$ −⟨s,t⟩∈G×G. (-s)+t ∈ V″ ⊆ U

```
    unfolding leftUniformity˙def by auto
  then have V⊆G by auto
  {
    fix x assume as2:x∈id(G)
    from as obtain V where V:V∈𝒩0 −⟨s,t⟩∈G×G. (-s)+t ∈ V″ ⊆ U
      unfolding leftUniformity˙def by auto
    from V(1) have 0∈int(V) by auto
    then have V0:0∈V using Top˙2˙L1 by auto
    from as2 obtain t where t:x=⟨t,t⟩ t:G by auto
    from t(2) have (-t)+t =0 using group0˙valid˙in˙tgroup group0.group0˙2˙L6
      by auto
    with V0 t V(2) have x∈U by auto
  }
  then have id(G)⊆U by auto
  moreover
  {
    {
      fix x assume ass:x∈−⟨s,t⟩∈G×G. (-s)+t ∈ -V″
      then obtain s t where as:s∈G t∈G (-s)+t ∈ -V x=⟨s,t⟩ by force
      from as(3) ‹V⊆G› have (-s)+t∈−-q. q∈V″ using ginv˙image˙add by simp
      then obtain q where q: q∈V (-s)+t = -q by auto
      with ‹V⊆G› have q∈G by auto
      with ‹s∈G› ‹t∈G› ‹(-s)+t = -q› have q=(-t)+s
        using simple˙equation1˙add by blast
      with q(1) have (-t)+s ∈ V by auto
      with as(1,2) have ⟨t,s⟩ ∈ U using V(2) by auto
      then have ⟨s,t⟩ ∈ converse(U) by auto
      with as(4) have x ∈ converse(U) by auto
    }
    then have −⟨s,t⟩∈G×G. (-s)+t ∈ -V″ ⊆ converse(U) by auto
    moreover have (-V):𝒩0 using neg˙neigh˙neigh V(1) by auto
    moreover note as
    ultimately have converse(U) ∈ leftUniformity unfolding leftUniformity˙def by
auto
  }
  moreover
  {
   from V(1) obtain W where W:W:𝒩0 W + W ⊆V using exists˙procls˙zerohood
by blast
    {
      fix x assume as:x ∈ −⟨s,t⟩∈G×G. (-s)+t ∈ W″ O −⟨s,t⟩∈G×G. (-s)+t ∈ W″
      then obtain x1 x2 x3 where
        x:x1∈G x2∈G x3∈G (-x1)+x2 ∈ W (-x2)+x3 ∈ W x=⟨x1,x3⟩
        unfolding comp˙def by auto
      from W(1) have W+W = f(W×W) using interval˙add(2) by auto
      moreover from W(1) have WW:W×W⊆G×G by auto
      moreover
      from x(4,5) have ⟨(-x1)+x2,(-x2)+x3⟩:W×W by auto
      with WW have f(⟨(-x1)+x2,(-x2)+x3⟩):f(W×W)
```

using func˙imagedef topgroup˙f˙binop by auto
ultimately have $((-x_1)+x_2)+((-x_2)+x_3)$:W+W by auto
moreover from x(1,2,3) have $((-x_1)+x_2)+((-x_2)+x_3) = (-x_1)+ x_3$
using cancel˙middle˙add(2) by simp
ultimately have $(-x_1)+ x_3 \in$W+W by auto
with W(2) have $(-x_1)+ x_3 \in$V by auto
with x(1,3,6) have x:$-\langle$s,t$\rangle\in$G×G. (-s)+t ∈ V″ by auto
}
then have $-\langle$s,t$\rangle\in$G×G. (-s)+t ∈ W″ O $-\langle$s,t$\rangle\in$G×G. (-s)+t ∈ W″ ⊆ U
using V(2) by auto moreover
have $-\langle$s,t$\rangle\in$G×G. (-s)+t ∈ W″∈leftUniformity
unfolding leftUniformity˙def using W(1) by auto
ultimately have ∃ Z∈leftUniformity. Z O Z⊆U by auto
}
ultimately have id(G)⊆U ∧ (∃ Z∈leftUniformity. Z O Z⊆U) ∧ converse(U)∈leftUniformity

by blast
}
then have
∀ U∈leftUniformity. id(G)⊆U ∧ (∃ Z∈leftUniformity. Z O Z⊆U) ∧ converse(U)∈leftUniformity

by auto
with leftFilter show leftUniformity –is a uniformity on″ G unfolding IsUniformity˙def by auto
{
fix U assume as:U∈rightUniformity
from as obtain V where V:V∈$\mathcal{N}_0$ $-\langle$s,t$\rangle\in$G×G. s+(-t) ∈ V″ ⊆ U
unfolding rightUniformity˙def by auto
{
fix x assume as2:x∈id(G)
from as obtain V where V:V∈$\mathcal{N}_0$ $-\langle$s,t$\rangle\in$G×G. s+(-t) ∈ V″ ⊆ U
unfolding rightUniformity˙def by auto
from V(1) have **0**∈int(V) by auto
then have V0:**0**∈V using Top˙2˙L1 by auto
from as2 obtain t where t:x=$\langle$t,t$\rangle$ t:G by auto
from t(2) have t+(-t) =**0** using group0˙valid˙in˙tgroup group0.group0˙2˙L6
by auto
with V0 t V(2) have x∈U by auto
}
then have id(G)⊆U by auto
moreover
{
{
fix x assume ass:x∈$-\langle$s,t$\rangle\in$G×G. s+(-t) ∈ -V″
then obtain s t where as:s∈G t∈G s+(-t) ∈ -V x=$\langle$s,t$\rangle$
by force
from as(3) V(1) have s+(-t) ∈ −-q. q∈V″
using ginv˙image˙add by simp
then obtain q where q:q∈V s+(-t) = -q by auto

with ‹V∈$\mathcal{N}_0$› have q∈G by auto
with as(1,2) q(1,2) have t+(-s) ∈ V using simple˙equation0˙add
by blast
with as(1,2,4) V(2) have x ∈ converse(U) by auto
}
then have –⟨s,t⟩∈G×G. s+(-t) ∈ -V″ ⊆ converse(U) by auto
moreover from V(1) have (-V) ∈ $\mathcal{N}_0$ using neg˙neigh˙neigh by auto
ultimately have converse(U) ∈ rightUniformity using as rightUniformity˙def
by auto
}
moreover
{
from V(1) obtain W where W:W:$\mathcal{N}_0$ W + W ⊆V using exists˙procls˙zerohood
by blast
{
fix x assume as:x:–⟨s,t⟩∈G×G. s+(-t) ∈ W″ O –⟨s,t⟩∈G×G. s+(-t) ∈ W″
then obtain x_1 x_2 x_3 where
x:x_1:G x_2∈G x_3∈G x_1+(-x_2) ∈ W x_2+(-x_3) ∈ W x=⟨x_1,x_3⟩
unfolding comp˙def by auto
from W(1) have W+W = f(W×W) using interval˙add(2) by auto
moreover from W(1) have WW:W×W⊆G×G by auto
moreover
from x(4,5) have ⟨x_1+(-x_2),x_2+(-x_3)⟩ ∈ W×W by auto
with WW have f(⟨x_1+(-x_2),x_2+(-x_3)⟩) ∈ f(W×W)
using func˙imagedef topgroup˙f˙binop by auto
ultimately have (x_1+(-x_2))+(x_2+(-x_3)) ∈ W+W by auto
moreover from x(1,2,3) have (x_1+(-x_2))+(x_2+(-x_3)) = x_1+ (-x_3)
using cancel˙middle˙add(1) by simp
ultimately have x_1+(-x_3) ∈ W+W by auto
with W(2) have x_1+(-x_3) ∈ V by auto
then have x ∈ –⟨s,t⟩∈G×G. s+(-t) ∈ V″ using x(1,3,6) by auto
}
with V(2) have –⟨s,t⟩∈G×G. s+(-t) ∈ W″ O –⟨s,t⟩∈G×G. s+(-t) ∈ W″ ⊆ U
by auto
moreover from W(1) have –⟨s,t⟩∈G×G. s+(-t) ∈ W″ ∈ rightUniformity
unfolding rightUniformity˙def by auto
ultimately have ∃Z∈rightUniformity. Z O Z⊆U by auto
}
ultimately have id(G)⊆U ∧ (∃Z∈rightUniformity. Z O Z⊆U) ∧ converse(U)∈rightUniformity

by blast
}
then have
∀U∈rightUniformity. id(G)⊆U ∧ (∃Z∈rightUniformity. Z O Z⊆U) ∧ converse(U)∈rightUniformity
by auto
with rightFilter show rightUniformity –is a uniformity on″ G unfolding IsUniformity˙def
by auto

qed

The topologies generated by the right and left uniformities are the original group topology.

```
lemma (in topgroup) top˙generated˙side˙uniformities:
 shows UniformTopology(leftUniformity,G) = T and UniformTopology(rightUniformity,G)
= T
proof-
  let M = –⟨t, –V  –t″ . V ∈ leftUniformity″⟩. t∈G″
  have fun:M:G→Pow(Pow(G)) using neigh˙from˙uniformity side˙uniformities(1) Is-
NeighSystem˙def
    by auto
  let N = –⟨t, –V  –t″ . V ∈ rightUniformity″⟩. t∈G″
   have funN:N:G→Pow(Pow(G)) using neigh˙from˙uniformity side˙uniformities(2)
IsNeighSystem˙def
    by auto
  {
    fix U assume op:U∈T
    hence U⊆G by auto
    {
      fix x assume x:x∈U
      with op have xg:x∈G and (-x) ∈ G using neg˙in˙tgroup by auto
     then have ⟨x, –V–x″. V ∈ leftUniformity″⟩ ∈ –⟨t, –V–t″. V ∈ leftUniformity″⟩.
t∈G″
        by auto
    with fun have app:M(x) = –V–x″. V ∈ leftUniformity″ using ZF˙fun˙from˙tot˙val

        by auto
      have (-x)+U : 𝒩0 using open˙trans˙neigh op x by auto
      then have V:–⟨s,t⟩∈G×G. (-s)+t∈((-x)+U)″ ∈ leftUniformity
        unfolding leftUniformity˙def by auto
      with xg have
        N:∀ t∈G. t:–⟨s,t⟩∈G×G. (-s)+t∈((-x)+U)″–x″ ⟷ (-x)+t∈((-x)+U)
        using image˙iff by auto
      {
        fix t assume t:t∈G
        {
          assume as:(-x)+t∈((-x)+U)
          then have (-x)+t∈LeftTranslation(G,f,-x)U by auto
          then obtain q where q:q∈U ⟨q,(-x)+t⟩∈LeftTranslation(G,f,-x)
            using image˙iff by auto
          with op have q∈G by auto
          from q(2) have (-x)+q = (-x)+t unfolding LeftTranslation˙def
            by auto
         with ‹(-x) ∈ G› ‹q∈G› ‹t∈G› have q = t using neg˙in˙tgroup cancel˙left˙add
            by blast
          with q(1) have t∈U by auto
        }
        moreover
```

```
  {
    assume t:t∈U
    with ‹U⊆G› ‹(-x)∈G› have (-x)+t ∈ ((-x)+U)
      using lrtrans˙image(1) by auto
  }
  ultimately have (-x)+t∈((-x)+U) ⟷ t:U by blast
}
with N have ∀t∈G. t:–⟨s,t⟩∈G×G. (-s)+t ∈ ((-x)+U)″–x″ ⟷ t∈U
  by blast
with op have ∀t. t:–⟨s,t⟩∈G×G. (-s)+t∈((-x)+U)″–x″ ⟷ t:U
  by auto
hence U = –⟨s,t⟩∈G×G. (-s)+t∈((-x)+U)″–x″ by auto
with V have ∃V∈leftUniformity. U=V–x″ by blast
with app have U ∈ –⟨t, –V  –t″ . V ∈ leftUniformity″⟩ . t ∈ G″(x) by auto
moreover from ‹x∈G› funN have app:N(x) = –V–x″. V ∈ rightUniformity″
  using ZF˙fun˙from˙tot˙val by simp
moreover
from x op have openTrans:U+(-x): 𝒩0 using open˙trans˙neigh˙2 by auto
then have V:–⟨s,t⟩∈G×G. s+(-t)∈(U+(-x))″ ∈ rightUniformity
  unfolding rightUniformity˙def by auto
with xg have
  N:∀t∈G. t:–⟨s,t⟩∈G×G. s+(-t)∈(U+(-x))″-–x″ ⟷ t+(-x)∈(U+(-x))
  using vimage˙iff by auto
moreover
{
  fix t assume t:t∈G
  {
    assume as:t+(-x)∈(U+(-x))
    hence t+(-x)∈RightTranslation(G,f,-x)U by auto
    then obtain q where q:q∈U ⟨q,t+(-x)⟩∈RightTranslation(G,f,-x)
      using image˙iff by auto
    with op have q∈G by auto
    from q(2) have q+(-x) = t+(-x) unfolding RightTranslation˙def by auto
    with ‹q∈G› ‹(-x) ∈ G› ‹t∈G› have q = t using cancel˙right˙add by simp
    with q(1) have t∈U by auto
  }
  moreover
  {
    assume t∈U
    with ‹(-x)∈G› ‹U⊆G› have t+(-x)∈(U+(-x)) using lrtrans˙image(2)
      by auto
  } ultimately have t+(-x)∈(U+(-x)) ⟷ t:U by blast
} with N have ∀t∈G. t:–⟨s,t⟩∈G×G. s+(-t)∈(U+(-x))″-–x″ ⟷ t:U
  by blast
with op have ∀t. t:–⟨s,t⟩∈G×G. s+(-t)∈(U+(-x))″-–x″ ⟷ t:U
  by auto
hence –⟨s,t⟩∈G×G. s+(-t)∈(U+(-x))″-–x″ = U by auto
then have U = converse(–⟨s,t⟩∈G×G. s+(-t)∈(U+(-x))″)–x″
  unfolding vimage˙def by simp
```

with V app have U ∈ −⟨t, −V −t″ . V ∈ rightUniformity″⟩ . t ∈ G″(x)
using side˙uniformities(2) IsUniformity˙def by auto
ultimately have
U ∈ −⟨t, −V −t″ . V ∈ leftUniformity″⟩ . t ∈ G″(x) and
U ∈ −⟨t, −V −t″ . V ∈ rightUniformity″⟩ . t ∈ G″(x)
by auto
}
hence
∀x∈U. U ∈ −⟨t, −V −t″ . V ∈ leftUniformity″⟩ . t ∈ G″ x and
∀x∈U. U ∈ −⟨t, −V −t″ . V ∈ rightUniformity″⟩ . t ∈ G″ x
by auto
}
hence
T⊆−U ∈ Pow(G) . ∀x∈U. U ∈ −⟨t, −V −t″ . V ∈ leftUniformity″⟩ . t ∈ G″ x″ and
T⊆−U ∈ Pow(G) . ∀x∈U. U ∈ −⟨t, −V −t″ . V ∈ rightUniformity″⟩ . t ∈ G″ x″
by auto
moreover
{
fix U assume as:U ∈ Pow(G) ∀x∈U. U ∈ −⟨t, −V −t″ . V ∈ leftUniformity″⟩ . t ∈ G″(x)
{
fix x assume x:x∈U
with as(1) have xg:x∈G by auto
from x as(2) have U∈−⟨t, −V −t″ . V ∈ leftUniformity″⟩ . t ∈ G″(x) by auto
with xg fun have U∈−V −x″ . V ∈ leftUniformity″ using apply˙equality by auto
then obtain V where V:U=V−x″ V∈leftUniformity by auto
from V(2) obtain W where W:W∈ $\mathcal{N}_0$ −⟨s,t⟩:G×G. (-s)+t:W″⊆V
unfolding leftUniformity˙def by auto
from W(2) have A:−⟨s,t⟩:G×G. (-s)+t:W″−x″⊆V−x″ by auto
from xg have ∀q∈G. q∈(−⟨s,t⟩:G×G. (-s)+t:W″−x″) ⟷ (-x)+q:W
using image˙iff by auto
hence B:−⟨s,t⟩:G×G. (-s)+t:W″−x″ = −t∈G. (-x)+t:W″ by auto
from W(1) have WG:W⊆G by auto
{
fix t assume t:t ∈ x+W
then have t ∈ LeftTranslation(G,f,x)W by auto
then obtain s where s:s∈W ⟨s,t⟩∈LeftTranslation(G,f,x) using image˙iff by auto
with ‹W⊆G› have s∈G by auto
from s(2) have t=x+s t∈G unfolding LeftTranslation˙def by auto
with ‹x∈G› ‹s∈G› have (-x)+t = s using put˙on˙the˙other˙side(2) by simp
with s(1) have (-x)+t∈W by auto
with ‹t∈G› have t ∈ −s∈G. (-x)+s:W″ by auto
}
then have x+W ⊆ −t∈G. (-x)+t∈W″ by auto
with B have x + W ⊆ −⟨s,t⟩ ∈ G × G . (- s) + t ∈ W″ −x″ by auto

with A have x + W ⊆ V –x″ by blast
with V(1) have x + W ⊆ U by auto
then have int(x + W) ⊆ U using Top˙2˙L1 by blast
moreover from xg W(1) have x∈int(x + W) using elem˙in˙int˙ltrans(1) by auto
moreover have int(x + W)∈T using Top˙2˙L2 by auto
ultimately have ∃Y∈T. x∈Y ∧ Y⊆U by auto
}
then have U∈T using open˙neigh˙open by auto
} hence –U ∈ Pow(G) . ∀x∈U. U ∈ –⟨t, –V –t″ . V ∈ leftUniformity″⟩ . t ∈ G″ x″ ⊆ T
by auto
moreover
{
fix U assume as:U ∈ Pow(G) ∀x∈U. U ∈ –⟨t, –V –t″ . V ∈ rightUniformity″⟩ . t ∈ G″ x
{
fix x assume x:x∈U
with as(1) have xg:x∈G by auto
from x as(2) have U∈–⟨t, –V –t″ . V ∈ rightUniformity″⟩ . t ∈ G″ x by auto
with xg funN have U∈–V –x″ . V ∈ rightUniformity″ using apply˙equality
by auto
then obtain V where V:U=V–x″ V ∈ rightUniformity by auto
then have converse(V) ∈ rightUniformity using side˙uniformities(2) IsUniformity˙def
by auto
then obtain W where W:W∈ $\mathcal{N}_0$ –⟨s,t⟩:G×G. s+(-t):W″⊆converse(V)
unfolding rightUniformity˙def by auto
from W(2) have A:–⟨s,t⟩:G×G. s+(-t):W″-–x″⊆V–x″ by auto
from xg have ∀q∈G. q∈(–⟨s,t⟩:G×G. s+(-t):W″-–x″) ⟷ q+(-x):W
using image˙iff by auto
hence B:–⟨s,t⟩:G×G. s+(-t):W″-–x″ = –t∈G. t+(-x):W″ by auto
from W(1) have WG:W⊆G by auto
{
fix t assume t ∈ W+x
with ‹x∈G› ‹W⊆G› obtain s where s∈W and t=s+x using lrtrans˙image(2)
by auto
with ‹W⊆G› have s∈G by auto
with ‹x∈G› ‹t=s+x› have t∈G using group˙op˙closed˙add by simp
from ‹x∈G› ‹s∈G› ‹t=s+x› have t+(-x) = s using put˙on˙the˙other˙side
by simp
with ‹s∈W› ‹t∈G› have t ∈ –s∈G. s+(-x) ∈ W″ by auto
}
then have W+x ⊆ –t:G. t+(-x):W″ by auto
with B have W + x ⊆ –⟨s,t⟩ ∈ G × G . s + (- t) ∈ W″-–x″ by auto
with A have W + x ⊆ V –x″ by blast
with V(1) have W + x ⊆ U by auto
then have int(W + x) ⊆ U using Top˙2˙L1 by blast
moreover
from xg W(1) have x∈int(W + x) using elem˙in˙int˙rtrans(1) by auto

```
    moreover have int(W + x)∈T using Top˙2˙L2 by auto
    ultimately have ∃Y∈T. x∈Y ∧ Y⊆U by auto
  }
  then have U∈T using open˙neigh˙open by auto
}
ultimately have
–U ∈ Pow(G). ∀x∈U. U ∈ –⟨t, –V–t″ . V ∈ leftUniformity″⟩. t ∈ G″(x)″ = T
–U ∈ Pow(G). ∀x∈U. U ∈ –⟨t, –V–t″ . V ∈ rightUniformity″⟩. t ∈ G″(x)″ = T
  by auto
then show UniformTopology(leftUniformity,G) = T and UniformTopology(rightUniformity,G)
= T
  unfolding UniformTopology˙def by auto
qed
```

The side uniformities are called this way because of how they affect left and right translations. In the next lemma we show that left translations are uniformly continuous with respect to the left uniformity.

```
lemma (in topgroup) left˙mult˙uniformity: assumes x∈G
  shows
    LeftTranslation(G,f,x) –is uniformly continuous between″ leftUniformity –and″
leftUniformity
proof -
  let P = ProdFunction(LeftTranslation(G, f, x), LeftTranslation(G, f, x))
  from assms have L: LeftTranslation(G,f,x):G→G and leftUniformity –is a unifor-
mity on″ G
    using group0˙5˙L1 side˙uniformities(1) by auto
  moreover have ∀V ∈ leftUniformity. P-(V) ∈ leftUniformity
  proof -
    { fix V assume V ∈ leftUniformity
      then obtain U where U ∈ 𝒩0 and –⟨s,t⟩ ∈ G × G . (- s) + t ∈ U″ ⊆ V
        unfolding leftUniformity˙def by auto
      with ‹V ∈ leftUniformity› have
        as:V ⊆ G × G U ∈ 𝒩0 –⟨s,t⟩ ∈ G × G . (- s) + t ∈ U″ ⊆ V
        unfolding leftUniformity˙def by auto
      { fix z assume z:z ∈ –⟨s,t⟩ ∈ G × G . (- s) + t ∈ U″
        then obtain s t where st:z=⟨s,t⟩ s∈G t∈G by auto
        from st(1) z have st2: (- s) + t ∈ U by auto
        from assms st have
          P(z) = ⟨LeftTranslation(G, f, x)(s), LeftTranslation(G, f, x)(t)⟩
          using prodFunctionApp group0˙5˙L1(2) by blast
        with assms st(2,3) have P(z) = ⟨x+s,x+t⟩ using group0˙5˙L2(2)
          by auto
        moreover
        from ‹x∈G› ‹s∈G› ‹t∈G› have (- (x+s)) + (x+t) = (-s)+t
          using cancel˙middle˙add(3) by simp
        with st2 have (- (x+s)) + (x+t) ∈ U by auto
        ultimately have P(z) ∈ –⟨s,t⟩ ∈ G × G . (- s) + t ∈ U″
          using assms st(2,3) group˙op˙closed by auto
        with as(3) have P(z) ∈ V by force
```

```
      with L z have z ∈ P-(V) using prodFunction func1˙1˙L5A vimage˙iff
        by blast
    }
    with as(2) have ∃U∈𝒩0. –⟨s,t⟩ ∈ G × G . (- s) + t ∈ U″ ⊆ P-(V)
      by blast
    with ‹LeftTranslation(G,f,x):G→G› ‹V ⊆ G × G› have P-(V) ∈ leftUniformity
      unfolding leftUniformity˙def using prodFunction func1˙1˙L6A by blast
  } thus thesis by simp
 qed
 ultimately show thesis using IsUniformlyCont˙def by auto
qed
```

Right translations are uniformly continuous with respect to the right uniformity.

```
lemma (in topgroup) right˙mult˙uniformity: assumes x∈G
 shows
    RightTranslation(G,f,x) –is uniformly continuous between″ rightUniformity
–and″ rightUniformity
proof -
 let P = ProdFunction(RightTranslation(G, f, x), RightTranslation(G, f, x))
 from assms have R: RightTranslation(G,f,x):G→G and rightUniformity –is a uni-
formity on″ G
   using group0˙5˙L1 side˙uniformities(2) by auto
 moreover have ∀V ∈ rightUniformity. P-(V) ∈ rightUniformity
 proof -
   { fix V assume V ∈ rightUniformity
    then obtain U where U ∈ 𝒩0 and –⟨s,t⟩ ∈ G × G . s + (- t) ∈ U″ ⊆ V
      unfolding rightUniformity˙def by auto
    with ‹V ∈ rightUniformity› have
      as:V ⊆ G × G U ∈ 𝒩0 –⟨s,t⟩ ∈ G × G . s + (- t) ∈ U″ ⊆ V
      unfolding rightUniformity˙def by auto
    { fix z assume z:z ∈ –⟨s,t⟩ ∈ G × G . s + (- t) ∈ U″
      then obtain s t where st:z=⟨s,t⟩ s∈G t∈G by auto
      from st(1) z have st2: s + (-t) ∈ U by auto
     from assms st have P(z) = ⟨RightTranslation(G, f, x)(s), RightTranslation(G,
f, x)(t)⟩
        using prodFunctionApp group0˙5˙L1(1) by blast
      with assms st(2,3) have P(z) = ⟨s+x,t+x⟩ using group0˙5˙L2(1)
        by auto
      moreover
      from ‹x∈G› ‹s∈G› ‹t∈G› have (s+x) + (-(t+x)) =s+ (-t)
        using cancel˙middle˙add(4) by simp
      with st2 have (s+x) + (-(t+x)) ∈ U by auto
      ultimately have P(z) ∈ –⟨s,t⟩ ∈ G × G . s + (- t) ∈ U″
        using assms st(2,3) group˙op˙closed by auto
      with as(3) have P(z) ∈ V by force
      with R z have z ∈ P-(V) using prodFunction func1˙1˙L5A vimage˙iff
        by blast
    }
```

```
    with as(2) have ∃U∈𝒩0. −⟨s,t⟩ ∈ G × G . s + (- t) ∈ U″ ⊆ P-(V)
      by blast
    with ‹RightTranslation(G,f,x):G→G› ‹V ⊆ G × G› have P-(V) ∈ rightUniformity
      unfolding rightUniformity˙def using prodFunction func1˙1˙L6A by blast
    } thus thesis by simp
  qed
  ultimately show thesis using IsUniformlyCont˙def by auto
qed
```

The third uniformity important on topological groups is called the uniformity of Roelcke.

```
definition(in topgroup) roelckeUniformity
 where roelckeUniformity ≡ −V∈Pow(G×G). ∃U∈𝒩0. −⟨s,t⟩∈G×G. t ∈(U+s)+U″⊆
V″
```

The Roelcke uniformity is indeed a uniformity on the group.

```
lemma (in topgroup) roelcke˙uniformity:
  shows roelckeUniformity −is a uniformity on″ G
proof -
  let Φ = roelckeUniformity
  have ∀U ∈ Φ. id(G)⊆U ∧ (∃V∈Φ. V O V ⊆ U) ∧ converse(U) ∈ Φ
  proof
    fix U assume U:U ∈ roelckeUniformity
    then obtain V where V:−⟨s,t⟩∈G×G. t ∈(V+s)+V″ ⊆ U V∈ 𝒩0 U:Pow(G×G)
      unfolding roelckeUniformity˙def by auto
    from V(2) have VG:V⊆G by auto
    have id(G) ⊆ U
    proof -
      from V(2) have 0∈int(V) by auto
      then have V0:0∈V using Top˙2˙L1 by auto
      { fix x assume x:x∈G
        with ‹V∈𝒩0› have x ∈ V+x using elem˙in˙int˙rtrans(1) Top˙2˙L1 by blast
        with ‹V⊆G› ‹x∈G› ‹0∈V› have x+0 : (V+x)+V
          using lrtrans˙in˙group˙add(2) interval˙add(4) by auto
        with ‹x∈G›  have x: (V+x)+V using group0˙2˙L2 by auto
        with ‹x∈G› have ⟨x,x⟩:−⟨s,t⟩∈G×G. t ∈(V+s)+V″ by auto
        with V(1) have ⟨x,x⟩∈U by auto
      } thus id(G) ⊆ U by auto
    qed
    moreover have converse(U) ∈ Φ
    proof -
      { fix l assume l ∈ −⟨s,t⟩∈G×G. t ∈((-V)+s)+(-V)″
        then obtain s t where st:s∈G t∈G t ∈((-V)+s)+(-V) l=⟨s,t⟩
          by force
        from ‹V⊆G› have smG:(-V) ⊆ G using ginv˙image˙add(1) by simp
        with ‹s∈G› have VxG:(-V)+s ⊆ G using lrtrans˙in˙group˙add(2) by simp
        from ‹V⊆G› ‹t∈G› have VsG:V+t ⊆ G using lrtrans˙in˙group˙add(2) by simp
        from st(3) VxG smG obtain x y where xy:t = x+y x ∈ (-V)+s y∈(-V)
          using elements˙in˙set˙sum by blast
```

from xy(2) smG st(1) obtain z where z:x = z+s z∈(-V) using elements˙in˙rtrans

by blast
with ‹y∈(-V)› ‹(-V)⊆G› ‹s∈G› ‹t = x+y›
have ts:(-z)+t+(-y) = s using cancel˙middle˙add(5) by blast
{
fix u assume u∈(-V)
with ‹V⊆G› have (-u) ∈ V using ginv˙image˙el˙add by simp
} hence R:∀u∈(-V).(-u) ∈ V by simp
with z(2) xy(3) have zy:(-z)∈V (-y)∈V by auto
from zy(1) VG st(2) have (-z)+t : V+t using lrtrans˙image(2) by auto
with zy(2) VG VsG have (-z)+t+(-y) : (V+t)+V
using interval˙add(4) by auto
with ts have s:(V+t)+V by auto
with st(1,2) have ⟨s,t⟩ ∈ converse(−⟨s,t⟩∈G×G. t ∈(V+s)+V″)
using converse˙iff by auto
with V(1) have ⟨s,t⟩ ∈ converse(U) by auto
with st(4) have l ∈ converse(U) by auto
} then have −⟨s,t⟩∈G×G. t ∈((-V)+s)+(-V)″ ⊆ converse(U) by auto
moreover from V(2) have (-V): $\mathcal{N}_0$ using neg˙neigh˙neigh by auto
ultimately have ∃V∈ $\mathcal{N}_0$. −⟨s,t⟩∈G×G. t ∈(V+s)+V″ ⊆ converse(U) by auto
moreover
from V(3) have converse(U) ⊆ G×G unfolding converse˙def by auto
ultimately show converse(U) ∈ roelckeUniformity unfolding roelckeUniformity˙def by auto
qed
moreover have ∃Z∈Φ. Z O Z ⊆ U
proof -
from V(2) obtain W where W:W∈$\mathcal{N}_0$ W+W ⊆ V using exists˙procls˙zerohood by blast
then have WG:W⊆G by auto
moreover
{ fix k assume as:k:−⟨s,t⟩∈G×G. t ∈(W+s)+W″ O −⟨s,t⟩∈G×G. t ∈(W+s)+W″
then obtain x_1 x_2 x_3 where
x:x_1∈G x_2∈G x_3∈G x_2 ∈ (W+x_1)+W x_3 ∈ (W+x_2)+W k=⟨x_1,x_3⟩
unfolding comp˙def by auto
from ‹x_1∈G› have VsG:W+x_1 ⊆ G and Vx1G:V+x_1 ⊆ G
using lrtrans˙in˙group˙add(2) by auto
from x(4) VsG WG obtain x y where xy:x_2 = x+y x ∈ W+x_1 y∈W
using elements˙in˙set˙sum by blast
from xy(2) WG x(1) obtain z where z:x = z+x_1 z∈W using elements˙in˙rtrans

by blast
from z(2) xy(3) WG have yzG:y∈G z∈G by auto
from x(2) have VsG:W+x_2 ⊆ G using lrtrans˙in˙group˙add by simp
from x(5) VsG WG obtain x' y' where xy2:x_3 = x'+y' x'∈ W+x_2 y'∈W
using elements˙in˙set˙sum by blast
from xy2(2) WG x(2) obtain z' where z2:x' = z'+x_2 z'∈W using elements˙in˙rtrans

```
      by blast
    from z2(2) xy2(3) WG have yzG2:y'∈G z'∈G by auto
    from xy(1) z(1) xy2(1) z2(1) have x3 = (z'+(z+x1+y))+y' by auto
    with yzG yzG2 x(1) have x3:x3 = ((z'+z)+x1)+(y+y')
      using group˙oper˙assoc group˙op˙closed by simp
    from xy(3) z(2) xy2(3) z2(2) WG have z'+z ∈  W+W y+y' ∈ W+W
      using interval˙add(4) by auto
    with W(2) have yzV:z'+z ∈ V y+y' ∈ V by auto
   from yzV(1) VG x(1) have (z'+z)+x1 ∈ V+x1 using lrtrans˙image(2) by auto
    with yzV(2) Vx1G VG have ((z'+z)+x1)+(y+y') ∈ (V+x1)+V
      using interval˙add(4) by auto
    with x3 have x3 ∈ (V+x1)+V by auto
    with x(1,3,6) have k:−⟨s,t⟩∈G×G. t ∈(V+s)+V″ by auto
  }
 with V(1) have −⟨s,t⟩∈G×G. t ∈(W+s)+W″ O −⟨s,t⟩∈G×G. t ∈(W+s)+W″⊆U

    by auto
  moreover from W(1) have −⟨s,t⟩∈G×G. t ∈(W+s)+W″∈roelckeUniformity
    unfolding roelckeUniformity˙def by auto
  ultimately show ∃Z∈roelckeUniformity. Z O Z ⊆ U by auto
 qed
 ultimately show id(G)⊆U ∧ (∃V∈Φ. V O V ⊆ U) ∧ converse(U) ∈ Φ by simp
qed
moreover
have roelckeUniformity −is a filter on″ (G × G)
proof -
 {
  assume 0 ∈ roelckeUniformity
  then obtain W where U:W∈𝒩0 −⟨s,t⟩∈G×G. t ∈(W+s)+W″⊆0
    unfolding roelckeUniformity˙def by auto
  have ⟨0,0⟩:G×G using zero˙in˙tgroup by auto
  moreover have 0 = 0+0+0 using group0˙2˙L2 zero˙in˙tgroup by auto
  moreover
  from U(1) have 0∈int(W) by auto
  then have 0∈W using Top˙2˙L1 by auto
  with ‹W∈𝒩0› have 0+0+0 ∈ (W+0)+W
    using  group0˙2˙L2 group˙op˙closed trans˙neutral˙image interval˙add˙zero
    by auto
  ultimately have ⟨0,0⟩∈−⟨s,t⟩∈G×G. t ∈(W+s)+W″ by auto
  with U(2) have False by blast
 }
 moreover
 {
  fix x xa assume as:x ∈ roelckeUniformity xa∈x
  have roelckeUniformity ⊆ Pow(G×G) unfolding roelckeUniformity˙def by auto
  with as have xa ∈ G×G by auto
 }
 moreover
 {
```

have G×G∈Pow(G×G) by auto
moreover
have −⟨s,t⟩:G×G. t ∈(G+s)+G″ ⊆ G×G by auto
moreover note zneigh˙not˙empty
ultimately have G×G∈roelckeUniformity unfolding roelckeUniformity˙def by auto
}
moreover
{
fix A B assume as:A∈roelckeUniformity B∈roelckeUniformity
from as(1) obtain AU where
AU:AU∈ $\mathcal{N}_0$ −⟨s,t⟩∈G×G. t ∈(AU+s)+AU″⊆ A A∈Pow(G×G)
unfolding roelckeUniformity˙def by auto
from as(2) obtain BU where
BU:BU∈ $\mathcal{N}_0$ −⟨s,t⟩∈G×G. t ∈(BU+s)+BU″⊆ B B∈Pow(G×G)
unfolding roelckeUniformity˙def by auto
from AU(1) BU(1) have **0** ∈ int(AU)∩int(BU) by auto
moreover have op:int(AU)∩int(BU)∈T using Top˙2˙L2 topSpaceAssum unfolding IsATopology˙def
by auto
moreover
have int(AU)∩int(BU) ⊆ AU∩BU using Top˙2˙L1 by auto
with op have int(AU)∩int(BU)⊆int(AU∩BU) using Top˙2˙L5
by auto
moreover note AU(1) BU(1)
ultimately have interNeigh:AU∩BU ∈ $\mathcal{N}_0$ unfolding zerohoods˙def by auto
moreover
{
fix z assume z ∈ −⟨s,t⟩∈G×G. t ∈((AU∩BU)+s)+(AU∩BU)″
then obtain s t where
z:z=⟨s,t⟩ s∈G t∈G t ∈ ((AU∩BU)+s)+(AU∩BU)
by force
from ‹AU∩BU ∈ $\mathcal{N}_0$› ‹s∈G› have AU∩BU ⊆ G and (AU∩BU)+s ⊆G
using lrtrans˙in˙group˙add(2) by auto
with z(4) obtain x y where t:t=x+y x∈(AU∩BU)+s y∈AU∩BU
using elements˙in˙set˙sum by blast
from t(2) z(2) interNeigh obtain q where x:x=q+s q ∈ AU∩BU using lrtrans˙image(2)
by auto
with AU(1) BU(1) z(2) have x ∈ AU+s x ∈ BU+s using lrtrans˙image(2) by auto
with ‹y ∈ AU∩BU› ‹AU∈ $\mathcal{N}_0$› ‹BU∈ $\mathcal{N}_0$› ‹s ∈ G› ‹t=x+y› have
t ∈ (AU+s)+AU and t ∈ (BU+s)+BU
using lrtrans˙in˙group˙add(2) elements˙in˙set˙sum˙inv by auto
with z(1,2,3) have
z ∈ −⟨s,t⟩∈G×G. t ∈(AU+s)+AU″ and z ∈−⟨s,t⟩∈G×G. t ∈(BU+s)+BU″
by auto
}
then have

−⟨s,t⟩∈G×G. t ∈((AU∩BU)+s)+(AU∩BU)″ ⊆
−⟨s,t⟩∈G×G. t ∈(AU+s)+AU″∩−⟨s,t⟩∈G×G. t ∈(BU+s)+BU″
by auto
with AU(2) BU(2) have −⟨s,t⟩∈G×G. t ∈((AU∩BU)+s)+(AU∩BU)″⊆A∩B by blast
ultimately have A∩B ∈ roelckeUniformity using AU(3) BU(3) unfolding roelckeUniformity˙def
by blast
}
moreover
{
fix B C assume as:B∈roelckeUniformity C⊆(G × G) B ⊆ C
from as(1) obtain BU where BU:BU∈$\mathcal{N}_0$ −⟨s,t⟩∈G×G. t ∈(BU+s)+BU″⊆B
unfolding roelckeUniformity˙def by blast
from as(3) BU(2) have −⟨s,t⟩∈G×G. t ∈(BU+s)+BU″⊆C by auto
then have C ∈ roelckeUniformity using as(2) BU(1) unfolding roelckeUniformity˙def
by auto
}
ultimately show thesis unfolding IsFilter˙def by auto
qed
ultimately show thesis using IsUniformity˙def by auto
qed

The topology given by the roelcke uniformity is the original topology

lemma (in topgroup) top˙generated˙roelcke˙uniformity:
shows UniformTopology(roelckeUniformity,G) = T
proof -
let M = −⟨t, −V −t″ . V ∈ roelckeUniformity″⟩ . t ∈ G″
have fun:M:G→Pow(Pow(G)) using IsNeighSystem˙def neigh˙from˙uniformity roelcke˙uniformity
by auto
{
fix U assume as:U ∈ −U ∈ Pow(G). ∀x∈U. U ∈ Mx″
{
fix x assume x:x∈U
with as have xg:x∈G by auto
from x as have U ∈ −⟨t, −V −t″ . V ∈ roelckeUniformity″⟩ . t ∈ G″(x) by auto
with fun ‹x∈G› have U ∈ −V −x″ . V ∈ roelckeUniformity″ using ZF˙fun˙from˙tot˙val

by simp
then obtain V where V:U=V−x″ V ∈ roelckeUniformity by auto
from V(2) obtain W where W:W∈$\mathcal{N}_0$ −⟨s,t⟩∈G×G. t∈ (W+s)+W″⊆V
unfolding roelckeUniformity˙def by auto
from W(1) have WG:W⊆G by auto
from W(2) have A:−⟨s,t⟩:G×G. t:(W+s)+W″−x″ ⊆ V−x″ by auto
have −⟨s,t⟩ ∈ G×G. t ∈ (W+s)+W″−x″ = (W+x)+W
proof -
let A = −⟨s,t⟩:G×G. t ∈ (W+s)+W″

```
      from ‹W⊆G› ‹x∈G› have I:(W+x)+W ⊆ G
          using lrtrans˙in˙group˙add interval˙add(1) by auto
      have A–x″ = –t∈G.⟨x,t⟩ ∈ A″ by blast
      moreover have –t∈G. ⟨x,t⟩ ∈ A″ ⊆ (W+x)+W by auto
      moreover from ‹W⊆G› ‹x∈G› I have (W+x)+W ⊆ –t∈G. ⟨x,t⟩ ∈ A″ by auto
      ultimately show thesis by auto
    qed
    with A V(1) have WU:(W+x)+W ⊆ U by auto
    have int(W)+x ⊆ W+x using image˙mono Top˙2˙L1 by simp
    then have (int(W)+x)×(int(W)) ⊆ (W+x)×W using Top˙2˙L1 by auto
    then have f((int(W)+x)×(int(W))) ⊆ f((W+x)×W) using image˙mono by auto
    moreover
    from xg WG have
      ⟨int(W)+x,int(W)⟩ ∈ Pow(G)×Pow(G) and ⟨(W+x),W⟩ ∈ Pow(G)×Pow(G)

      using Top˙2˙L2 lrtrans˙in˙group˙add(2) by auto
    then have
      (int(W)+x)+(int(W)) = f((int(W)+x)×(int(W))) and
      (W+x)+W = f((W+x)×W)
      using interval˙add(2) by auto
    ultimately have (int(W)+x)+(int(W)) ⊆ (W+x)+W by auto
    with xg WG have int(W+x)+(int(W)) ⊆ (W+x)+W using rtrans˙interior
      by auto
    moreover
    {
      have int(W+x)+(int(W)) = (⋃t∈int(W+x). t+(int(W)))
        using interval˙add(3) Top˙2˙L2 by auto
      moreover have ∀t∈int(W+x). t+(int(W)) = int(t+W)
      proof -
        { fix t assume t ∈ int(W+x)
          from ‹x∈G› have (W+x) ⊆ G using lrtrans˙in˙group˙add(2) by simp
          with ‹t ∈ int(W+x)› have t∈G using Top˙2˙L2 by auto
          with ‹W⊆G› have t + int(W) = int(t+W) using ltrans˙interior by simp
        } thus thesis by simp
      qed
      ultimately have int(W+x)+(int(W)) = (⋃t∈int(W+x). int(t+W))
        by auto
    with topSpaceAssum have int(W+x)+(int(W)) ∈ T using Top˙2˙L2 union˙open
        by auto
    }
    moreover from ‹x∈G› ‹W∈𝒩0› have x ∈ int(W+x)+(int(W))
      using elem˙in˙int˙rtrans(2) by simp
    moreover note WU
    ultimately have ∃Y∈T. x∈Y ∧ Y⊆U by auto
  }
  then have U∈T using open˙neigh˙open by auto
}
then have –U ∈ Pow(G). ∀x∈U. U ∈ –⟨t, –V  –t″ . V ∈ roelckeUniformity″⟩ . t
∈ G″(x)″ ⊆ T
```

```
  by auto
moreover
{
  fix U assume op:U∈T
  {
    fix x assume x:x∈U
    with op have xg:x∈G by auto
    have (-x)+U ∈ 𝒩0 using open˙trans˙neigh op x by auto
   then obtain W where W:W∈𝒩0 W + W ⊆(-x)+U using exists˙procls˙zerohood

      by blast
    let V = x+(W+(-x)) ∩ W
    from ‹W ∈ 𝒩0› ‹x∈G› have  xWx:x+(W+(-x)) :𝒩0 using lrtrans˙neigh
      by simp
    from W(1) have WG:W⊆G by auto
    from xWx W(1) have 0∈int(x+(W+(-x)))∩int(W) by auto
    have int:int(x+(W+(-x)))∩int(W)∈T
      using Top˙2˙L2 topSpaceAssum  unfolding IsATopology˙def by auto
    have int(x+(W+(-x)))∩int(W) ⊆ (x+(W+(-x)))∩W using Top˙2˙L1
      by auto
    with int have int(x+(W+(-x)))∩int(W)⊆int((x+(W+(-x)))∩W)
      using Top˙2˙L5 by auto
    moreover note xWx W(1)
    ultimately have V˙NEIG:V ∈ 𝒩0 unfolding zerohoods˙def by auto
    {
      fix z assume z:z ∈ (V+x)+V
      from W(1) have VG:V ⊆ G by auto
      with ‹x∈G› have VxG:V+x ⊆ G using lrtrans˙in˙group˙add(2) by simp
      from z VG VxG W(1) obtain a1 b1 where ab:z=a1+b1 a1 ∈ V+x b1 ∈ V
        using elements˙in˙set˙sum by blast
     from ab(2) xg VG obtain c1 where c:a1=c1+x c1 ∈ V using elements˙in˙rtrans

        by blast
      from ab(3) c(2) have bc:b1∈W c1 ∈ x+(W+(-x)) by auto
      from ‹x∈G› have x+(W+(-x)) = –x+y. y∈(W+(-x))″
        using neg˙in˙tgroup lrtrans˙in˙group˙add lrtrans˙image by auto
      with ‹c1 ∈ x+(W+(-x))› obtain d where d:c1=x+d d ∈ W+(-x)
        by auto
      from ‹x∈G› ‹W∈𝒩0› ‹d ∈ W+(-x)› obtain e where e:d=e+(-x) e∈W
        using neg˙in˙tgroup lrtrans˙in˙group lrtrans˙image(2) by auto
      from e(2) WG have eG:e∈G by auto
      from ‹e∈W› ‹W⊆G› ‹b1∈W› have e∈G b1∈G by auto
      from ‹z = a1+b1› ‹a1 = c1+x›  ‹c1 = x+d› ‹d = e+(-x)›
      have z = x + (e+(-x)) + x + b1 by simp
      with ‹x∈G› ‹e∈G› have z=(x+e)+b1 using cancel˙middle(4) by simp
     with ‹x∈G› ‹e∈G› ‹b1∈G› have z = x+(e+b1) using group˙oper˙assoc by simp
   moreover from e(2) ab(3) WG have e+b1 ∈ W+W using elements˙in˙set˙sum˙inv

        by auto
```

moreover note xg WG
ultimately have z∈x+(W+W) using elements˙in˙ltrans˙inv interval˙add(1)
by auto
moreover
from ‹W⊆G› ‹U∈T› have W + W ⊆ G and U⊆G using interval˙add(1) by auto
with ‹W + W ⊆(-x)+U› ‹x∈G› have x+(W+W) ⊆ U using trans˙subset
by simp
ultimately have z∈U by auto
}
then have sub:(V+x)+V ⊆ U by auto
moreover from V˙NEIG have unif:−⟨s,t⟩ ∈ G×G. t : (V+s)+V″ ∈ roelckeUniformity
unfolding roelckeUniformity˙def by auto
moreover from xg have
∀q. q∈−⟨s,t⟩ ∈ G×G. t : (V+s)+V″−x″ ⟷ q∈((V+x)+V)∩G
by auto
then have −⟨s,t⟩ ∈ G×G. t ∈ (V+s)+V″−x″ = ((V+x)+V)∩G
by auto
ultimately have basic:−⟨s,t⟩ ∈ G×G. t : (V+s)+V″−x″ ⊆ U using op
by auto
have add:(−x″×U)−x″ =U by auto
from basic add have (−⟨s,t⟩ ∈ G×G. t ∈ (V+s)+V″∪(−x″×U))−x″ = U
by auto
moreover have R:∀B∈roelckeUniformity.(∀ C∈Pow(G × G). B ⊆ C ⟶ C ∈ roelckeUniformity)
using roelcke˙uniformity unfolding IsUniformity˙def IsFilter˙def by auto
moreover from op xg have GG:(−⟨s,t⟩ ∈ G×G. t ∈ (V+s)+V″∪(−x″×U)):Pow(G×G) by auto
moreover have
−⟨s,t⟩ ∈ G×G. t∈(V+s)+V″ ⊆ (−⟨s,t⟩ ∈ G×G. t∈(V+s)+V″∪(−x″×U))
by auto
moreover from R unif GG have
(−⟨s,t⟩ ∈ G×G. t ∈ (V+s)+V″∪(−x″×U)) ∈ roelckeUniformity by auto
ultimately have ∃V∈roelckeUniformity. V−x″ = U by auto
then have U ∈ −V −x″ . V ∈ roelckeUniformity″ by auto
with xg fun have U ∈ −⟨t, −V −t″ . V ∈ roelckeUniformity″⟩ . t ∈ G″ x
using apply˙equality by auto
} hence ∀x∈U. U ∈ −⟨t, −V −t″ . V ∈ roelckeUniformity″⟩ . t ∈ G″ x by auto
with op have U:−U ∈ Pow(G). ∀x∈U. U ∈ −⟨t, −V −t″ . V ∈ roelckeUniformity″⟩ . t ∈ G″(x)″
by auto
} then have T ⊆ −U ∈ Pow(G). ∀x∈U. U ∈ −⟨t, −V −t″ . V ∈ roelckeUniformity″⟩ . t ∈ G″(x)″
by auto
ultimately have
−U ∈ Pow(G). ∀x∈U. U ∈ −⟨t, −V −t″ . V ∈ roelckeUniformity″⟩ . t ∈ G″(x)″ = T by auto
then show thesis unfolding UniformTopology˙def by auto

qed

The inverse map is uniformly continuous in the Roelcke uniformity

theorem (in topgroup) inv˙uniform˙roelcke:
shows
GroupInv(G,f) –is uniformly continuous between″ roelckeUniformity –and″ roelckeUniformity
proof -
let P = ProdFunction(GroupInv(G,f), GroupInv(G,f))
have L: GroupInv(G,f):G→G and R:roelckeUniformity –is a uniformity on″ G
using groupAssum group0˙2˙T2 roelcke˙uniformity by auto
have ∀ V ∈ roelckeUniformity. P-(V) ∈ roelckeUniformity
proof
fix V assume v:V∈ roelckeUniformity
then obtain U where U ∈ $\mathcal{N}_0$ and –⟨s,t⟩ ∈ G × G . t ∈ U + s + U″ ⊆ V
unfolding roelckeUniformity˙def by auto
with ‹V ∈ roelckeUniformity› have
as:V ⊆ G × G U ∈ $\mathcal{N}_0$ –⟨s,t⟩ ∈ G × G . t ∈ U + s + U″ ⊆ V
unfolding roelckeUniformity˙def by auto
from as(2) obtain W where w:W ∈ $\mathcal{N}_0$ W ⊆ U (-W) = W using exists˙sym˙zerohood by blast
from w(1) have wg:W⊆G by auto
{
fix z assume z:z ∈ –⟨s,t⟩ ∈ G × G . t ∈ W + s + W″
then obtain s t where st:z=⟨s,t⟩ s∈G t∈G by auto
from st(1) z have st2: t ∈ W + s + W by auto
with ‹W ∈ $\mathcal{N}_0$› st(2) obtain u v where uv:t=u+v u∈W+s v∈W
using interval˙add(4) lrtrans˙in˙group˙add(2) by auto
from ‹W⊆G› ‹s∈G› ‹u∈W+s› obtain q where q:q∈W u=q+s using elements˙in˙rtrans
by blast
from w(2) as(2) q st(2) have u∈U+s using lrtrans˙image(2) by auto
with w(2) uv(1,3) as(2) st(2) have t∈U + s + U using interval˙add(4)
lrtrans˙in˙group˙add(2) by auto
with st have z ∈ –⟨s,t⟩ ∈ G × G . t ∈ U + s + U″ by auto
}
then have
sub:–⟨s,t⟩ ∈ G × G . t ∈ W + s + W″ ⊆ –⟨s,t⟩ ∈ G × G . t ∈ U + s + U″
by auto
{
fix z assume z:z ∈ –⟨s,t⟩ ∈ G × G . t ∈ W + s + W″
then obtain s t where st:z=⟨s,t⟩ s∈G t∈G by auto
from st(1) z have st2: t ∈ W + s + W by auto
with ‹W ∈ $\mathcal{N}_0$› obtain u v where uv:t=u+v u∈W+s v∈W
using interval˙add(4) lrtrans˙in˙group˙add(2) st(2) by auto
from ‹W⊆G› ‹s∈G› ‹u∈W+s› obtain q where q:q∈W u=q+s using elements˙in˙rtrans
by blast
from ‹W⊆G› ‹q∈W› ‹v∈W› have q∈G v∈G by auto

```
      with ‹q∈G› ‹v∈G› ‹u=q+s› st(2) uv(1) q(2) have t=q+(s+v)
        using group˙op˙closed˙add group˙oper˙assoc by auto
      with st(2) ‹q∈G› ‹v∈G› have minust:(-t) = (-v)+(-s)+(-q)
        using group˙inv˙of˙two group˙op˙closed group˙inv˙of˙two by auto
      from q(1) wg have (-q)∈-W using ginv˙image˙add(2) by auto
      with w(3) have minusq:(-q)∈W by auto
      from uv(3) wg have (-v)∈-W using ginv˙image˙add(2) by auto
      with w(3) have minusv:(-v)∈W by auto
      with st(2) wg have (-v)+(-s) ∈ W+(-s)
        using lrtrans˙image(2) inverse˙in˙group by auto
      with minust minusq st(2) wg have (-t)∈(W+(-s))+W
        using interval˙add(4) inverse˙in˙group lrtrans˙in˙group˙add(2) by auto
      moreover
      from st groupAssum have P(z) = ⟨GroupInv(G,f)(s), GroupInv(G,f)(t)⟩
        using prodFunctionApp group0˙2˙T2 by blast
      with st(2,3) have P(z) = ⟨-s,-t⟩ by auto
      ultimately have P(z) ∈ –⟨s,t⟩ ∈ G × G . t ∈ W + s + W″
        using st(2,3) inverse˙in˙group by auto
      with sub have P(z) ∈ –⟨s,t⟩ ∈ G × G . t ∈ U + s + U″ by force
      with as(3) have P(z) ∈ V by force
      with z L have z ∈ P-(V) using prodFunction func1˙1˙L5A vimage˙iff
        by blast
    }
    with w(1) have ∃U∈𝒩0. –⟨s,t⟩ ∈ G × G . t ∈ U + s + U″ ⊆ P-(V)
      by blast
    with L show P-(V) ∈ roelckeUniformity
      unfolding roelckeUniformity˙def using prodFunction func1˙1˙L6A by blast
  qed
  with L R show thesis using IsUniformlyCont˙def by auto
qed

end
```

75 Topological groups 2

```
theory TopologicalGroup˙ZF˙2 imports Topology˙ZF˙8 TopologicalGroup˙ZF Group˙ZF˙2
begin
```

This theory deals with quotient topological groups.

75.1 Quotients of topological groups

The quotient topology given by the quotient group equivalent relation, has an open quotient map.

```
theorem(in topgroup) quotient˙map˙topgroup˙open:
  assumes IsAsubgroup(H,f) A∈T
  defines r ≡ QuotientGroupRel(G,f,H)
  shows –⟨b,r–b″⟩. b∈⋃T″A∈(T–quotient by″r)
```

```
proof-
 have eqT:equiv(⋃T,r) and eqG:equiv(G,r) using group0.Group˙ZF˙2˙4˙L3 assms(1)
unfolding r˙def IsAnormalSubgroup˙def
   using group0˙valid˙in˙tgroup by auto
 have subA:A⊆G using assms(2) by auto
 have subH:H⊆G using group0.group0˙3˙L2[OF group0˙valid˙in˙tgroup assms(1)].
 have A1:–⟨b,r–b″⟩. b∈⋃T″-(–⟨b,r–b″⟩. b∈⋃T″A)=H+A
 proof
   {
     fix t assume t∈–⟨b,r–b″⟩. b∈⋃T″-(–⟨b,r–b″⟩. b∈⋃T″A)
   then have ∃m∈(–⟨b,r–b″⟩. b∈⋃T″A). ⟨t,m⟩∈–⟨b,r–b″⟩. b∈⋃T″ using vimage˙iff
by auto
      then obtain m where m∈(–⟨b,r–b″⟩. b∈⋃T″A)⟨t,m⟩∈–⟨b,r–b″⟩. b∈⋃T″ by
auto
     then obtain b where b∈A⟨b,m⟩∈–⟨b,r–b″⟩. b∈⋃T″t∈G and rel:r–t″=m using
image˙iff by auto
     then have r–b″=m by auto
     then have r–t″=r–b″ using rel by auto
     with ‹b∈A›subA have ⟨t,b⟩∈r using eq˙equiv˙class[OF ˙ eqT] by auto
      then have f⟨t,GroupInv(G,f)b⟩∈H unfolding r˙def QuotientGroupRel˙def by
auto
     then obtain h where h∈H and prd:f⟨t,GroupInv(G,f)b⟩=h by auto
     then have h∈G using subH by auto
     have b∈G using ‹b∈A›‹A∈T› by auto
     then have (-b)∈G using neg˙in˙tgroup by auto
     from prd have h=t+(-b) by simp
     with ‹t∈G› ‹b∈G› have t = h+b using inv˙cancel˙two˙add(1) by simp
      then have ⟨⟨h,b⟩,t⟩∈f using apply˙Pair[OF topgroup˙f˙binop] ‹h∈G›‹b∈G› by
auto
     moreover from ‹h∈H›‹b∈A› have ⟨h,b⟩∈H×A by auto
     ultimately have t∈f(H×A) using image˙iff by auto
     with subA subH have t∈H+A using interval˙add(2) by auto
   }
   then show (–⟨b,r–b″⟩. b∈⋃T″-(–⟨b,r–b″⟩. b∈⋃T″A))⊆H+A by force
   {
     fix t assume t∈H+A
     with subA subH have t ∈ f(H×A) using interval˙add(2) by auto
     then obtain ha where ha∈H×A ⟨ha,t⟩∈f using image˙iff by auto
     then obtain h aa where ha=⟨h,aa⟩h∈Haa∈A by auto
     then have h∈Gaa∈G using subH subA by auto
     from ‹⟨ha,t⟩∈f› have t∈G using topgroup˙f˙binop unfolding Pi˙def by auto
    from ‹ha=⟨h,aa⟩› ‹⟨ha,t⟩∈f› have t=h+aa using apply˙equality topgroup˙f˙binop

      by auto
     with ‹h∈G› ‹aa∈G› have t+(-aa) = h using inv˙cancel˙two˙add(2) by simp
      with ‹h∈H›‹t∈G›‹aa∈G› have ⟨t,aa⟩∈r unfolding r˙def QuotientGroupRel˙def
by auto
     then have r–t″=r–aa″ using eqT equiv˙class˙eq by auto
     with ‹aa∈G› have ⟨aa,r–t″⟩∈–⟨b,r–b″⟩. b∈⋃T″ by auto
```

```
      with ‹aa∈A› have A1:r–t″∈(–⟨b,r–b″⟩. b∈⋃T″A) using image˙iff by auto
      from ‹t∈G› have ⟨t,r–t″⟩∈–⟨b,r–b″⟩. b∈⋃T″ by auto
      with A1 have t∈–⟨b,r–b″⟩. b∈⋃T″-(–⟨b,r–b″⟩. b∈⋃T″A) using vimage˙iff by
auto
    }
    then show H+A⊆–⟨b,r–b″⟩. b∈⋃T″-(–⟨b,r–b″⟩. b∈⋃T″A) by auto
  qed
  have H+A=(⋃x∈H. x + A) using interval˙add(3) subH subA by auto moreover
  have ∀x∈H. x + A∈T using open˙tr˙open(1) assms(2) subH by blast
  then have –x + A. x∈H″⊆T by auto
  then have (⋃x∈H. x + A)∈T using topSpaceAssum unfolding IsATopology˙def
by auto
  ultimately have H+A∈T by auto
  with A1 have –⟨b,r–b″⟩. b∈⋃T″-(–⟨b,r–b″⟩. b∈⋃T″A)∈T by auto
  then have (–⟨b,r–b″⟩. b∈⋃T″A)∈–quotient topology in″((⋃T)//r)–by″–⟨b,r–b″⟩.
b∈⋃T″–from″T
    using QuotientTop˙def topSpaceAssum quotient˙proj˙surj using
    func1˙1˙L6(2)[OF quotient˙proj˙fun] by auto
  then show (–⟨b,r–b″⟩. b∈⋃T″A)∈(T–quotient by″r) using EquivQuo˙def[OF eqT]
by auto
qed
```

A quotient of a topological group is just a quotient group with an appropiate topology that makes product and inverse continuous.

```
theorem (in topgroup) quotient˙top˙group˙F˙cont:
  assumes IsAnormalSubgroup(G,f,H)
  defines r ≡ QuotientGroupRel(G,f,H)
  defines F ≡ QuotientGroupOp(G,f,H)
  shows IsContinuous(ProductTopology(T–quotient by″r,T–quotient by″r),T–quotient
by″r,F)
proof-
  have eqT:equiv(⋃T,r) and eqG:equiv(G,r) using group0.Group˙ZF˙2˙4˙L3 assms(1)
unfolding r˙def IsAnormalSubgroup˙def
    using group0˙valid˙in˙tgroup by auto
  have fun:–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″:G×G→(G//r)×(G//r) using prod-
uct˙equiv˙rel˙fun unfolding G˙def by auto
  have C:Congruent2(r,f) using Group˙ZF˙2˙4˙L5A[OF Ggroup assms(1)] unfolding
r˙def.
  with eqT have IsContinuous(ProductTopology(T,T),ProductTopology(T–quotient
by″r,T–quotient by″r),–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″)
    using product˙quo˙fun by auto
  have tprod:topology0(ProductTopology(T,T)) unfolding topology0˙def using Top˙1˙4˙T1(1)[OF
topSpaceAssum topSpaceAssum].
  have Hfun:–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″∈surj(⋃ProductTopology(T,T),⋃((–quotient
topology in″(((⋃T)//r)×((⋃T)//r))–by″–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″–from″(ProductTopology(T,T
using prod˙equiv˙rel˙surj
     total˙quo˙equi[OF eqT] topology0.total˙quo˙func[OF tprod prod˙equiv˙rel˙surj]
unfolding F˙def QuotientGroupOp˙def r˙def
    by auto
```

have Ffun:F:⋃((–quotient topology in″(((⋃T)//r)×((⋃T)//r))–by″–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″–from″(ProductTopology(T,T))))→⋃(T–quotient by″r)
using EquivClass˙1˙T1[OF eqG C] using total˙quo˙equi[OF eqT] topology0.total˙quo˙func[OF tprod prod˙equiv˙rel˙surj] unfolding F˙def QuotientGroupOp˙def r˙def
by auto
have cc:(F O –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″):G×G→G//r using comp˙fun[OF fun EquivClass˙1˙T1[OF eqG C]]
unfolding F˙def QuotientGroupOp˙def r˙def by auto
then have (F O –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″):⋃(ProductTopology(T,T))→⋃(T–quotient by″r) using Top˙1˙4˙T1(3)[OF topSpaceAssum topSpaceAssum]
total˙quo˙equi[OF eqT] by auto
then have two:two˙top˙spaces0(ProductTopology(T,T),T–quotient by″r,(F O –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″)) unfolding two˙top˙spaces0˙def
using Top˙1˙4˙T1(1)[OF topSpaceAssum topSpaceAssum] equiv˙quo˙is˙top[OF eqT] by auto
have IsContinuous(ProductTopology(T,T),T,f) using fcon prodtop˙def by auto moreover
have IsContinuous(T,T–quotient by″r,–⟨b,r–b″⟩. b∈⋃T″) using quotient˙func˙cont[OF quotient˙proj˙surj]
unfolding EquivQuo˙def[OF eqT] by auto
ultimately have cont:IsContinuous(ProductTopology(T,T),T–quotient by″r,–⟨b,r–b″⟩. b∈⋃T″ O f)
using comp˙cont by auto
{
fix A assume A:A∈G×G
then obtain g1 g2 where A˙def:A=⟨g1,g2⟩ g1∈Gg2∈G by auto
then have fA=g1+g2 and p:g1+g2∈⋃T unfolding grop˙def using
apply˙type[OF topgroup˙f˙binop] by auto
then have –⟨b,r–b″⟩. b∈⋃T″(fA)=–⟨b,r–b″⟩. b∈⋃T″(g1+g2) by auto
with p have –⟨b,r–b″⟩. b∈⋃T″(fA)=r–g1+g2″ using apply˙equality[OF ˙ quotient˙proj˙fun]
by auto
then have Pr1:(–⟨b,r–b″⟩. b∈⋃T″ O f)A=r–g1+g2″ using comp˙fun˙apply[OF topgroup˙f˙binop A] by auto
from A˙def(2,3) have ⟨g1,g2⟩∈⋃T×⋃T by auto
then have ⟨⟨g1,g2⟩,⟨r–g1″,r–g2″⟩⟩∈–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″ by auto
then have –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″A=⟨r–g1″,r–g2″⟩ using A˙def(1) apply˙equality[OF ˙ product˙equiv˙rel˙fun]
by auto
then have F(–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″A)=F⟨r–g1″,r–g2″⟩ by auto
then have F(–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″A)=r(–g1+g2″) using group0.Group˙ZF˙2˙2˙L2[OF group0˙valid˙in˙tgroup eqG C
˙ A˙def(2,3)] unfolding F˙def QuotientGroupOp˙def r˙def by auto moreover
note fun ultimately have (F O –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″)A=r(–g1+g2″) using comp˙fun˙apply[OF ˙ A] by auto
then have (F O –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″)A=(–⟨b,r–b″⟩. b∈⋃T″ O f)A using Pr1 by auto
}
then have (F O –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″)=(–⟨b,r–b″⟩. b∈⋃T″ O f)

using fun˙extension[OF cc comp˙fun[OF topgroup˙f˙binop quotient˙proj˙fun]]
unfolding F˙def QuotientGroupOp˙def r˙def by auto
then have A:IsContinuous(ProductTopology(T,T),T–quotient by″r,F O –⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″) using cont by auto
have IsAsubgroup(H,f) using assms(1) unfolding IsAnormalSubgroup˙def by auto
then have ∀A∈T. –⟨b, r –b″⟩ . b ∈ ⋃T″ A ∈ (–quotient by″r) using quotient˙map˙topgroup˙open unfolding r˙def by auto
with eqT have ProductTopology(–quotient by″r,–quotient by″r)=(–quotient topology in″(((⋃T)//r)×((⋃T)//r))–by″–⟨⟨b,c⟩,⟨r–b″,r–c″⟩⟩. ⟨b,c⟩∈⋃T×⋃T″–from″(ProductTopology(T,T))) using prod˙quotient
by auto
with A show IsContinuous(ProductTopology(T–quotient by″r,T–quotient by″r),T–quotient by″r,F)
using two˙top˙spaces0.cont˙quotient˙top[OF two Hfun Ffun] topology0.total˙quo˙func[OF tprod prod˙equiv˙rel˙surj] unfolding F˙def QuotientGroupOp˙def r˙def
by auto
qed

lemma (in group0) Group˙ZF˙2˙4˙L8:
assumes IsAnormalSubgroup(G,P,H)
defines r ≡ QuotientGroupRel(G,P,H)
and F ≡ QuotientGroupOp(G,P,H)
shows GroupInv(G//r,F):G//r→G//r
using group0˙2˙T2[OF Group˙ZF˙2˙4˙T1[OF ˙ assms(1)]] groupAssum using assms(2,3)
by auto

theorem (in topgroup) quotient˙top˙group˙INV˙cont:
assumes IsAnormalSubgroup(G,f,H)
defines r ≡ QuotientGroupRel(G,f,H)
defines F ≡ QuotientGroupOp(G,f,H)
shows IsContinuous(T–quotient by″r,T–quotient by″r,GroupInv(G//r,F))
proof-
have eqT:equiv(⋃T,r) and eqG:equiv(G,r) using group0.Group˙ZF˙2˙4˙L3 assms(1) unfolding r˙def IsAnormalSubgroup˙def
using group0˙valid˙in˙tgroup by auto
have two:two˙top˙spaces0(T,T–quotient by″r,–⟨b,r–b″⟩. b∈G″) unfolding two˙top˙spaces0˙def
using topSpaceAssum equiv˙quo˙is˙top[OF eqT] quotient˙proj˙fun total˙quo˙equi[OF eqT] by auto
have IsContinuous(T,T,GroupInv(G,f)) using inv˙cont. moreover
{
fix g assume G:g∈G
then have GroupInv(G,f)g=-g using grinv˙def by auto
then have r(–GroupInv(G,f)g″)=GroupInv(G//r,F)(r–g″) using group0.Group˙ZF˙2˙4˙L7
[OF group0˙valid˙in˙tgroup assms(1) G] unfolding r˙def F˙def by auto
then have –⟨b,r–b″⟩. b∈G″(GroupInv(G,f)g)=GroupInv(G//r,F)(–⟨b,r–b″⟩. b∈G″g)
using apply˙equality[OF ˙ quotient˙proj˙fun] G neg˙in˙tgroup unfolding grinv˙def
by auto
then have (–⟨b,r–b″⟩. b∈G″O GroupInv(G,f))g=(GroupInv(G//r,F)O –⟨b,r–b″⟩. b∈G″)g

using comp˙fun˙apply[OF quotient˙proj˙fun G] comp˙fun˙apply[OF group0˙2˙T2[OF Ggroup] G] by auto
}
then have A1:–⟨b,r–b″⟩. b∈G″O GroupInv(G,f)=GroupInv(G//r,F)O –⟨b,r–b″⟩. b∈G″ using fun˙extension[
OF comp˙fun[OF quotient˙proj˙fun group0.Group˙ZF˙2˙4˙L8[OF group0˙valid˙in˙tgroup assms(1)]]
comp˙fun[OF group0˙2˙T2[OF Ggroup] quotient˙proj˙fun[of Gr]]] unfolding r˙def F˙def by auto
have IsContinuous(T,T–quotient by″r,–⟨b,r–b″⟩. b∈⋃T″) using quotient˙func˙cont[OF quotient˙proj˙surj]
unfolding EquivQuo˙def[OF eqT] by auto
ultimately have IsContinuous(T,T–quotient by″r,–⟨b,r–b″⟩. b∈⋃T″O GroupInv(G,f))
using comp˙cont by auto
with A1 have IsContinuous(T,T–quotient by″r,GroupInv(G//r,F)O –⟨b,r–b″⟩. b∈G″) by auto
then have IsContinuous(–quotient topology in″(⋃T) // r–by″–⟨b, r –b″⟩ . b ∈ ⋃T″–from″T,T–quotient by″r,GroupInv(G//r,F))
using two˙top˙spaces0.cont˙quotient˙top[OF two quotient˙proj˙surj, of GroupInv(G//r,F)r] group0.Group˙ZF˙2˙4˙L8[OF group0˙valid˙in˙tgroup assms(1)]
using total˙quo˙equi[OF eqT] unfolding r˙def F˙def by auto
then show thesis unfolding EquivQuo˙def[OF eqT].
qed

Finally we can prove that quotient groups of topological groups are topological groups.

theorem(in topgroup) quotient˙top˙group:
assumes IsAnormalSubgroup(G,f,H)
defines r ≡ QuotientGroupRel(G,f,H)
defines F ≡ QuotientGroupOp(G,f,H)
shows IsAtopologicalGroup(–quotient by″r,F)
unfolding IsAtopologicalGroup˙def using total˙quo˙equi equiv˙quo˙is˙top
Group˙ZF˙2˙4˙T1 Ggroup assms(1) quotient˙top˙group˙INV˙cont quotient˙top˙group˙F˙cont
group0.Group˙ZF˙2˙4˙L3 group0˙valid˙in˙tgroup assms(1) unfolding r˙def F˙def IsAnormalSubgroup˙def
by auto

end

76 Topological groups 3

theory TopologicalGroup˙ZF˙3 imports Topology˙ZF˙10 TopologicalGroup˙ZF˙2 TopologicalGroup˙ZF˙1
Group˙ZF˙4

begin

This theory deals with topological properties of subgroups, quotient groups

and relations between group theorical properties and topological properties.

76.1 Subgroups topologies

The closure of a subgroup is a subgroup.

```
theorem (in topgroup) closure˙subgroup:
  assumes IsAsubgroup(H,f)
  shows IsAsubgroup(cl(H),f)
proof-
 have two:two˙top˙spaces0(ProductTopology(T,T),T,f) unfolding two˙top˙spaces0˙def
using
    topSpaceAssum Top˙1˙4˙T1(1,3) topgroup˙f˙binop by auto
  from fcon have cont:IsContinuous(ProductTopology(T,T),T,f) by auto
 then have closed:∀ D. D–is closed in″T ⟶ f-D–is closed in″τ using two˙top˙spaces0.TopZF˙2˙1˙L1
    two by auto
 then have closure:∀ A∈Pow(⋃τ). f(Closure(A,τ))⊆cl(fA) using two˙top˙spaces0.Top˙ZF˙2˙1˙L2
    two by force
  have sub1:H⊆G using group0.group0˙3˙L2 group0˙valid˙in˙tgroup assms by force
  then have sub:(H)×(H)⊆⋃τ using prod˙top˙on˙G(2) by auto
  from sub1 have clHG:cl(H)⊆G using Top˙3˙L11(1) by auto
  then have clHsub1:cl(H)×cl(H)⊆G×G by auto
  have Closure(H×H,ProductTopology(T,T))=cl(H)×cl(H) using cl˙product
    topSpaceAssum group0.group0˙3˙L2 group0˙valid˙in˙tgroup assms by auto
  then have f(Closure(H×H,ProductTopology(T,T)))=f(cl(H)×cl(H)) by auto
  with closure sub have clcl:f(cl(H)×cl(H))⊆cl(f(H×H)) by force
  from assms have fun:restrict(f,H×H):H×H→H unfolding IsAsubgroup˙def using
    group0.group˙oper˙assocA unfolding group0˙def by auto
  then have restrict(f,H×H)(H×H)=f(H×H) using restrict˙image by auto
  moreover from fun have restrict(f,H×H)(H×H)⊆H using func1˙1˙L6(2) by blast
  ultimately have f(H×H)⊆H by auto
  with sub1 have f(H×H)⊆Hf(H×H)⊆GH⊆G by auto
  then have cl(f(H×H))⊆cl(H) using top˙closure˙mono by auto
  with clcl have img:f(cl(H)×cl(H))⊆cl(H) by auto
  {
    fix x y assume x∈cl(H)y∈cl(H)
    then have ⟨x,y⟩∈cl(H)×cl(H) by auto moreover
    have f(cl(H)×cl(H))=–ft. t∈cl(H)×cl(H)″ using func˙imagedef topgroup˙f˙binop
      clHsub1 by auto ultimately
    have f⟨x,y⟩∈f(cl(H)×cl(H)) by auto
    with img have f⟨x,y⟩∈cl(H) by auto
  }
  then have A1:cl(H)–is closed under″ f unfolding IsOpClosed˙def by auto
 have two:two˙top˙spaces0(T,T,GroupInv(G,f)) unfolding two˙top˙spaces0˙def using
    topSpaceAssum Ggroup group0˙2˙T2 by auto
  from inv˙cont have cont:IsContinuous(T,T,GroupInv(G,f)) by auto
  then have closed:∀ D. D–is closed in″T ⟶ GroupInv(G,f)-D–is closed in″T using
two˙top˙spaces0.TopZF˙2˙1˙L1
    two by auto
```

then have closure:∀A∈Pow(⋃T). GroupInv(G,f)(cl(A))⊆cl(GroupInv(G,f)A) using two˙top˙spaces0.Top˙ZF˙2˙1˙L2
two by force
with sub1 have Inv:GroupInv(G,f)(cl(H))⊆cl(GroupInv(G,f)H) by auto moreover
have GroupInv(H,restrict(f,H×H)):H→H using assms unfolding IsAsubgroup˙def using group0˙2˙T2 by auto then
have GroupInv(H,restrict(f,H×H))H⊆H using func1˙1˙L6(2) by auto
then have restrict(GroupInv(G,f),H)H⊆H using group0.group0˙3˙T1 assms group0˙valid˙in˙tgroup by auto
then have sss:GroupInv(G,f)H⊆H using restrict˙image by auto
then have H⊆G GroupInv(G,f)H⊆G using sub1 by auto
with sub1 sss have cl(GroupInv(G,f)H)⊆cl(H) using top˙closure˙mono by auto ultimately
have img:GroupInv(G,f)(cl(H))⊆cl(H) by auto
{
fix x assume x∈cl(H) moreover
have GroupInv(G,f)(cl(H))=–GroupInv(G,f)t. t∈cl(H)″ using func˙imagedef Ggroup group0˙2˙T2
clHG by force ultimately
have GroupInv(G,f)x∈GroupInv(G,f)(cl(H)) by auto
with img have GroupInv(G,f)x∈cl(H) by auto
}
then have A2:∀x∈cl(H). GroupInv(G,f)x∈cl(H) by auto
from assms have H≠0 using group0.group0˙3˙L5 group0˙valid˙in˙tgroup by auto moreover
have H⊆cl(H) using cl˙contains˙set sub1 by auto ultimately
have cl(H)≠0 by auto
with clHG A2 A1 show thesis using group0.group0˙3˙T3 group0˙valid˙in˙tgroup by auto
qed

The closure of a normal subgroup is normal.

theorem (in topgroup) normal˙subg:
assumes IsAnormalSubgroup(G,f,H)
shows IsAnormalSubgroup(G,f,cl(H))
proof-
have A:IsAsubgroup(cl(H),f) using closure˙subgroup assms unfolding IsAnormalSubgroup˙def by auto
have sub1:H⊆G using group0.group0˙3˙L2 group0˙valid˙in˙tgroup assms unfolding IsAnormalSubgroup˙def by auto
then have sub2:cl(H)⊆G using Top˙3˙L11(1) by auto
{
fix g assume g:g∈G
then have cl1:cl(g+H)=g+cl(H) using trans˙closure sub1 by auto
have ss:g+cl(H)⊆G unfolding ltrans˙def LeftTranslation˙def by auto
have g+H⊆G unfolding ltrans˙def LeftTranslation˙def by auto
moreover from g have (-g)∈G using neg˙in˙tgroup by auto
ultimately have cl2:cl((g+H)+(-g))=cl(g+H)+(-g) using trans˙closure2
by auto

```
  with cl1 have clcon:cl((g+H)+(-g))=(g+(cl(H)))+(-g) by auto
  {
    fix r assume r∈(g+H)+(-g)
     then obtain q where q:q∈g+H r=q+(-g) unfolding rtrans˙def RightTransla-
tion˙def
      by force
    from q(1) obtain h where h∈H q=g+h unfolding ltrans˙def LeftTranslation˙def
by auto
    with q(2) have r=(g+h)+(-g) by auto
     with ‹h∈H› ‹g∈G› ‹(-g)∈G› have r∈H using assms unfolding IsAnormalSub-
group˙def
      grinv˙def grop˙def by auto
  }
  then have (g+H)+(-g)⊆H by auto
  moreover then have (g+H)+(-g)⊆GH⊆G using sub1 by auto ultimately
  have cl((g+H)+(-g))⊆cl(H) using top˙closure˙mono by auto
  with clcon have (g+(cl(H)))+(-g)⊆cl(H) by auto moreover
  {
    fix b assume b∈–g+(d-g). d∈cl(H)″
    then obtain d where d:d∈cl(H) b=g+(d-g) by auto moreover
    then have d∈G using sub2 by auto
      then have g+d∈G using group0.group˙op˙closed[OF group0˙valid˙in˙tgroup
‹g∈G›] by auto
  from d(2) have b:b=(g+d)-g using group0.group˙oper˙assoc[OF group0˙valid˙in˙tgroup
‹g∈G› ‹d∈G›‹(-g)∈G›]
      unfolding grsub˙def grop˙def grinv˙def by blast
  have (g+d)=LeftTranslation(G,f,g)d using group0.group0˙5˙L2(2)[OF group0˙valid˙in˙tgroup]
      ‹g∈G›‹d∈G› by auto
   with ‹d∈cl(H)› have g+d∈g+cl(H) unfolding ltrans˙def using func˙imagedef[OF
group0.group0˙5˙L1(2)[
      OF group0˙valid˙in˙tgroup ‹g∈G›] sub2] by auto
  moreover from b have b=RightTranslation(G,f,-g)(g+d) using group0.group0˙5˙L2(1)[OF
group0˙valid˙in˙tgroup]
      ‹(-g)∈G›‹g+d∈G› by auto
   ultimately have b∈(g+cl(H))+(-g) unfolding rtrans˙def using func˙imagedef[OF
group0.group0˙5˙L1(1)[
      OF group0˙valid˙in˙tgroup ‹(-g)∈G›] ss] by force
  }
  ultimately have –g+(d-g). d∈cl(H)″⊆cl(H) by force
 }
 then show thesis using A group0.cont˙conj˙is˙normal[OF group0˙valid˙in˙tgroup,
of cl(H)]
  unfolding grsub˙def grinv˙def grop˙def by auto
qed
```

Every open subgroup is also closed.

```
theorem (in topgroup) open˙subgroup˙closed:
 assumes IsAsubgroup(H,f) H∈T
 shows H–is closed in″T
```

```
proof-
  from assms(1) have sub:H⊆G using group0.group0˙3˙L2 group0˙valid˙in˙tgroup by
force
  {
    fix t assume t∈G-H
    then have tnH:t∉H and tG:t∈G by auto
     from assms(1) have sub:H⊆G using group0.group0˙3˙L2 group0˙valid˙in˙tgroup
by force
   from assms(1) have nSubG:0∈H using group0.group0˙3˙L5 group0˙valid˙in˙tgroup
by auto
    from assms(2) tG have P:t+H∈T using open˙tr˙open(1) by auto
   from nSubG sub tG have tp:t∈t+H using group0˙valid˙in˙tgroup group0.neut˙trans˙elem
      by auto
    {
      fix x assume x∈(t+H)∩H
      then obtain u where x=t+u u∈H x∈H unfolding ltrans˙def LeftTranslation˙def
by auto
      then have u∈Gx∈Gt∈G using sub tG by auto
    with ‹x=t+u› have x+(-u)=t using group0.group0˙2˙L18(1) group0˙valid˙in˙tgroup
        unfolding grop˙def grinv˙def by auto
    from ‹u∈H› have (-u)∈H unfolding grinv˙def using assms(1) group0.group0˙3˙T3A
group0˙valid˙in˙tgroup
        by auto
    with ‹x∈H› have x+(-u)∈H unfolding grop˙def using assms(1) group0.group0˙3˙L6
group0˙valid˙in˙tgroup
        by auto
      with ‹x+(-u)=t› have False using tnH by auto
    }
    then have (t+H)∩H=0 by auto moreover
    have t+H⊆G unfolding ltrans˙def LeftTranslation˙def by auto ultimately
    have (t+H)⊆G-H by auto
    with tp P have ∃V∈T. t∈V ∧ V⊆G-H unfolding Bex˙def by auto
  }
  then have ∀t∈G-H. ∃V∈T. t∈V ∧ V⊆G-H by auto
  then have G-H∈T using open˙neigh˙open by auto
  then show thesis unfolding IsClosed˙def using sub by auto
qed
```

Any subgroup with non-empty interior is open.

```
theorem (in topgroup) clopen˙or˙emptyInt:
  assumes IsAsubgroup(H,f) int(H)≠0
  shows H∈T
proof-
  from assms(1) have sub:H⊆G using group0.group0˙3˙L2 group0˙valid˙in˙tgroup by
force
  {
    fix h assume h∈H
    have intsub:int(H)⊆H using Top˙2˙L1 by auto
    from assms(2) obtain u where u∈int(H) by auto
```

```
    with intsub have u∈H by auto
      then have (-u)∈H unfolding grinv˙def using assms(1) group0.group0˙3˙T3A
group0˙valid˙in˙tgroup
      by auto
    with ‹h∈H› have h-u∈H unfolding grop˙def using assms(1) group0.group0˙3˙L6
group0˙valid˙in˙tgroup
      by auto
    {
      fix t assume t∈(h-u)+(int(H))
      then obtain r where r∈int(H)t=(h-u)+r unfolding grsub˙def grinv˙def grop˙def
        ltrans˙def LeftTranslation˙def by auto
      then have r∈H using intsub by auto
    with ‹h-u∈H› have (h-u)+r∈H unfolding grop˙def using assms(1) group0.group0˙3˙L6
group0˙valid˙in˙tgroup
        by auto
      with ‹t=(h-u)+r› have t∈H by auto
    }
    then have ss:(h-u)+(int(H))⊆H by auto
    have P:(h-u)+(int(H))∈T using open˙tr˙open(1) ‹h-u∈H› Top˙2˙L2 sub by blast
    from ‹h-u∈H›‹u∈H›‹h∈H› sub have (h-u)∈G u∈Gh∈G by auto
    have int(H)⊆G using sub intsub by auto moreover
   have LeftTranslation(G,f,(h-u))∈G→G using group0.group0˙5˙L1(2) group0˙valid˙in˙tgroup
‹(h-u)∈G›
      by auto ultimately
   have LeftTranslation(G,f,(h-u))(int(H))=–LeftTranslation(G,f,(h-u))r. r∈int(H)″

      using func˙imagedef by auto moreover
   from ‹(h-u)∈G› ‹u∈G› have LeftTranslation(G,f,(h-u))u=(h-u)+u using group0.group0˙5˙L2(2)
group0˙valid˙in˙tgroup
      by auto
    with ‹u∈int(H)› have (h-u)+u∈–LeftTranslation(G,f,(h-u))r. r∈int(H)″ by force
ultimately
    have (h-u)+u∈(h-u)+(int(H)) unfolding ltrans˙def by auto moreover
    have (h-u)+u=h using group0.inv˙cancel˙two(1) group0˙valid˙in˙tgroup
      ‹u∈G›‹h∈G› by auto ultimately
    have h∈(h-u)+(int(H)) by auto
    with P ss have ∃V∈T. h∈V∧ V⊆H unfolding Bex˙def by auto
  }
  then show thesis using open˙neigh˙open by auto
qed
```

In conclusion, a subgroup is either open or has empty interior.

```
corollary(in topgroup) emptyInterior˙xor˙op:
  assumes IsAsubgroup(H,f)
  shows (int(H)=0) Xor (H∈T)
  unfolding Xor˙def using clopen˙or˙emptyInt assms Top˙2˙L3
  group0.group0˙3˙L5 group0˙valid˙in˙tgroup by force
```

Then no connected topological groups has proper subgroups with non-empty

interior.

```
corollary(in topgroup) connected˙emptyInterior:
  assumes IsAsubgroup(H,f) T–is connected″
  shows (int(H)=0) Xor (H=G)
proof-
 have (int(H)=0) Xor (H∈T) using emptyInterior˙xor˙op assms(1) by auto moreover
  {
    assume H∈T moreover
     then have H–is closed in″T using open˙subgroup˙closed assms(1) by auto ultimately
    have H=0∨H=G using assms(2) unfolding IsConnected˙def by auto
     then have H=G using group0.group0˙3˙L5 group0˙valid˙in˙tgroup assms(1) by auto
  } moreover
  have G∈T using topSpaceAssum unfolding IsATopology˙def G˙def by auto
  ultimately show thesis unfolding Xor˙def by auto
qed
```

Every locally-compact subgroup of a T_0 group is closed.

```
theorem (in topgroup) loc˙compact˙T0˙closed:
  assumes IsAsubgroup(H,f) (T–restricted to″H)–is locally-compact″ T–is T0″
  shows H–is closed in″T
proof-
  from assms(1) have clsub:IsAsubgroup(cl(H),f) using closure˙subgroup by auto
  then have subcl:cl(H)⊆G using group0.group0˙3˙L2 group0˙valid˙in˙tgroup by force
  from assms(1) have sub:H⊆G using group0.group0˙3˙L2 group0˙valid˙in˙tgroup by force
 from assms(3) have T–is T2″ using T1˙imp˙T2 neu˙closed˙imp˙T1 T0˙imp˙neu˙closed by auto
  then have (T–restricted to″H)–is T2″ using T2˙here sub by auto
  have tot:⋃(T–restricted to″H)=H using sub unfolding RestrictedTo˙def by auto
  with assms(2) have ∀x∈H. ∃A∈Pow(H). A –is compact in″ (T–restricted to″H) ∧ x ∈ Interior(A, (T–restricted to″H)) using
   topology0.locally˙compact˙exist˙compact˙neig[of T–restricted to″H] Top˙1˙L4 unfolding topology0˙def
    by auto
 then obtain K where K:K⊆H K–is compact in″ (T–restricted to″H)0∈Interior(K,(T–restricted to″H))
     using group0.group0˙3˙L5 group0˙valid˙in˙tgroup assms(1) unfolding gzero˙def by force
  from K(1,2) have K–is compact in″ T using compact˙subspace˙imp˙compact by auto
  with ‹T–is T2″› have Kcl:K–is closed in″T using in˙t2˙compact˙is˙cl by auto
 have Interior(K,(T–restricted to″H))∈(T–restricted to″H) using topology0.Top˙2˙L2 unfolding topology0˙def
    using Top˙1˙L4 by auto
  then obtain U where U:U∈TInterior(K,(T–restricted to″H))=H∩U unfolding RestrictedTo˙def by auto
   then have H∩U⊆K using topology0.Top˙2˙L1[of T–restricted to″H] unfolding
```

topology0˙def using Top˙1˙L4 by force
moreover have U2:U⊆U∪K by auto
have ksub:K⊆H using tot K(2) unfolding IsCompact˙def by auto
ultimately have int:H∩(U∪K)=K by auto
from U(2) K(3) have **0**∈U by auto
with U(1) U2 have **0**∈int(U ∪ K) using Top˙2˙L6 by auto
then have U∪K∈$\mathcal{N}_0$ unfolding zerohoods˙def using U(1) ksub sub by auto
then obtain V where V:V⊆U∪K V∈$\mathcal{N}_0$ V+V⊆U∪K(- V) = V using exists˙procls˙zerohood[of U∪K]
by auto
{
fix h assume AS:h∈cl(H)
with clsub have (-h)∈cl(H) using group0.group0˙3˙T3A group0˙valid˙in˙tgroup by auto moreover
then have (-h)∈G using subcl by auto
with V(2) have (-h)∈int((-h)+V) using elem˙in˙int˙ltrans by auto ultimately
have (-h)∈(cl(H))∩(int((-h)+V)) by auto moreover
have int((-h)+V)∈T using Top˙2˙L2 by auto moreover
note sub ultimately
have H∩(int((-h)+V))≠0 using cl˙inter˙neigh by auto moreover
from ‹(-h)∈G› V(2) have int((-h)+V)=(-h)+int(V) unfolding zerohoods˙def
using ltrans˙interior by force
ultimately have H∩((-h)+int(V))≠0 by auto
then obtain y where y:y∈H y∈(-h)+int(V) by blast
then obtain v where v:v∈int(V) y=(-h)+v unfolding ltrans˙def LeftTranslation˙def by auto
with ‹(-h)∈G› V(2) y(1) sub have v∈G(-h)∈Gy∈G using Top˙2˙L1[of V] unfolding zerohoods˙def by auto
with v(2) have (-(-h))+y=v using group0.group0˙2˙L18(2) group0˙valid˙in˙tgroup
unfolding grop˙def grinv˙def by auto moreover
have h∈G using AS subcl by auto
then have (-(-h))=h using group0.group˙inv˙of˙inv group0˙valid˙in˙tgroup by auto ultimately
have h+y=v by auto
with v(1) have hyV:h+y∈int(V) by auto
have y∈cl(H) using y(1) cl˙contains˙set sub by auto
with AS have hycl:h+ y∈cl(H) using clsub group0.group0˙3˙L6 group0˙valid˙in˙tgroup by auto
{
fix W assume W:W∈Th+y∈W
with hyV have h+y∈int(V)∩W by auto moreover
from W(1) have int(V)∩W∈T using Top˙2˙L2 topSpaceAssum unfolding IsATopology˙def by auto moreover
note hycl sub
ultimately have (int(V)∩W)∩H≠0 using cl˙inter˙neigh[of Hint(V)∩Wh+y] by auto
then have V∩W∩H≠0 using Top˙2˙L1 by auto
with V(1) have (U∪K)∩W∩H≠0 by auto
then have (H∩(U∪K))∩W≠0 by auto

```
    with int have K∩W≠0 by auto
   }
   then have ∀W∈T. h+y∈W ⟶ K∩W≠0 by auto moreover
   have K⊆G h+y∈G using ksub sub hycl subcl by auto ultimately
   have h+y∈cl(K) using inter˙neigh˙cl[of Kh+y] unfolding G˙def by force
   then have h+y∈K using Kcl Top˙3˙L8 ‹K⊆G› by auto
   with ksub have h+y∈H by auto
  moreover from y(1) have (-y)∈H using group0.group0˙3˙T3A assms(1) group0˙valid˙in˙tgroup
    by auto
  ultimately have (h+y)-y∈H unfolding grsub˙def using group0.group0˙3˙L6 group0˙valid˙in˙tgroup
    assms(1) by auto
   moreover
   have (-y)∈G using ‹(-y)∈H› sub by auto
   then have h+(y-y)=(h+y)-y using ‹y∈G›‹h∈G› group0.group˙oper˙assoc
    group0˙valid˙in˙tgroup unfolding grsub˙def by auto
  then have h+0=(h+y)-y using group0.group0˙2˙L6 group0˙valid˙in˙tgroup ‹y∈G›
    unfolding grsub˙def grinv˙def grop˙def gzero˙def by auto
   then have h=(h+y)-y using group0.group0˙2˙L2 group0˙valid˙in˙tgroup
    ‹h∈G› unfolding gzero˙def by auto
   ultimately have h∈H by auto
  }
  then have cl(H)⊆H by auto
  then have H=cl(H) using cl˙contains˙set sub by auto
  then show thesis using Top˙3˙L8 sub by auto
qed
```

We can always consider a factor group which is T_2.

```
theorem(in topgroup) factor˙haus:
  shows (T–quotient by″QuotientGroupRel(G,f,cl(–0″)))–is T₂″
proof-
  let r=QuotientGroupRel(G,f,cl(–0″))
  let f=QuotientGroupOp(G,f,cl(–0″))
  let i=GroupInv(G//r,f)
  have IsAnormalSubgroup(G,f,–0″) using group0.trivial˙normal˙subgroup Ggroup
unfolding group0˙def
    by auto
  then have normal:IsAnormalSubgroup(G,f,cl(–0″)) using normal˙subg by auto
 then have eq:equiv(⋃T,r) using group0.Group˙ZF˙2˙4˙L3[OF group0˙valid˙in˙tgroup]
    unfolding IsAnormalSubgroup˙def by auto
  then have tot:⋃(T–quotient by″r)=G//r using total˙quo˙equi by auto
  have neu:r–0″=TheNeutralElement(G//r,f) using Group˙ZF˙2˙4˙L5B[OF Ggroup
normal] by auto
 then have r–0″∈G//r using group0.group0˙2˙L2 Group˙ZF˙2˙4˙T1[OF Ggroup nor-
mal] unfolding group0˙def by auto
  then have sub1:–r–0″″⊆G//r by auto
  then have sub:–r–0″″⊆⋃(T–quotient by″r) using tot by auto
  have zG:0∈⋃T using group0.group0˙2˙L2[OF group0˙valid˙in˙tgroup] by auto
  from zG have cla:r–0″∈G//r unfolding quotient˙def by auto
  let x=G//r-–r–0″″
```

{
fix s assume A:s∈⋃(G//r-–r–**0**″″)
then obtain U where s∈U U∈G//r-–r–**0**″″ by auto
then have U∈G//r U≠r–**0**″ s∈U by auto
then have U∈G//r s∈U s∉r–**0**″ using cla quotient˙disj[OF eq] by auto
then have s∈⋃(G//r)-r–**0**″ by auto
}
moreover
{
fix s assume A:s∈⋃(G//r)-r–**0**″
then obtain U where s∈U U∈G//r s∉r–**0**″ by auto
then have s∈U U∈G//r-–r–**0**″″ by auto
then have s∈⋃(G//r-–r–**0**″″) by auto
}
ultimately have ⋃(G//r-–r–**0**″″)=⋃(G//r)-r–**0**″ by auto
then have A:⋃(G//r-–r–**0**″″)=G-r–**0**″ using Union˙quotient eq by auto
{
fix s assume A:s∈r–**0**″
then have ⟨**0**,s⟩∈r by auto
then have ⟨s,**0**⟩∈r using eq unfolding equiv˙def sym˙def by auto
then have s∈cl(–**0**″) using group0.Group˙ZF˙2˙4˙L5C[OF group0˙valid˙in˙tgroup] unfolding QuotientGroupRel˙def by auto
}
moreover
{
fix s assume A:s∈cl(–**0**″)
then have s∈G using Top˙3˙L11(1) zG by auto
then have ⟨s,**0**⟩∈r using group0.Group˙ZF˙2˙4˙L5C[OF group0˙valid˙in˙tgroup] A by auto
then have ⟨**0**,s⟩∈r using eq unfolding equiv˙def sym˙def by auto
then have s∈r–**0**″ by auto
}
ultimately have r–**0**″=cl(–**0**″) by blast
with A have ⋃(G//r-–r–**0**″″)=G-cl(–**0**″) by auto
moreover have cl(–**0**″)–is closed in″T using cl˙is˙closed zG by auto
ultimately have ⋃(G//r-–r–**0**″″)∈T unfolding IsClosed˙def by auto
then have (G//r-–r–**0**″″)∈–quotient by″r using quotient˙equiv˙rel eq by auto
then have (⋃(T–quotient by″r)-–r–**0**″″)∈–quotient by″r using total˙quo˙equi[OF eq] by auto
moreover from sub1 have –r–**0**″″⊆(⋃(T–quotient by″r)) using total˙quo˙equi[OF eq] by auto
ultimately have –r–**0**″″–is closed in″(T–quotient by″r) unfolding IsClosed˙def by auto
then have –TheNeutralElement(G//r,f)″–is closed in″(T–quotient by″r) using neu by auto
then have (T–quotient by″r)–is T_1″ using topgroup.neu˙closed˙imp˙T1[OF topGroupLocale[OF quotient˙top˙group[OF normal]]]
total˙quo˙equi[OF eq] by auto
then show thesis using topgroup.T1˙imp˙T2[OF topGroupLocale[OF quotient˙top˙group[OF

normal]]] by auto
qed

end

77 Metamath introduction

theory MMI˙prelude imports Order˙ZF˙1

begin

Metamath's set.mm features a large (over 8000) collection of theorems proven in the ZFC set theory. This theory is part of an attempt to translate those theorems to Isar so that they are available for Isabelle/ZF users. A total of about 1200 assertions have been translated, 600 of that with proofs (the rest was proven automatically by Isabelle). The translation was done with the support of the mmisar tool, whose source is included in the IsarMathLib distributions prior to version 1.6.4. The translation tool was doing about 99 percent of work involved, with the rest mostly related to the difference between Isabelle/ZF and Metamath metalogics. Metamath uses Tarski-Megill metalogic that does not have a notion of bound variables (see http://planetx.cc.vt.edu/AsteroidMeta/Distinctors˙vs˙binders for details and discussion). The translation project is closed now as I decided that it was too boring and tedious even with the support of mmisar software. Also, the translated proofs are not as readable as native Isar proofs which goes against IsarMathLib philosophy.

77.1 Importing from Metamath - how is it done

We are interested in importing the theorems about complex numbers that start from the "recnt" theorem on. This is done mostly automatically by the mmisar tool that is included in the IsarMathLib distributions prior to version 1.6.4. The tool works as follows:

First it reads the list of (Metamath) names of theorems that are already imported to IsarMathlib ("known theorems") and the list of theorems that are intended to be imported in this session ("new theorems"). The new theorems are consecutive theorems about complex numbers as they appear in the Metamath database. Then mmisar creates a "Metamath script" that contains Metamath commands that open a log file and put the statements and proofs of the new theorems in that file in a readable format. The tool writes this script to a disk file and executes metamath with standard input redirected from that file. Then the log file is read and its contents converted to the Isar format. In Metamath, the proofs of theorems about complex numbers depend only on 28 axioms of complex numbers and some basic

logic and set theory theorems. The tool finds which of these dependencies are not known yet and repeats the process of getting their statements from Metamath as with the new theorems. As a result of this process mmisar creates files new_theorems.thy, new_deps.thy and new_known_theorems.txt. The file new_theorems.thy contains the theorems (with proofs) imported from Metamath in this session. These theorems are added (by hand) to the current MMI˙Complex˙ZF˙x.thy file. The file new_deps.thy contains the statements of new dependencies with generic proofs "by auto". These are added to the MMI˙logic˙and˙sets.thy. Most of the dependencies can be proven automatically by Isabelle. However, some manual work has to be done for the dependencies that Isabelle can not prove by itself and to correct problems related to the fact that Metamath uses a metalogic based on distinct variable constraints (Tarski-Megill metalogic), rather than an explicit notion of free and bound variables.

The old list of known theorems is replaced by the new list and mmisar is ready to convert the next batch of new theorems. Of course this rarely works in practice without tweaking the mmisar source files every time a new batch is processed.

77.2 The context for Metamath theorems

We list the Metamth's axioms of complex numbers and define notation here.

The next definition is what Metamath $X \in V$ is translated to. I am not sure why it works, probably because Isabelle does a type inference and the "=" sign indicates that both sides are sets.

```
definition
  IsASet :: i⇒o (˙ isASet [90] 90) where

  IsASet˙def[simp]: X isASet ≡  X = X
```

The next locale sets up the context to which Metamath theorems about complex numbers are imported. It assumes the axioms of complex numbers and defines the notation used for complex numbers.

One of the problems with importing theorems from Metamath is that Metamath allows direct infix notation for binary operations so that the notation afb is allowed where f is a function (that is, a set of pairs). To my knowledge, Isar allows only notation f⟨a,b⟩ with a possibility of defining a syntax say a + b to mean the same as f⟨a,b⟩ (please correct me if I am wrong here). This is why we have two objects for addition: one called caddset that represents the binary function, and the second one called ca which defines the a + b notation for caddset⟨a,b⟩. The same applies to multiplication of real numbers.

Another difficulty is that Metamath allows to define sets with syntax $\{x|p\}$

where p is some formula that (usually) depends on x. Isabelle allows the set comprehension like this only as a subset of another set i.e. $\{x \in A.p(x)\}$. This forces us to have a sligtly different definition of (complex) natural numbers, requiring explicitly that natural numbers is a subset of reals. Because of that, the proofs of Metamath theorems that reference the definition directly can not be imported.

```
locale MMIsar0 =
  fixes real (ℝ)
  fixes complex (ℂ)
  fixes one (𝟏)
  fixes zero (𝟎)
  fixes iunit (i)
  fixes caddset (+)
  fixes cmulset (·)
  fixes lessrrel (<ℝ)

  fixes ca (infixl + 69)
  defines ca˙def: a + b ≡ +⟨a,b⟩
  fixes cm (infixl · 71)
  defines cm˙def: a · b ≡ ·⟨a,b⟩
  fixes sub (infixl - 69)
  defines sub˙def: a - b ≡ ⋃ − x ∈ ℂ. b + x = a ″
  fixes cneg (-˙ 95)
  defines cneg˙def: - a ≡ 𝟎 - a
  fixes cdiv (infixl / 70)
  defines cdiv˙def: a / b ≡ ⋃ − x ∈ ℂ. b · x = a ″
  fixes cpnf (+∞)
  defines cpnf˙def: +∞ ≡ ℂ
  fixes cmnf (−∞)
  defines cmnf˙def: −∞ ≡ −ℂ ″
  fixes cxr (ℝ*)
  defines cxr˙def: ℝ* ≡ ℝ ∪ −+∞,−∞″
  fixes cxn (ℕ)
  defines cxn˙def: ℕ ≡ ⋂ −N ∈ Pow(ℝ). 𝟏 ∈ N ∧ (∀n. n∈N ⟶ n+𝟏 ∈ N)″
  fixes lessr (infix <ℝ 68)
  defines lessr˙def: a <ℝ b ≡ ⟨a,b⟩ ∈ <ℝ
  fixes cltrrset (¡)
  defines cltrrset˙def:
  ¡ ≡ (<ℝ ∩ ℝ×ℝ) ∪ −⟨−∞,+∞⟩″ ∪
  (ℝ×−+∞″) ∪ (−−∞″×ℝ )
  fixes cltrr (infix ¡ 68)
  defines cltrr˙def: a ¡ b ≡ ⟨a,b⟩ ∈ ¡
  fixes convcltrr (infix ¿ 68)
  defines convcltrr˙def: a ¿ b ≡ ⟨a,b⟩ ∈ converse(¡)
  fixes lsq (infix ≤ 68)
  defines lsq˙def: a ≤ b ≡ ¬ (b ¡ a)
  fixes two (𝟐)
  defines two˙def: 𝟐 ≡ 𝟏+𝟏
```

fixes three ($\mathbf{3}$)
defines three˙def: $\mathbf{3} \equiv \mathbf{2}+\mathbf{1}$
fixes four ($\mathbf{4}$)
defines four˙def: $\mathbf{4} \equiv \mathbf{3}+\mathbf{1}$
fixes five ($\mathbf{5}$)
defines five˙def: $\mathbf{5} \equiv \mathbf{4}+\mathbf{1}$
fixes six ($\mathbf{6}$)
defines six˙def: $\mathbf{6} \equiv \mathbf{5}+\mathbf{1}$
fixes seven ($\mathbf{7}$)
defines seven˙def: $\mathbf{7} \equiv \mathbf{6}+\mathbf{1}$
fixes eight ($\mathbf{8}$)
defines eight˙def: $\mathbf{8} \equiv \mathbf{7}+\mathbf{1}$
fixes nine ($\mathbf{9}$)
defines nine˙def: $\mathbf{9} \equiv \mathbf{8}+\mathbf{1}$

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: $+ : (\mathbb{C} \times \mathbb{C}) \to \mathbb{C}$
assumes MMI˙axmulopr: $\cdot : (\mathbb{C} \times \mathbb{C}) \to \mathbb{C}$
assumes MMI˙axmulcom: $A \in \mathbb{C} \wedge B \in \mathbb{C} \longrightarrow A \cdot B = B \cdot A$
assumes MMI˙axaddcl: $A \in \mathbb{C} \wedge B \in \mathbb{C} \longrightarrow A + B \in \mathbb{C}$
assumes MMI˙axmulcl: $A \in \mathbb{C} \wedge B \in \mathbb{C} \longrightarrow A \cdot B \in \mathbb{C}$
assumes MMI˙axdistr:
$A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C} \longrightarrow A{\cdot}(B + C) = A{\cdot}B + A{\cdot}C$
assumes MMI˙axaddcom: $A \in \mathbb{C} \wedge B \in \mathbb{C} \longrightarrow A + B = B + A$
assumes MMI˙axaddass:
$A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C} \longrightarrow A + B + C = A + (B + C)$
assumes MMI˙axmulass:
$A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C} \longrightarrow A \cdot B \cdot C = A \cdot (B \cdot C)$
assumes MMI˙ax1re: $\mathbf{1} \in \mathbb{R}$
assumes MMI˙axi2m1: $i \cdot i + \mathbf{1} = \mathbf{0}$
assumes MMI˙ax0id: $A \in \mathbb{C} \longrightarrow A + \mathbf{0} = A$
assumes MMI˙axicn: $i \in \mathbb{C}$
assumes MMI˙axnegex: $A \in \mathbb{C} \longrightarrow (\exists\ x \in \mathbb{C}.\ (A + x) = \mathbf{0})$
assumes MMI˙axrecex: $A \in \mathbb{C} \wedge A \neq \mathbf{0} \longrightarrow (\exists\ x \in \mathbb{C}.\ A \cdot x = \mathbf{1})$
assumes MMI˙ax1id: $A \in \mathbb{C} \longrightarrow A \cdot \mathbf{1} = A$
assumes MMI˙axaddrcl: $A \in \mathbb{R} \wedge B \in \mathbb{R} \longrightarrow A + B \in \mathbb{R}$

assumes MMI˙axmulrcl: $A \in \mathbb{R} \wedge B \in \mathbb{R} \longrightarrow A \cdot B \in \mathbb{R}$
assumes MMI˙axrnegex: $A \in \mathbb{R} \longrightarrow (\exists\ x \in \mathbb{R}.\ A + x = \mathbf{0})$
assumes MMI˙axrrecex: $A \in \mathbb{R} \wedge A \neq \mathbf{0} \longrightarrow (\exists\ x \in \mathbb{R}.\ A \cdot x = \mathbf{1})$

end

78 Logic and sets in Metamatah

theory MMI˙logic˙and˙sets imports MMI˙prelude

begin

78.1 Basic Metamath theorems

This section contains Metamath theorems that the more advanced theorems from MMIsar.thy depend on. Most of these theorems are proven automatically by Isabelle, some have to be proven by hand and some have to be modified to convert from Tarski-Megill metalogic used by Metamath to one based on explicit notion of free and bound variables.

lemma MMI˙ax˙mp: assumes φ and $\varphi \longrightarrow \psi$ shows ψ
using assms by auto

lemma MMI˙sseli: assumes A1: $A \subseteq B$
shows $C \in A \longrightarrow C \in B$
using assms by auto

lemma MMI˙sselii: assumes A1: $A \subseteq B$ and
A2: $C \in A$
shows $C \in B$
using assms by auto

lemma MMI˙syl: assumes A1: $\varphi \longrightarrow ps$ and
A2: $ps \longrightarrow ch$
shows $\varphi \longrightarrow ch$
using assms by auto

lemma MMI˙elimhyp: assumes A1: $A = \text{if} (\varphi , A , B) \longrightarrow (\varphi \longleftrightarrow \psi)$ and
A2: $B = \text{if} (\varphi , A , B) \longrightarrow (ch \longleftrightarrow \psi)$ and
A3: ch
shows ψ
proof -
{ assume φ
with A1 have ψ by simp }
moreover
{ assume $\neg\varphi$
with A2 A3 have ψ by simp }
ultimately show ψ by auto

qed

lemma MMI˙neeq1:
shows A = B ⟶ (A ≠ C ⟷ B ≠ C)
by auto

lemma MMI˙mp2: assumes A1: φ and
A2: ψ and
A3: φ ⟶ (ψ ⟶ chi)
shows chi
using assms by auto

lemma MMI˙xpex: assumes A1: A isASet and
A2: B isASet
shows (A × B) isASet
using assms by auto

lemma MMI˙fex:
shows
A ∈ C ⟶ (F : A → B ⟶ F isASet)
A isASet ⟶ (F : A → B ⟶ F isASet)
by auto

lemma MMI˙3eqtr4d: assumes A1: φ ⟶ A = B and
A2: φ ⟶ C = A and
A3: φ ⟶ D = B
shows φ ⟶ C = D
using assms by auto

lemma MMI˙3coml: assumes A1: (φ ∧ ψ ∧ chi) ⟶ th
shows (ψ ∧ chi ∧ φ) ⟶ th
using assms by auto

lemma MMI˙sylan: assumes A1: (φ ∧ ψ) ⟶ chi and
A2: th ⟶ φ
shows (th ∧ ψ) ⟶ chi
using assms by auto

lemma MMI˙3impa: assumes A1: ((φ ∧ ψ) ∧ chi) ⟶ th
shows (φ ∧ ψ ∧ chi) ⟶ th
using assms by auto

lemma MMI˙3adant2: assumes A1: (φ ∧ ψ) ⟶ chi
shows (φ ∧ th ∧ ψ) ⟶ chi
using assms by auto

lemma MMI˙3adant1: assumes A1: (φ ∧ ψ) ⟶ chi
shows (th ∧ φ ∧ ψ) ⟶ chi
using assms by auto

lemma (in MMIsar0) MMI˙opreq12d: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\varphi \longrightarrow$ C = D
shows
$\varphi \longrightarrow$ (A + C) = (B + D)
$\varphi \longrightarrow$ (A · C) = (B · D)
$\varphi \longrightarrow$ (A - C) = (B - D)
$\varphi \longrightarrow$ (A / C) = (B / D)
using assms by auto

lemma MMI˙mp2an: assumes A1: φ and
A2: ψ and
A3: ($\varphi \wedge \psi$) $\longrightarrow$ chi
shows chi
using assms by auto

lemma MMI˙mp3an: assumes A1: φ and
A2: ψ and
A3: ch and
A4: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
shows ϑ
using assms by auto

lemma MMI˙eqeltrr: assumes A1: A = B and
A2: A $\in$ C
shows B $\in$ C
using assms by auto

lemma MMI˙eqtr: assumes A1: A = B and
A2: B = C
shows A = C
using assms by auto

lemma MMI˙impbi: assumes A1: $\varphi \longrightarrow \psi$ and
A2: $\psi \longrightarrow \varphi$
shows $\varphi \longleftrightarrow \psi$
proof
assume φ with A1 show ψ by simp
next
assume ψ with A2 show φ by simp
qed

lemma MMI˙mp3an3: assumes A1: ch and
A2: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
shows ($\varphi \wedge \psi$) $\longrightarrow \vartheta$
using assms by auto

lemma MMI˙eqeq12d: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\varphi \longrightarrow$ C = D
shows $\varphi \longrightarrow$ (A = C $\longleftrightarrow$ B = D)
using assms by auto

lemma MMI˙mpan2: assumes A1: ψ and
A2: ($\varphi \wedge \psi$) $\longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
using assms by auto

lemma (in MMIsar0) MMI˙opreq2:
shows
A = B $\longrightarrow$ (C + A) = (C + B)
A = B $\longrightarrow$ (C · A) = (C · B)
A = B $\longrightarrow$ (C - A) = (C - B)
A = B $\longrightarrow$ (C / A) = (C / B)
by auto

lemma MMI˙syl5bir: assumes A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) and
A2: $\vartheta \longrightarrow$ ch
shows $\varphi \longrightarrow$ ($\vartheta \longrightarrow \psi$)
using assms by auto

lemma MMI˙adantr: assumes A1: $\varphi \longrightarrow \psi$
shows ($\varphi \wedge$ ch) $\longrightarrow \psi$
using assms by auto

lemma MMI˙mpan: assumes A1: φ and
A2: ($\varphi \wedge \psi$) $\longrightarrow$ ch
shows $\psi \longrightarrow$ ch
using assms by auto

lemma MMI˙eqeq1d: assumes A1: $\varphi \longrightarrow$ A = B
shows $\varphi \longrightarrow$ (A = C $\longleftrightarrow$ B = C)
using assms by auto

lemma (in MMIsar0) MMI˙opreq1:
shows
A = B $\longrightarrow$ (A · C) = (B · C)
A = B $\longrightarrow$ (A + C) = (B + C)
A = B $\longrightarrow$ (A - C) = (B - C)
A = B $\longrightarrow$ (A / C) = (B / C)
by auto

lemma MMI˙syl6eq: assumes A1: $\varphi \longrightarrow$ A = B and
A2: B = C
shows $\varphi \longrightarrow$ A = C
using assms by auto

lemma MMI˙syl6bi: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$ and
A2: $\mathrm{ch} \longrightarrow \vartheta$
shows $\varphi \longrightarrow (\psi \longrightarrow \vartheta)$
using assms by auto

lemma MMI˙imp: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow \mathrm{ch})$
shows $(\varphi \wedge \psi) \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙sylibd: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow \mathrm{ch})$ and
A2: $\varphi \longrightarrow (\mathrm{ch} \longleftrightarrow \vartheta)$
shows $\varphi \longrightarrow (\psi \longrightarrow \vartheta)$
using assms by auto

lemma MMI˙ex: assumes A1: $(\varphi \wedge \psi) \longrightarrow \mathrm{ch}$
shows $\varphi \longrightarrow (\psi \longrightarrow \mathrm{ch})$
using assms by auto

lemma MMI˙r19˙23aiv: assumes A1: $\forall \mathrm{x}. \ (\mathrm{x} \in \mathrm{A} \ \longrightarrow (\varphi(\mathrm{x}) \longrightarrow \psi))$
shows $(\exists \ \mathrm{x} \in \mathrm{A} \ . \ \varphi(\mathrm{x})) \longrightarrow \psi$
using assms by auto

lemma MMI˙bitr: assumes A1: $\varphi \longleftrightarrow \psi$ and
A2: $\psi \longleftrightarrow \mathrm{ch}$
shows $\varphi \longleftrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙eqeq12i: assumes A1: $\mathrm{A} = \mathrm{B}$ and
A2: $\mathrm{C} = \mathrm{D}$
shows $\mathrm{A} = \mathrm{C} \longleftrightarrow \mathrm{B} = \mathrm{D}$
using assms by auto

lemma MMI˙dedth3h:
assumes A1: $\mathrm{A} = \mathrm{if} (\varphi , \mathrm{A} , \mathrm{D}) \longrightarrow (\vartheta \longleftrightarrow \mathrm{ta})$ and
A2: $\mathrm{B} = \mathrm{if} (\psi , \mathrm{B} , \mathrm{R}) \longrightarrow (\mathrm{ta} \longleftrightarrow \mathrm{et})$ and
A3: $\mathrm{C} = \mathrm{if} (\mathrm{ch} , \mathrm{C} , \mathrm{S}) \longrightarrow (\mathrm{et} \longleftrightarrow \mathrm{ze})$ and
A4: ze
shows $(\varphi \wedge \psi \wedge \mathrm{ch}) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙bibi1d: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$
shows $\varphi \longrightarrow ((\psi \longleftrightarrow \vartheta) \longleftrightarrow (\mathrm{ch} \longleftrightarrow \vartheta))$
using assms by auto

lemma MMI˙eqeq1:
shows $\mathrm{A} = \mathrm{B} \longrightarrow (\mathrm{A} = \mathrm{C} \longleftrightarrow \mathrm{B} = \mathrm{C})$
by auto

lemma MMI˙bibi12d: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$ and

A2: $\varphi \longrightarrow (\vartheta \longleftrightarrow \text{ta})$
shows $\varphi \longrightarrow ((\psi \longleftrightarrow \vartheta) \longleftrightarrow (\text{ch} \longleftrightarrow \text{ta}))$
using assms by auto

lemma MMI˙eqeq2d: assumes A1: $\varphi \longrightarrow A = B$
shows $\varphi \longrightarrow (C = A \longleftrightarrow C = B)$
using assms by auto

lemma MMI˙eqeq2:
shows $A = B \longrightarrow (C = A \longleftrightarrow C = B)$
by auto

lemma MMI˙elimel: assumes A1: $B \in C$
shows if $(A \in C , A , B) \in C$
using assms by auto

lemma MMI˙3adant3: assumes A1: $(\varphi \wedge \psi) \longrightarrow \text{ch}$
shows $(\varphi \wedge \psi \wedge \vartheta) \longrightarrow \text{ch}$
using assms by auto

lemma MMI˙bitr3d: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \text{ch})$ and
A2: $\varphi \longrightarrow (\psi \longleftrightarrow \vartheta)$
shows $\varphi \longrightarrow (\text{ch} \longleftrightarrow \vartheta)$
using assms by auto

lemma MMI˙3eqtr3d: assumes A1: $\varphi \longrightarrow A = B$ and
A2: $\varphi \longrightarrow A = C$ and
A3: $\varphi \longrightarrow B = D$
shows $\varphi \longrightarrow C = D$
using assms by auto

lemma (in MMIsar0) MMI˙opreq1d: assumes A1: $\varphi \longrightarrow A = B$
shows
$\varphi \longrightarrow (A + C) = (B + C)$
$\varphi \longrightarrow (A - C) = (B - C)$
$\varphi \longrightarrow (A \cdot C) = (B \cdot C)$
$\varphi \longrightarrow (A / C) = (B / C)$
using assms by auto

lemma MMI˙3com12: assumes A1: $(\varphi \wedge \psi \wedge \text{ch}) \longrightarrow \vartheta$
shows $(\psi \wedge \varphi \wedge \text{ch}) \longrightarrow \vartheta$
using assms by auto

lemma (in MMIsar0) MMI˙opreq2d: assumes A1: $\varphi \longrightarrow A = B$
shows
$\varphi \longrightarrow (C + A) = (C + B)$
$\varphi \longrightarrow (C - A) = (C - B)$

$\varphi \longrightarrow (C \cdot A) = (C \cdot B)$
$\varphi \longrightarrow (C / A) = (C / B)$
using assms by auto

lemma MMI˙3com23: assumes A1: $(\varphi \wedge \psi \wedge ch) \longrightarrow \vartheta$
shows $(\varphi \wedge ch \wedge \psi) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙3expa: assumes A1: $(\varphi \wedge \psi \wedge ch) \longrightarrow \vartheta$
shows $((\varphi \wedge \psi) \wedge ch) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙adantrr: assumes A1: $(\varphi \wedge \psi) \longrightarrow ch$
shows $(\varphi \wedge (\psi \wedge \vartheta)) \longrightarrow ch$
using assms by auto

lemma MMI˙3expb: assumes A1: $(\varphi \wedge \psi \wedge ch) \longrightarrow \vartheta$
shows $(\varphi \wedge (\psi \wedge ch)) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙an4s: assumes A1: $((\varphi \wedge \psi) \wedge (ch \wedge \vartheta)) \longrightarrow \tau$
shows $((\varphi \wedge ch) \wedge (\psi \wedge \vartheta)) \longrightarrow \tau$
using assms by auto

lemma MMI˙eqtrd: assumes A1: $\varphi \longrightarrow A = B$ and
A2: $\varphi \longrightarrow B = C$
shows $\varphi \longrightarrow A = C$
using assms by auto

lemma MMI˙ad2ant2l: assumes A1: $(\varphi \wedge \psi) \longrightarrow ch$
shows $((\vartheta \wedge \varphi) \wedge (\tau \wedge \psi)) \longrightarrow ch$
using assms by auto

lemma MMI˙pm3˙2i: assumes A1: φ and
A2: ψ
shows $\varphi \wedge \psi$
using assms by auto

lemma (in MMIsar0) MMI˙opreq2i: assumes A1: $A = B$
shows
$(C + A) = (C + B)$
$(C - A) = (C - B)$
$(C \cdot A) = (C \cdot B)$
using assms by auto

lemma MMI˙mpbir2an: assumes A1: $\varphi \longleftrightarrow (\psi \wedge ch)$ and
A2: ψ and

A3: ch
shows φ
using assms by auto

lemma MMI˙reu4: assumes A1: $\forall$x y. x = y $\longrightarrow$ (φ(x) $\longleftrightarrow$ ψ(y))
shows ($\exists$! x . x $\in$ A $\wedge$ φ(x)) $\longleftrightarrow$
(($\exists$ x $\in$ A . φ(x)) $\wedge$ ($\forall$ x $\in$ A . $\forall$ y $\in$ A .
((φ(x) $\wedge$ ψ(y)) $\longrightarrow$ x = y)))
using assms by auto

lemma MMI˙risset:
shows A $\in$ B $\longleftrightarrow$ ($\exists$ x $\in$ B . x = A)
by auto

lemma MMI˙sylib: assumes A1: φ $\longrightarrow$ ψ and
A2: ψ $\longleftrightarrow$ ch
shows φ $\longrightarrow$ ch
using assms by auto

lemma MMI˙mp3an13: assumes A1: φ and
A2: ch and
A3: (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ
shows ψ $\longrightarrow$ ϑ
using assms by auto

lemma MMI˙eqcomd: assumes A1: φ $\longrightarrow$ A = B
shows φ $\longrightarrow$ B = A
using assms by auto

lemma MMI˙sylan9eqr: assumes A1: φ $\longrightarrow$ A = B and
A2: ψ $\longrightarrow$ B = C
shows (ψ $\wedge$ φ) $\longrightarrow$ A = C
using assms by auto

lemma MMI˙exp32: assumes A1: (φ $\wedge$ (ψ $\wedge$ ch)) $\longrightarrow$ ϑ
shows φ $\longrightarrow$ (ψ $\longrightarrow$ (ch $\longrightarrow$ ϑ))
using assms by auto

lemma MMI˙impcom: assumes A1: φ $\longrightarrow$ (ψ $\longrightarrow$ ch)
shows (ψ $\wedge$ φ) $\longrightarrow$ ch
using assms by auto

lemma MMI˙a1d: assumes A1: φ $\longrightarrow$ ψ
shows φ $\longrightarrow$ (ch $\longrightarrow$ ψ)
using assms by auto

lemma MMI˙r19˙21aiv: assumes A1: $\forall$x. φ $\longrightarrow$ (x $\in$ A $\longrightarrow$ ψ(x))
shows φ $\longrightarrow$ ($\forall$ x $\in$ A . ψ(x))
using assms by auto

lemma MMI˙r19˙22:
shows (∀ x ∈ A . (φ(x) ⟶ ψ(x))) ⟶
((∃ x ∈ A . φ(x)) ⟶ (∃ x ∈ A . ψ(x)))
by auto

lemma MMI˙syl6: assumes A1: φ ⟶ (ψ ⟶ ch) and
A2: ch ⟶ ϑ
shows φ ⟶ (ψ ⟶ ϑ)
using assms by auto

lemma MMI˙mpid: assumes A1: φ ⟶ ch and
A2: φ ⟶ (ψ ⟶ (ch ⟶ ϑ))
shows φ ⟶ (ψ ⟶ ϑ)
using assms by auto

lemma MMI˙eqtr3t:
shows (A = C ∧ B = C) ⟶ A = B
by auto

lemma MMI˙syl5bi: assumes A1: φ ⟶ (ψ ⟷ ch) and
A2: ϑ ⟶ ψ
shows φ ⟶ (ϑ ⟶ ch)
using assms by auto

lemma MMI˙mp3an1: assumes A1: φ and
A2: (φ ∧ ψ ∧ ch) ⟶ ϑ
shows (ψ ∧ ch) ⟶ ϑ
using assms by auto

lemma MMI˙rgen2: assumes A1: ∀x y. (x ∈ A ∧ y ∈ A) ⟶ φ(x,y)
shows ∀ x ∈ A . ∀ y ∈ A . φ(x,y)
using assms by auto

lemma MMI˙ax˙17: shows φ ⟶ (∀x. φ) by simp

lemma MMI˙3eqtr4g: assumes A1: φ ⟶ A = B and
A2: C = A and
A3: D = B
shows φ ⟶ C = D
using assms by auto

lemma MMI˙3imtr4: assumes A1: φ ⟶ ψ and
A2: ch ⟷ φ and

A3: $\vartheta \longleftrightarrow \psi$
shows ch $\longrightarrow \vartheta$
using assms by auto

lemma MMI˙eleq2i: assumes A1: A = B
shows C $\in$ A $\longleftrightarrow$ C $\in$ B
using assms by auto

lemma MMI˙albii: assumes A1: $\varphi \longleftrightarrow \psi$
shows ($\forall$ x . φ) $\longleftrightarrow$ ($\forall$ x . ψ)
using assms by auto

lemma MMI˙reucl:
shows ($\exists!$ x . x $\in$ A $\wedge$ φ(x)) $\longrightarrow$ $\bigcup$ − x $\in$ A . φ(x) ″ $\in$ A
proof
assume A1: $\exists!$ x . x $\in$ A $\wedge$ φ(x)
then obtain a where I: a$\in$A and φ(a) by auto
with A1 have − x $\in$ A . φ(x) ″ = −a″ by blast
with I show $\bigcup$ − x $\in$ A . φ(x) ″ $\in$ A by simp
qed

lemma MMI˙dedth2h: assumes A1: A = if (φ , A , C) $\longrightarrow$ (ch $\longleftrightarrow \vartheta$) and
A2: B = if (ψ , B , D) $\longrightarrow$ ($\vartheta \longleftrightarrow \tau$) and
A3: τ
shows ($\varphi \wedge \psi$) $\longrightarrow$ ch
using assms by auto

lemma MMI˙eleq1d: assumes A1: $\varphi \longrightarrow$ A = B
shows $\varphi \longrightarrow$ (A $\in$ C $\longleftrightarrow$ B $\in$ C)
using assms by auto

lemma MMI˙syl5eqel: assumes A1: $\varphi \longrightarrow$ A $\in$ B and
A2: C = A
shows $\varphi \longrightarrow$ C $\in$ B
using assms by auto

lemma IML˙eeuni: assumes A1: x $\in$ A and A2: $\exists!$ t . t $\in$ A $\wedge$ φ(t)
shows φ(x) $\longleftrightarrow$ $\bigcup$ − x $\in$ A . φ(x) ″ = x
proof
assume φ(x)
with A1 A2 show $\bigcup$ − x $\in$ A . φ(x) ″ = x by auto
next assume A3: $\bigcup$ − x $\in$ A . φ(x) ″ = x
from A2 obtain y where y$\in$A and I: φ(y) by auto
with A2 A3 have x = y by auto

with I show φ(x) by simp
qed

lemma MMI˙reuuni1:
shows (x $\in$ A $\wedge$ ($\exists$! x . x $\in$ A $\wedge$ φ(x))) $\longrightarrow$
(φ(x) $\longleftrightarrow$ $\bigcup$ − x $\in$ A . φ(x) ″ = x)
using IML˙eeuni by simp

lemma MMI˙eqeq1i: assumes A1: A = B
shows A = C $\longleftrightarrow$ B = C
using assms by auto

lemma MMI˙syl6rbbr: assumes A1: $\forall$x. φ(x) $\longrightarrow$ (ψ(x) $\longleftrightarrow$ ch(x)) and
A2: $\forall$x. ϑ(x) $\longleftrightarrow$ ch(x)
shows $\forall$ x. φ(x) $\longrightarrow$ (ϑ(x) $\longleftrightarrow$ ψ(x))
using assms by auto

lemma MMI˙syl6rbbrA: assumes A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch) and
A2: ϑ $\longleftrightarrow$ ch
shows φ $\longrightarrow$ (ϑ $\longleftrightarrow$ ψ)
using assms by auto

lemma MMI˙vtoclga: assumes A1: $\forall$x. x = A $\longrightarrow$ (φ(x) $\longleftrightarrow$ ψ) and
A2: $\forall$x. x $\in$ B $\longrightarrow$ φ(x)
shows A $\in$ B $\longrightarrow$ ψ
using assms by auto

lemma MMI˙3bitr4: assumes A1: φ $\longleftrightarrow$ ψ and
A2: ch $\longleftrightarrow$ φ and
A3: ϑ $\longleftrightarrow$ ψ
shows ch $\longleftrightarrow$ ϑ
using assms by auto

lemma MMI˙mpbii: assumes Amin: ψ and
Amaj: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch)
shows φ $\longrightarrow$ ch
using assms by auto

lemma MMI˙eqid:
shows A = A
by auto

lemma MMI˙pm3˙27:
shows (φ $\wedge$ ψ) $\longrightarrow$ ψ
by auto

lemma MMI˙pm3˙26:
shows ($\varphi \wedge \psi$) $\longrightarrow \varphi$
by auto

lemma MMI˙ancoms: assumes A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch
shows ($\psi \wedge \varphi$) $\longrightarrow$ ch
using assms by auto

lemma MMI˙syl3anc: assumes A1: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$ and
A2: $\tau \longrightarrow \varphi$ and
A3: $\tau \longrightarrow \psi$ and
A4: $\tau \longrightarrow$ ch
shows $\tau \longrightarrow \vartheta$
using assms by auto

lemma MMI˙syl5eq: assumes A1: $\varphi \longrightarrow$ A = B and
A2: C = A
shows $\varphi \longrightarrow$ C = B
using assms by auto

lemma MMI˙eqcomi: assumes A1: A = B
shows B = A
using assms by auto

lemma MMI˙3eqtr: assumes A1: A = B and
A2: B = C and
A3: C = D
shows A = D
using assms by auto

lemma MMI˙mpbir: assumes Amin: ψ and
Amaj: $\varphi \longleftrightarrow \psi$
shows φ
using assms by auto

lemma MMI˙syl3an3: assumes A1: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$ and
A2: $\tau \longrightarrow$ ch
shows ($\varphi \wedge \psi \wedge \tau$) $\longrightarrow \vartheta$
using assms by auto

lemma MMI˙3eqtrd: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\varphi \longrightarrow$ B = C and
A3: $\varphi \longrightarrow$ C = D
shows $\varphi \longrightarrow$ A = D
using assms by auto

lemma MMI˙syl5: assumes A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ ch) and
A2: $\vartheta \longrightarrow \psi$

shows $\varphi \longrightarrow (\vartheta \longrightarrow \mathrm{ch})$
using assms by auto

lemma MMI˙exp3a: assumes A1: $\varphi \longrightarrow ((\psi \wedge \mathrm{ch}) \longrightarrow \vartheta)$
shows $\varphi \longrightarrow (\psi \longrightarrow (\mathrm{ch} \longrightarrow \vartheta))$
using assms by auto

lemma MMI˙com12: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow \mathrm{ch})$
shows $\psi \longrightarrow (\varphi \longrightarrow \mathrm{ch})$
using assms by auto

lemma MMI˙3imp: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow (\mathrm{ch} \longrightarrow \vartheta))$
shows $(\varphi \wedge \psi \wedge \mathrm{ch}) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙3eqtr3: assumes A1: A = B and
A2: A = C and
A3: B = D
shows C = D
using assms by auto

lemma (in MMIsar0) MMI˙opreq1i: assumes A1: A = B
shows
(A + C) = (B + C)
(A - C) = (B - C)
(A / C) = (B / C)
(A · C) = (B · C)
using assms by auto

lemma MMI˙eqtr3: assumes A1: A = B and
A2: A = C
shows B = C
using assms by auto

lemma MMI˙dedth: assumes A1: A = if $(\varphi , \mathrm{A} , \mathrm{B}) \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$ and
A2: ch
shows $\varphi \longrightarrow \psi$
using assms by auto

lemma MMI˙id:
shows $\varphi \longrightarrow \varphi$
by auto

lemma MMI˙eqtr3d: assumes A1: $\varphi \longrightarrow \mathrm{A} = \mathrm{B}$ and
A2: $\varphi \longrightarrow \mathrm{A} = \mathrm{C}$
shows $\varphi \longrightarrow \mathrm{B} = \mathrm{C}$
using assms by auto

lemma MMI˙sylan2: assumes A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch and
A2: $\vartheta \longrightarrow \psi$
shows ($\varphi \wedge \vartheta$) $\longrightarrow$ ch
using assms by auto

lemma MMI˙adantl: assumes A1: $\varphi \longrightarrow \psi$
shows (ch $\wedge \varphi$) $\longrightarrow \psi$
using assms by auto

lemma (in MMIsar0) MMI˙opreq12:
shows
(A = B $\wedge$ C = D) $\longrightarrow$ (A + C) = (B + D)
(A = B $\wedge$ C = D) $\longrightarrow$ (A - C) = (B - D)
(A = B $\wedge$ C = D) $\longrightarrow$ (A $\cdot$ C) = (B $\cdot$ D)
(A = B $\wedge$ C = D) $\longrightarrow$ (A / C) = (B / D)
by auto

lemma MMI˙anidms: assumes A1: ($\varphi \wedge \varphi$) $\longrightarrow \psi$
shows $\varphi \longrightarrow \psi$
using assms by auto

lemma MMI˙anabsan2: assumes A1: ($\varphi \wedge$ ($\psi \wedge \psi$)) $\longrightarrow$ ch
shows ($\varphi \wedge \psi$) $\longrightarrow$ ch
using assms by auto

lemma MMI˙3simp2:
shows ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \psi$
by auto

lemma MMI˙3simp3:
shows ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow$ ch
by auto

lemma MMI˙sylbir: assumes A1: $\psi \longleftrightarrow \varphi$ and
A2: $\psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
using assms by auto

lemma MMI˙3eqtr3g: assumes A1: $\varphi \longrightarrow$ A = B and
A2: A = C and
A3: B = D
shows $\varphi \longrightarrow$ C = D
using assms by auto

lemma MMI˙3bitr: assumes A1: $\varphi \longleftrightarrow \psi$ and
A2: $\psi \longleftrightarrow$ ch and

A3: ch $\longleftrightarrow \vartheta$
shows $\varphi \longleftrightarrow \vartheta$
using assms by auto

lemma MMI˙3bitr3: assumes A1: $\varphi \longleftrightarrow \psi$ and
A2: $\varphi \longleftrightarrow$ ch and
A3: $\psi \longleftrightarrow \vartheta$
shows ch $\longleftrightarrow \vartheta$
using assms by auto

lemma MMI˙eqcom:
shows A = B $\longleftrightarrow$ B = A
by auto

lemma MMI˙syl6bb: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch) and
A2: ch $\longleftrightarrow \vartheta$
shows $\varphi \longrightarrow (\psi \longleftrightarrow \vartheta)$
using assms by auto

lemma MMI˙3bitr3d: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch) and
A2: $\varphi \longrightarrow (\psi \longleftrightarrow \vartheta)$ and
A3: $\varphi \longrightarrow (\text{ch} \longleftrightarrow \tau)$
shows $\varphi \longrightarrow (\vartheta \longleftrightarrow \tau)$
using assms by auto

lemma MMI˙syl3an2: assumes A1: $(\varphi \wedge \psi \wedge \text{ch}) \longrightarrow \vartheta$ and
A2: $\tau \longrightarrow \psi$
shows $(\varphi \wedge \tau \wedge \text{ch}) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙df˙rex:
shows $(\exists\ \text{x} \in \text{A} .\ \varphi(\text{x})) \longleftrightarrow (\exists\ \text{x} .\ (\text{x} \in \text{A} \wedge \varphi(\text{x})))$
by auto

lemma MMI˙mpbi: assumes Amin: φ and
Amaj: $\varphi \longleftrightarrow \psi$
shows ψ
using assms by auto

lemma MMI˙mp3an12: assumes A1: φ and
A2: ψ and
A3: $(\varphi \wedge \psi \wedge \text{ch}) \longrightarrow \vartheta$
shows ch $\longrightarrow \vartheta$
using assms by auto

lemma MMI˙syl5bb: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$ and
A2: $\vartheta \longleftrightarrow \psi$
shows $\varphi \longrightarrow (\vartheta \longleftrightarrow \mathrm{ch})$
using assms by auto

lemma MMI˙eleq1a:
shows $\mathrm{A} \in \mathrm{B} \longrightarrow (\mathrm{C} = \mathrm{A} \longrightarrow \mathrm{C} \in \mathrm{B})$
by auto

lemma MMI˙sylbird: assumes A1: $\varphi \longrightarrow (\mathrm{ch} \longleftrightarrow \psi)$ and
A2: $\varphi \longrightarrow (\mathrm{ch} \longrightarrow \vartheta)$
shows $\varphi \longrightarrow (\psi \longrightarrow \vartheta)$
using assms by auto

lemma MMI˙19˙23aiv: assumes A1: $\forall \mathrm{x}. \varphi(\mathrm{x}) \longrightarrow \psi$
shows $(\exists \mathrm{x} . \varphi(\mathrm{x})) \longrightarrow \psi$
using assms by auto

lemma MMI˙eqeltrrd: assumes A1: $\varphi \longrightarrow \mathrm{A} = \mathrm{B}$ and
A2: $\varphi \longrightarrow \mathrm{A} \in \mathrm{C}$
shows $\varphi \longrightarrow \mathrm{B} \in \mathrm{C}$
using assms by auto

lemma MMI˙syl2an: assumes A1: $(\varphi \wedge \psi) \longrightarrow \mathrm{ch}$ and
A2: $\vartheta \longrightarrow \varphi$ and
A3: $\tau \longrightarrow \psi$
shows $(\vartheta \wedge \tau) \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙adantrl: assumes A1: $(\varphi \wedge \psi) \longrightarrow \mathrm{ch}$
shows $(\varphi \wedge (\vartheta \wedge \psi)) \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙ad2ant2r: assumes A1: $(\varphi \wedge \psi) \longrightarrow \mathrm{ch}$
shows $((\varphi \wedge \vartheta) \wedge (\psi \wedge \tau)) \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙adantll: assumes A1: $(\varphi \wedge \psi) \longrightarrow \mathrm{ch}$
shows $((\vartheta \wedge \varphi) \wedge \psi) \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙anandirs: assumes A1: $((\varphi \wedge \mathrm{ch}) \wedge (\psi \wedge \mathrm{ch})) \longrightarrow \tau$
shows $((\varphi \wedge \psi) \wedge \mathrm{ch}) \longrightarrow \tau$
using assms by auto

lemma MMI˙adantlr: assumes A1: $(\varphi \wedge \psi) \longrightarrow \mathrm{ch}$
shows $((\varphi \wedge \vartheta) \wedge \psi) \longrightarrow \mathrm{ch}$

using assms by auto

lemma MMI˙an42s: assumes A1: $((\varphi \wedge \psi) \wedge (\mathrm{ch} \wedge \vartheta)) \longrightarrow \tau$
shows $((\varphi \wedge \mathrm{ch}) \wedge (\vartheta \wedge \psi)) \longrightarrow \tau$
using assms by auto

lemma MMI˙mp3an2: assumes A1: ψ and
A2: $(\varphi \wedge \psi \wedge \mathrm{ch}) \longrightarrow \vartheta$
shows $(\varphi \wedge \mathrm{ch}) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙3simp1:
shows $(\varphi \wedge \psi \wedge \mathrm{ch}) \longrightarrow \varphi$
by auto

lemma MMI˙3impb: assumes A1: $(\varphi \wedge (\psi \wedge \mathrm{ch})) \longrightarrow \vartheta$
shows $(\varphi \wedge \psi \wedge \mathrm{ch}) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙mpbird: assumes Amin: $\varphi \longrightarrow \mathrm{ch}$ and
Amaj: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$
shows $\varphi \longrightarrow \psi$
using assms by auto

lemma (in MMIsar0) MMI˙opreq12i: assumes A1: A = B and
A2: C = D
shows
(A + C) = (B + D)
(A · C) = (B · D)
(A - C) = (B - D)
using assms by auto

lemma MMI˙3eqtr4: assumes A1: A = B and
A2: C = A and
A3: D = B
shows C = D
using assms by auto

lemma MMI˙eqtr4d: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\varphi \longrightarrow$ C = B
shows $\varphi \longrightarrow$ A = C
using assms by auto

lemma MMI˙3eqtr3rd: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\varphi \longrightarrow$ A = C and
A3: $\varphi \longrightarrow$ B = D
shows $\varphi \longrightarrow$ D = C
using assms by auto

lemma MMI˙sylanc: assumes A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch and
A2: $\vartheta \longrightarrow \varphi$ and
A3: $\vartheta \longrightarrow \psi$
shows $\vartheta \longrightarrow$ ch
using assms by auto

lemma MMI˙anim12i: assumes A1: $\varphi \longrightarrow \psi$ and
A2: ch $\longrightarrow \vartheta$
shows ($\varphi \wedge$ ch) $\longrightarrow$ ($\psi \wedge \vartheta$)
using assms by auto

lemma (in MMIsar0) MMI˙opreqan12d: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\psi \longrightarrow$ C = D
shows
($\varphi \wedge \psi$) $\longrightarrow$ (A + C) = (B + D)
($\varphi \wedge \psi$) $\longrightarrow$ (A - C) = (B - D)
($\varphi \wedge \psi$) $\longrightarrow$ (A $\cdot$ C) = (B $\cdot$ D)
using assms by auto

lemma MMI˙sylanr2: assumes A1: ($\varphi \wedge$ ($\psi \wedge$ ch)) $\longrightarrow \vartheta$ and
A2: $\tau \longrightarrow$ ch
shows ($\varphi \wedge$ ($\psi \wedge \tau$)) $\longrightarrow \vartheta$
using assms by auto

lemma MMI˙sylanl2: assumes A1: (($\varphi \wedge \psi$) $\wedge$ ch) $\longrightarrow \vartheta$ and
A2: $\tau \longrightarrow \psi$
shows (($\varphi \wedge \tau$) $\wedge$ ch) $\longrightarrow \vartheta$
using assms by auto

lemma MMI˙ancom2s: assumes A1: ($\varphi \wedge$ ($\psi \wedge$ ch)) $\longrightarrow \vartheta$
shows ($\varphi \wedge$ (ch $\wedge \psi$)) $\longrightarrow \vartheta$
using assms by auto

lemma MMI˙anandis: assumes A1: (($\varphi \wedge \psi$) $\wedge$ ($\varphi \wedge$ ch)) $\longrightarrow \tau$
shows ($\varphi \wedge$ ($\psi \wedge$ ch)) $\longrightarrow \tau$
using assms by auto

lemma MMI˙sylan9eq: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\psi \longrightarrow$ B = C

shows ($\varphi \wedge \psi$) $\longrightarrow$ A = C
using assms by auto

lemma MMI˙keephyp: assumes A1: A = if (φ , A , B) $\longrightarrow$ ($\psi \longleftrightarrow \vartheta$) and
A2: B = if (φ , A , B) $\longrightarrow$ (ch $\longleftrightarrow \vartheta$) and
A3: ψ and
A4: ch
shows ϑ
proof -
{ assume φ
with A1 A3 have ϑ by simp }
moreover
{ assume $\neg\varphi$
with A2 A4 have ϑ by simp }
ultimately show ϑ by auto
qed

lemma MMI˙eleq1:
shows A = B $\longrightarrow$ (A $\in$ C $\longleftrightarrow$ B $\in$ C)
by auto

lemma MMI˙pm4˙2i:
shows $\varphi \longrightarrow$ ($\psi \longleftrightarrow \psi$)
by auto

lemma MMI˙3anbi123d: assumes A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) and
A2: $\varphi \longrightarrow$ ($\vartheta \longleftrightarrow \tau$) and
A3: $\varphi \longrightarrow$ ($\eta \longleftrightarrow \zeta$)
shows $\varphi \longrightarrow$ (($\psi \wedge \vartheta \wedge \eta$) $\longleftrightarrow$ (ch $\wedge \tau \wedge \zeta$))
using assms by auto

lemma MMI˙imbi12d: assumes A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) and
A2: $\varphi \longrightarrow$ ($\vartheta \longleftrightarrow \tau$)
shows $\varphi \longrightarrow$ (($\psi \longrightarrow \vartheta$) $\longleftrightarrow$ (ch $\longrightarrow \tau$))
using assms by auto

lemma MMI˙bitrd: assumes A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) and
A2: $\varphi \longrightarrow$ (ch $\longleftrightarrow \vartheta$)
shows $\varphi \longrightarrow$ ($\psi \longleftrightarrow \vartheta$)
using assms by auto

lemma MMI˙df˙ne:
shows (A $\neq$ B $\longleftrightarrow \neg$ (A = B))
by auto

lemma MMI˙3pm3˙2i: assumes A1: φ and
A2: ψ and

A3: ch
shows $\varphi \wedge \psi \wedge$ ch
using assms by auto

lemma MMI˙eqeq2i: assumes A1: A = B
shows C = A $\longleftrightarrow$ C = B
using assms by auto

lemma MMI˙syl5bbr: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch) and
A2: $\psi \longleftrightarrow \vartheta$
shows $\varphi \longrightarrow (\vartheta \longleftrightarrow$ ch)
using assms by auto

lemma MMI˙biimpd: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow (\psi \longrightarrow$ ch)
using assms by auto

lemma MMI˙orrd: assumes A1: $\varphi \longrightarrow (\neg (\psi) \longrightarrow$ ch)
shows $\varphi \longrightarrow (\psi \vee$ ch)
using assms by auto

lemma MMI˙jaoi: assumes A1: $\varphi \longrightarrow \psi$ and
A2: ch $\longrightarrow \psi$
shows $(\varphi \vee$ ch $) \longrightarrow \psi$
using assms by auto

lemma MMI˙oridm:
shows $(\varphi \vee \varphi) \longleftrightarrow \varphi$
by auto

lemma MMI˙orbi1d: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow ((\psi \vee \vartheta) \longleftrightarrow (\text{ch} \vee \vartheta))$
using assms by auto

lemma MMI˙orbi2d: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow ((\vartheta \vee \psi) \longleftrightarrow (\vartheta \vee \text{ch}))$
using assms by auto

lemma MMI˙3bitr4g: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch) and
A2: $\vartheta \longleftrightarrow \psi$ and
A3: $\tau \longleftrightarrow$ ch
shows $\varphi \longrightarrow (\vartheta \longleftrightarrow \tau)$
using assms by auto

lemma MMI˙negbid: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow (\neg (\psi) \longleftrightarrow \neg (\text{ch}))$
using assms by auto

lemma MMI˙ioran:
shows $\neg$ (($\varphi \vee \psi$)) $\longleftrightarrow$
($\neg$ (φ) $\wedge$ $\neg$ (ψ))
by auto

lemma MMI˙syl6rbb: assumes A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) and
A2: ch $\longleftrightarrow \vartheta$
shows $\varphi \longrightarrow$ ($\vartheta \longleftrightarrow \psi$)
using assms by auto

lemma MMI˙anbi12i: assumes A1: $\varphi \longleftrightarrow \psi$ and
A2: ch $\longleftrightarrow \vartheta$
shows ($\varphi \wedge$ ch) $\longleftrightarrow$ ($\psi \wedge \vartheta$)
using assms by auto

lemma MMI˙keepel: assumes A1: A $\in$ C and
A2: B $\in$ C
shows if (φ , A , B) $\in$ C
using assms by auto

lemma MMI˙imbi2d: assumes A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow$ (($\vartheta \longrightarrow \psi$) $\longleftrightarrow$ ($\vartheta \longrightarrow$ ch))
using assms by auto

lemma MMI˙eqeltr: assumes A = B and B $\in$ C
shows A $\in$ C using assms by auto

lemma MMI˙3impia: assumes A1: ($\varphi \wedge \psi$) $\longrightarrow$ (ch $\longrightarrow \vartheta$)
shows ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
using assms by auto

lemma MMI˙eqneqd: assumes A1: $\varphi \longrightarrow$ (A = B $\longleftrightarrow$ C = D)
shows $\varphi \longrightarrow$ (A $\neq$ B $\longleftrightarrow$ C $\neq$ D)
using assms by auto

lemma MMI˙3ad2ant2: assumes A1: $\varphi \longrightarrow$ ch
shows ($\psi \wedge \varphi \wedge \vartheta$) $\longrightarrow$ ch
using assms by auto

lemma MMI˙mp3anl3: assumes A1: ch and
A2: $((\varphi \wedge \psi \wedge \mathrm{ch}) \wedge \vartheta) \longrightarrow \tau$
shows $((\varphi \wedge \psi) \wedge \vartheta) \longrightarrow \tau$
using assms by auto

lemma MMI˙bitr4d: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$ and
A2: $\varphi \longrightarrow (\vartheta \longleftrightarrow \mathrm{ch})$
shows $\varphi \longrightarrow (\psi \longleftrightarrow \vartheta)$
using assms by auto

lemma MMI˙neeq1d: assumes A1: $\varphi \longrightarrow \mathrm{A} = \mathrm{B}$
shows $\varphi \longrightarrow (\mathrm{A} \neq \mathrm{C} \longleftrightarrow \mathrm{B} \neq \mathrm{C})$
using assms by auto

lemma MMI˙3anim123i: assumes A1: $\varphi \longrightarrow \psi$ and
A2: $\mathrm{ch} \longrightarrow \vartheta$ and
A3: $\tau \longrightarrow \eta$
shows $(\varphi \wedge \mathrm{ch} \wedge \tau) \longrightarrow (\psi \wedge \vartheta \wedge \eta)$
using assms by auto

lemma MMI˙3exp: assumes A1: $(\varphi \wedge \psi \wedge \mathrm{ch}) \longrightarrow \vartheta$
shows $\varphi \longrightarrow (\psi \longrightarrow (\mathrm{ch} \longrightarrow \vartheta))$
using assms by auto

lemma MMI˙exp4a: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow ((\mathrm{ch} \wedge \vartheta) \longrightarrow \tau))$
shows $\varphi \longrightarrow (\psi \longrightarrow (\mathrm{ch} \longrightarrow (\vartheta \longrightarrow \tau)))$
using assms by auto

lemma MMI˙3imp1: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow (\mathrm{ch} \longrightarrow (\vartheta \longrightarrow \tau)))$
shows $((\varphi \wedge \psi \wedge \mathrm{ch}) \wedge \vartheta) \longrightarrow \tau$
using assms by auto

lemma MMI˙anim1i: assumes A1: $\varphi \longrightarrow \psi$
shows $(\varphi \wedge \mathrm{ch}) \longrightarrow (\psi \wedge \mathrm{ch})$
using assms by auto

lemma MMI˙3adantl1: assumes A1: $((\varphi \wedge \psi) \wedge \mathrm{ch}) \longrightarrow \vartheta$
shows $((\tau \wedge \varphi \wedge \psi) \wedge \mathrm{ch}) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙3adantl2: assumes A1: $((\varphi \wedge \psi) \wedge \mathrm{ch}) \longrightarrow \vartheta$
shows $((\varphi \wedge \tau \wedge \psi) \wedge \mathrm{ch}) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙3comr: assumes A1: $(\varphi \wedge \psi \wedge \mathrm{ch}) \longrightarrow \vartheta$
shows $(\mathrm{ch} \wedge \varphi \wedge \psi) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙bitr3: assumes A1: $\psi \longleftrightarrow \varphi$ and
A2: $\psi \longleftrightarrow$ ch
shows $\varphi \longleftrightarrow$ ch
using assms by auto

lemma MMI˙anbi12d: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \text{ch})$ and
A2: $\varphi \longrightarrow (\vartheta \longleftrightarrow \tau)$
shows $\varphi \longrightarrow ((\psi \wedge \vartheta) \longleftrightarrow (\text{ch} \wedge \tau))$
using assms by auto

lemma MMI˙pm3˙26i: assumes A1: $\varphi \wedge \psi$
shows φ
using assms by auto

lemma MMI˙pm3˙27i: assumes A1: $\varphi \wedge \psi$
shows ψ
using assms by auto

lemma MMI˙anabsan: assumes A1: $((\varphi \wedge \varphi) \wedge \psi) \longrightarrow \text{ch}$
shows $(\varphi \wedge \psi) \longrightarrow \text{ch}$
using assms by auto

lemma MMI˙3eqtr4rd: assumes A1: $\varphi \longrightarrow \text{A} = \text{B}$ and
A2: $\varphi \longrightarrow \text{C} = \text{A}$ and
A3: $\varphi \longrightarrow \text{D} = \text{B}$
shows $\varphi \longrightarrow \text{D} = \text{C}$
using assms by auto

lemma MMI˙syl3an1: assumes A1: $(\varphi \wedge \psi \wedge \text{ch}) \longrightarrow \vartheta$ and
A2: $\tau \longrightarrow \varphi$
shows $(\tau \wedge \psi \wedge \text{ch}) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙syl3anl2: assumes A1: $((\varphi \wedge \psi \wedge \text{ch}) \wedge \vartheta) \longrightarrow \tau$ and
A2: $\eta \longrightarrow \psi$
shows $((\varphi \wedge \eta \wedge \text{ch}) \wedge \vartheta) \longrightarrow \tau$
using assms by auto

lemma MMI˙jca: assumes A1: $\varphi \longrightarrow \psi$ and
A2: $\varphi \longrightarrow \text{ch}$
shows $\varphi \longrightarrow (\psi \wedge \text{ch})$

using assms by auto

lemma MMI˙3ad2ant3: assumes A1: $\varphi \longrightarrow$ ch
shows ($\psi \wedge \vartheta \wedge \varphi$) $\longrightarrow$ ch
using assms by auto

lemma MMI˙anim2i: assumes A1: $\varphi \longrightarrow \psi$
shows (ch $\wedge \varphi$) $\longrightarrow$ (ch $\wedge \psi$)
using assms by auto

lemma MMI˙ancom:
shows ($\varphi \wedge \psi$) $\longleftrightarrow$ ($\psi \wedge \varphi$)
by auto

lemma MMI˙anbi1i: assumes Aaa: $\varphi \longleftrightarrow \psi$
shows ($\varphi \wedge$ ch) $\longleftrightarrow$ ($\psi \wedge$ ch)
using assms by auto

lemma MMI˙an42:
shows (($\varphi \wedge \psi$) $\wedge$ (ch $\wedge \vartheta$)) $\longleftrightarrow$
(($\varphi \wedge$ ch) $\wedge$ ($\vartheta \wedge \psi$))
by auto

lemma MMI˙sylanb: assumes A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch and
A2: $\vartheta \longleftrightarrow \varphi$
shows ($\vartheta \wedge \psi$) $\longrightarrow$ ch
using assms by auto

lemma MMI˙an4:
shows (($\varphi \wedge \psi$) $\wedge$ (ch $\wedge \vartheta$)) $\longleftrightarrow$
(($\varphi \wedge$ ch) $\wedge$ ($\psi \wedge \vartheta$))
by auto

lemma MMI˙syl2anb: assumes A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch and
A2: $\vartheta \longleftrightarrow \varphi$ and
A3: $\tau \longleftrightarrow \psi$
shows ($\vartheta \wedge \tau$) $\longrightarrow$ ch
using assms by auto

lemma MMI˙eqtr2d: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\varphi \longrightarrow$ B = C
shows $\varphi \longrightarrow$ C = A
using assms by auto

lemma MMI˙sylbid: assumes A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) and
A2: $\varphi \longrightarrow$ (ch $\longrightarrow \vartheta$)
shows $\varphi \longrightarrow$ ($\psi \longrightarrow \vartheta$)
using assms by auto

lemma MMI˙sylanl1: assumes A1: $((\varphi \wedge \psi) \wedge \mathrm{ch}) \longrightarrow \vartheta$ and
A2: $\tau \longrightarrow \varphi$
shows $((\tau \wedge \psi) \wedge \mathrm{ch}) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙sylan2b: assumes A1: $(\varphi \wedge \psi) \longrightarrow \mathrm{ch}$ and
A2: $\vartheta \longleftrightarrow \psi$
shows $(\varphi \wedge \vartheta) \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙pm3˙22:
shows $(\varphi \wedge \psi) \longrightarrow (\psi \wedge \varphi)$
by auto

lemma MMI˙ancli: assumes A1: $\varphi \longrightarrow \psi$
shows $\varphi \longrightarrow (\varphi \wedge \psi)$
using assms by auto

lemma MMI˙ad2antlr: assumes A1: $\varphi \longrightarrow \psi$
shows $((\mathrm{ch} \wedge \varphi) \wedge \vartheta) \longrightarrow \psi$
using assms by auto

lemma MMI˙biimpa: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$
shows $(\varphi \wedge \psi) \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙sylan2i: assumes A1: $\varphi \longrightarrow ((\psi \wedge \mathrm{ch}) \longrightarrow \vartheta)$ and
A2: $\tau \longrightarrow \mathrm{ch}$
shows $\varphi \longrightarrow ((\psi \wedge \tau) \longrightarrow \vartheta)$
using assms by auto

lemma MMI˙3jca: assumes A1: $\varphi \longrightarrow \psi$ and
A2: $\varphi \longrightarrow \mathrm{ch}$ and
A3: $\varphi \longrightarrow \vartheta$
shows $\varphi \longrightarrow (\psi \wedge \mathrm{ch} \wedge \vartheta)$
using assms by auto

lemma MMI˙com34: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow (\mathrm{ch} \longrightarrow (\vartheta \longrightarrow \tau)))$
shows $\varphi \longrightarrow (\psi \longrightarrow (\vartheta \longrightarrow (\mathrm{ch} \longrightarrow \tau)))$
using assms by auto

lemma MMI˙imp43: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow (\mathrm{ch} \longrightarrow (\vartheta \longrightarrow \tau)))$
shows $((\varphi \wedge \psi) \wedge (\mathrm{ch} \wedge \vartheta)) \longrightarrow \tau$
using assms by auto

lemma MMI˙3anass:
shows $(\varphi \wedge \psi \wedge \mathrm{ch}) \longleftrightarrow (\varphi \wedge (\psi \wedge \mathrm{ch}))$
by auto

lemma MMI˙3eqtr4r: assumes A1: A = B and
A2: C = A and
A3: D = B
shows D = C
using assms by auto

lemma MMI˙jctl: assumes A1: ψ
shows $\varphi \longrightarrow (\psi \wedge \varphi)$
using assms by auto

lemma MMI˙sylibr: assumes A1: $\varphi \longrightarrow \psi$ and
A2: ch $\longleftrightarrow \psi$
shows $\varphi \longrightarrow$ ch
using assms by auto

lemma MMI˙mpanl1: assumes A1: φ and
A2: $((\varphi \wedge \psi) \wedge$ ch $) \longrightarrow \vartheta$
shows $(\psi \wedge$ ch $) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙a1i: assumes A1: φ
shows $\psi \longrightarrow \varphi$
using assms by auto

lemma (in MMIsar0) MMI˙opreqan12rd: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\psi \longrightarrow$ C = D
shows
$(\psi \wedge \varphi) \longrightarrow (A + C) = (B + D)$
$(\psi \wedge \varphi) \longrightarrow (A \cdot C) = (B \cdot D)$
$(\psi \wedge \varphi) \longrightarrow (A - C) = (B - D)$
$(\psi \wedge \varphi) \longrightarrow (A / C) = (B / D)$
using assms by auto

lemma MMI˙3adantl3: assumes A1: $((\varphi \wedge \psi) \wedge$ ch $) \longrightarrow \vartheta$
shows $((\varphi \wedge \psi \wedge \tau) \wedge$ ch $) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙sylbi: assumes A1: $\varphi \longleftrightarrow \psi$ and
A2: $\psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
using assms by auto

lemma MMI˙eirr:
shows $\neg (A \in A)$
by (rule mem˙not˙refl)

lemma MMI˙eleq1i: assumes A1: A = B
shows A ∈ C ⟷ B ∈ C
using assms by auto

lemma MMI˙mtbir: assumes A1: ¬ (ψ) and
A2: φ ⟷ ψ
shows ¬ (φ)
using assms by auto

lemma MMI˙mto: assumes A1: ¬ (ψ) and
A2: φ ⟶ ψ
shows ¬ (φ)
using assms by auto

lemma MMI˙df˙nel:
shows (A ∉ B ⟷ ¬ (A ∈ B))
by auto

lemma MMI˙snid: assumes A1: A isASet
shows A ∈ – A ″
using assms by auto

lemma MMI˙en2lp:
shows ¬ (A ∈ B ∧ B ∈ A)
proof
assume A1: A ∈ B ∧ B ∈ A
then have A ∈ B by simp
moreover
{ assume ¬ (¬ (A ∈ B ∧ B ∈ A))
then have B∈A by auto}
ultimately have ¬(A ∈ B ∧ B ∈ A)
by (rule mem˙asym)
with A1 show False by simp
qed

lemma MMI˙imnan:
shows (φ ⟶ ¬ (ψ)) ⟷ ¬ ((φ ∧ ψ))
by auto

lemma MMI˙sseqtr4: assumes A1: A ⊆ B and
A2: C = B
shows A ⊆ C
using assms by auto

lemma MMI˙ssun1:
shows A ⊆ (A ∪ B)

by auto

lemma MMI˙ibar:
shows $\varphi \longrightarrow (\psi \longleftrightarrow (\varphi \wedge \psi))$
by auto

lemma MMI˙mtbiri: assumes Amin: $\neg (\mathrm{ch})$ and
Amaj: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$
shows $\varphi \longrightarrow \neg (\psi)$
using assms by auto

lemma MMI˙con2i: assumes Aa: $\varphi \longrightarrow \neg (\psi)$
shows $\psi \longrightarrow \neg (\varphi)$
using assms by auto

lemma MMI˙intnand: assumes A1: $\varphi \longrightarrow \neg (\psi)$
shows $\varphi \longrightarrow \neg ((\mathrm{ch} \wedge \psi))$
using assms by auto

lemma MMI˙intnanrd: assumes A1: $\varphi \longrightarrow \neg (\psi)$
shows $\varphi \longrightarrow \neg ((\psi \wedge \mathrm{ch}))$
using assms by auto

lemma MMI˙biorf:
shows $\neg (\varphi) \longrightarrow (\psi \longleftrightarrow (\varphi \vee \psi))$
by auto

lemma MMI˙bitr2d: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$ and
A2: $\varphi \longrightarrow (\mathrm{ch} \longleftrightarrow \vartheta)$
shows $\varphi \longrightarrow (\vartheta \longleftrightarrow \psi)$
using assms by auto

lemma MMI˙orass:
shows $((\varphi \vee \psi) \vee \mathrm{ch}) \longleftrightarrow (\varphi \vee (\psi \vee \mathrm{ch}))$
by auto

lemma MMI˙orcom:
shows $(\varphi \vee \psi) \longleftrightarrow (\psi \vee \varphi)$
by auto

lemma MMI˙3bitr4d: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$ and
A2: $\varphi \longrightarrow (\vartheta \longleftrightarrow \psi)$ and
A3: $\varphi \longrightarrow (\tau \longleftrightarrow \mathrm{ch})$
shows $\varphi \longrightarrow (\vartheta \longleftrightarrow \tau)$
using assms by auto

lemma MMI˙3imtr4d: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow \mathrm{ch})$ and
A2: $\varphi \longrightarrow (\vartheta \longleftrightarrow \psi)$ and
A3: $\varphi \longrightarrow (\tau \longleftrightarrow \mathrm{ch})$
shows $\varphi \longrightarrow (\vartheta \longrightarrow \tau)$
using assms by auto

lemma MMI˙3impdi: assumes A1: $((\varphi \wedge \psi) \wedge (\varphi \wedge \mathrm{ch})) \longrightarrow \vartheta$
shows $(\varphi \wedge \psi \wedge \mathrm{ch}) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙bi2anan9: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$ and
A2: $\vartheta \longrightarrow (\tau \longleftrightarrow \eta)$
shows $(\varphi \wedge \vartheta) \longrightarrow ((\psi \wedge \tau) \longleftrightarrow (\mathrm{ch} \wedge \eta))$
using assms by auto

lemma MMI˙ssel2:
shows $((\mathrm{A} \subseteq \mathrm{B} \wedge \mathrm{C} \in \mathrm{A}) \longrightarrow \mathrm{C} \in \mathrm{B})$
by auto

lemma MMI˙an1rs: assumes A1: $((\varphi \wedge \psi) \wedge \mathrm{ch}) \longrightarrow \vartheta$
shows $((\varphi \wedge \mathrm{ch}) \wedge \psi) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙ralbidva: assumes A1: $\forall \mathrm{x}. (\varphi \wedge \mathrm{x} \in \mathrm{A}) \longrightarrow (\psi(\mathrm{x}) \longleftrightarrow \mathrm{ch}(\mathrm{x}))$
shows $\varphi \longrightarrow ((\forall \mathrm{x} \in \mathrm{A} . \psi(\mathrm{x})) \longleftrightarrow (\forall \mathrm{x} \in \mathrm{A} . \mathrm{ch}(\mathrm{x})))$
using assms by auto

lemma MMI˙rexbidva: assumes A1: $\forall \mathrm{x}. (\varphi \wedge \mathrm{x} \in \mathrm{A}) \longrightarrow (\psi(\mathrm{x}) \longleftrightarrow \mathrm{ch}(\mathrm{x}))$
shows $\varphi \longrightarrow ((\exists \mathrm{x} \in \mathrm{A} . \psi(\mathrm{x})) \longleftrightarrow (\exists \mathrm{x} \in \mathrm{A} . \mathrm{ch}(\mathrm{x})))$
using assms by auto

lemma MMI˙con2bid: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \neg (\mathrm{ch}))$
shows $\varphi \longrightarrow (\mathrm{ch} \longleftrightarrow \neg (\psi))$
using assms by auto

lemma MMI˙so: assumes
A1: $\forall \mathrm{x}\ \mathrm{y}\ \mathrm{z}. (\mathrm{x} \in \mathrm{A} \wedge \mathrm{y} \in \mathrm{A} \wedge \mathrm{z} \in \mathrm{A}) \longrightarrow$
$((\langle \mathrm{x},\mathrm{y} \rangle \in \mathrm{R} \longleftrightarrow \neg ((\mathrm{x} = \mathrm{y} \vee \langle \mathrm{y}, \mathrm{x} \rangle \in \mathrm{R}))) \wedge$
$((\langle \mathrm{x}, \mathrm{y} \rangle \in \mathrm{R} \wedge \langle \mathrm{y}, \mathrm{z} \rangle \in \mathrm{R}) \longrightarrow \langle \mathrm{x}, \mathrm{z} \rangle \in \mathrm{R}))$
shows R Orders A
using assms StrictOrder˙def by auto

lemma MMI˙con1bid: assumes A1: $\varphi \longrightarrow (\neg (\psi) \longleftrightarrow$ ch)
shows $\varphi \longrightarrow (\neg (ch) \longleftrightarrow \psi)$
using assms by auto

lemma MMI˙sotrieq:
shows ((R Orders A) $\wedge$ (B $\in$ A $\wedge$ C $\in$ A)) $\longrightarrow$
(B = C $\longleftrightarrow \neg$ (($\langle$B,C$\rangle \in$ R $\vee$ $\langle$C, B$\rangle \in$ R)))
proof -
{ assume A1: R Orders A and A2: B $\in$ A $\wedge$ C $\in$ A
from A1 have $\forall$x y z. (x$\in$A $\wedge$ y$\in$A $\wedge$ z$\in$A) $\longrightarrow$
($\langle$x,y$\rangle \in$ R $\longleftrightarrow \neg$(x=y $\vee$ $\langle$y,x$\rangle \in$ R)) $\wedge$
($\langle$x,y$\rangle \in$ R $\wedge$ $\langle$y,z$\rangle \in$ R $\longrightarrow$ $\langle$x,z$\rangle \in$ R)
by (unfold StrictOrder˙def)
then have
$\forall$x y. x$\in$A $\wedge$ y$\in$A $\longrightarrow$ ($\langle$x,y$\rangle \in$ R $\longleftrightarrow \neg$(x=y $\vee$ $\langle$y,x$\rangle \in$ R))
by auto
with A2 have I: $\langle$B,C$\rangle \in$ R $\longleftrightarrow \neg$(B=C $\vee$ $\langle$C,B$\rangle \in$ R)
by blast
then have B = C $\longleftrightarrow \neg$ ($\langle$B,C$\rangle \in$ R $\vee$ $\langle$C, B$\rangle \in$ R)
by auto
} then show ((R Orders A) $\wedge$ (B $\in$ A $\wedge$ C $\in$ A)) $\longrightarrow$
(B = C $\longleftrightarrow \neg$ (($\langle$B,C$\rangle \in$ R $\vee$ $\langle$C, B$\rangle \in$ R))) by simp
qed

lemma MMI˙bicomd: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow (ch \longleftrightarrow \psi)$
using assms by auto

lemma MMI˙sotrieq2:
shows (R Orders A $\wedge$ (B $\in$ A $\wedge$ C $\in$ A)) $\longrightarrow$
(B = C $\longleftrightarrow$ ($\neg$ ($\langle$B, C$\rangle \in$ R) $\wedge \neg$ ($\langle$C, B$\rangle \in$ R)))
using MMI˙sotrieq by auto

lemma MMI˙orc:
shows $\varphi \longrightarrow (\varphi \vee \psi)$
by auto

lemma MMI˙syl6bbr: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch) and
A2: $\vartheta \longleftrightarrow$ ch
shows $\varphi \longrightarrow (\psi \longleftrightarrow \vartheta)$
using assms by auto

lemma MMI˙orbi1i: assumes A1: $\varphi \longleftrightarrow \psi$
shows ($\varphi \vee$ ch) $\longleftrightarrow$ ($\psi \vee$ ch)
using assms by auto

lemma MMI˙syl5rbbr: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch) and

A2: $\psi \longleftrightarrow \vartheta$
shows $\varphi \longrightarrow (\text{ch} \longleftrightarrow \vartheta)$
using assms by auto

lemma MMI˙anbi2d: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \text{ch})$
shows $\varphi \longrightarrow ((\vartheta \wedge \psi) \longleftrightarrow (\vartheta \wedge \text{ch}))$
using assms by auto

lemma MMI˙ord: assumes A1: $\varphi \longrightarrow (\psi \vee \text{ch})$
shows $\varphi \longrightarrow (\neg (\psi) \longrightarrow \text{ch})$
using assms by auto

lemma MMI˙impbid: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow \text{ch})$ and
A2: $\varphi \longrightarrow (\text{ch} \longrightarrow \psi)$
shows $\varphi \longrightarrow (\psi \longleftrightarrow \text{ch})$
using assms by blast

lemma MMI˙jcad: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow \text{ch})$ and
A2: $\varphi \longrightarrow (\psi \longrightarrow \vartheta)$
shows $\varphi \longrightarrow (\psi \longrightarrow (\text{ch} \wedge \vartheta))$
using assms by auto

lemma MMI˙ax˙1:
shows $\varphi \longrightarrow (\psi \longrightarrow \varphi)$
by auto

lemma MMI˙pm2˙24:
shows $\varphi \longrightarrow (\neg (\varphi) \longrightarrow \psi)$
by auto

lemma MMI˙imp3a: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow (\text{ch} \longrightarrow \vartheta))$
shows $\varphi \longrightarrow ((\psi \wedge \text{ch}) \longrightarrow \vartheta)$
using assms by auto

lemma (in MMIsar0) MMI˙breq1:
shows
$\text{A} = \text{B} \longrightarrow (\text{A} \leq \text{C} \longleftrightarrow \text{B} \leq \text{C})$
$\text{A} = \text{B} \longrightarrow (\text{A ¡ C} \longleftrightarrow \text{B ¡ C})$
by auto

lemma MMI˙biimprd: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \text{ch})$
shows $\varphi \longrightarrow (\text{ch} \longrightarrow \psi)$
using assms by auto

lemma MMI˙jaod: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow \text{ch})$ and
A2: $\varphi \longrightarrow (\vartheta \longrightarrow \text{ch})$
shows $\varphi \longrightarrow ((\psi \vee \vartheta) \longrightarrow \text{ch})$
using assms by auto

lemma MMI˙com23: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow (\mathrm{ch} \longrightarrow \vartheta))$
shows $\varphi \longrightarrow (\mathrm{ch} \longrightarrow (\psi \longrightarrow \vartheta))$
using assms by auto

lemma (in MMIsar0) MMI˙breq2:
shows
$\mathrm{A} = \mathrm{B} \longrightarrow (\mathrm{C} \leq \mathrm{A} \longleftrightarrow \mathrm{C} \leq \mathrm{B})$
$\mathrm{A} = \mathrm{B} \longrightarrow (\mathrm{C} \text{ ¡ } \mathrm{A} \longleftrightarrow \mathrm{C} \text{ ¡ } \mathrm{B})$
by auto

lemma MMI˙syld: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow \mathrm{ch})$ and
A2: $\varphi \longrightarrow (\mathrm{ch} \longrightarrow \vartheta)$
shows $\varphi \longrightarrow (\psi \longrightarrow \vartheta)$
using assms by auto

lemma MMI˙biimpcd: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$
shows $\psi \longrightarrow (\varphi \longrightarrow \mathrm{ch})$
using assms by auto

lemma MMI˙mp2and: assumes A1: $\varphi \longrightarrow \psi$ and
A2: $\varphi \longrightarrow \mathrm{ch}$ and
A3: $\varphi \longrightarrow ((\psi \wedge \mathrm{ch}) \longrightarrow \vartheta)$
shows $\varphi \longrightarrow \vartheta$
using assms by auto

lemma MMI˙sonr:
shows $(\mathrm{R} \text{ Orders } \mathrm{A} \wedge \mathrm{B} \in \mathrm{A}) \longrightarrow \neg (\langle \mathrm{B},\mathrm{B} \rangle \in \mathrm{R})$
unfolding StrictOrder˙def by auto

lemma MMI˙orri: assumes A1: $\neg (\varphi) \longrightarrow \psi$
shows $\varphi \vee \psi$
using assms by auto

lemma MMI˙mpbiri: assumes Amin: ch and
Amaj: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$
shows $\varphi \longrightarrow \psi$
using assms by auto

lemma MMI˙pm2˙46:
shows $\neg ((\varphi \vee \psi)) \longrightarrow \neg (\psi)$
by auto

lemma MMI˙elun:
shows $\mathrm{A} \in (\mathrm{B} \cup \mathrm{C}) \longleftrightarrow (\mathrm{A} \in \mathrm{B} \vee \mathrm{A} \in \mathrm{C})$
by auto

lemma (in MMIsar0) MMI˙pnfxr:

shows $+\infty \in \mathbb{R}^*$
using cxr˙def by simp

lemma MMI˙elisseti: assumes A1: $A \in B$
shows A isASet
using assms by auto

lemma (in MMIsar0) MMI˙mnfxr:
shows $-\infty \in \mathbb{R}^*$
using cxr˙def by simp

lemma MMI˙elpr2: assumes A1: B isASet and
A2: C isASet
shows $A \in - B , C '' \longleftrightarrow (A = B \vee A = C)$
using assms by auto

lemma MMI˙orbi2i: assumes A1: $\varphi \longleftrightarrow \psi$
shows $(ch \vee \varphi) \longleftrightarrow (ch \vee \psi)$
using assms by auto

lemma MMI˙3orass:
shows $(\varphi \vee \psi \vee ch) \longleftrightarrow (\varphi \vee (\psi \vee ch))$
by auto

lemma MMI˙bitr4: assumes A1: $\varphi \longleftrightarrow \psi$ and
A2: $ch \longleftrightarrow \psi$
shows $\varphi \longleftrightarrow ch$
using assms by auto

lemma MMI˙eleq2:
shows $A = B \longrightarrow (C \in A \longleftrightarrow C \in B)$
by auto

lemma MMI˙nelneq:
shows $(A \in C \wedge \neg (B \in C)) \longrightarrow \neg (A = B)$
by auto

lemma MMI˙df˙pr:
shows $- A , B '' = (- A '' \cup - B '')$
by auto

lemma MMI˙ineq2i: assumes A1: $A = B$
shows $(C \cap A) = (C \cap B)$
using assms by auto

lemma MMI˙mt2: assumes A1: ψ and
A2: $\varphi \longrightarrow \neg (\psi)$

shows $\neg$ (φ)
using assms by auto

lemma MMI˙disjsn:
shows (A $\cap$ − B ″) = 0 $\longleftrightarrow$ $\neg$ (B $\in$ A)
by auto

lemma MMI˙undisj2:
shows ((A $\cap$ B) =
0 $\wedge$ (A $\cap$ C) =
0) $\longleftrightarrow$ (A $\cap$ (B $\cup$ C)) = 0
by auto

lemma MMI˙disjssun:
shows ((A $\cap$ B) = 0 $\longrightarrow$ (A $\subseteq$ (B $\cup$ C) $\longleftrightarrow$ A $\subseteq$ C))
by auto

lemma MMI˙uncom:
shows (A $\cup$ B) = (B $\cup$ A)
by auto

lemma MMI˙sseq2i: assumes A1: A = B
shows (C $\subseteq$ A $\longleftrightarrow$ C $\subseteq$ B)
using assms by auto

lemma MMI˙disj:
shows (A $\cap$ B) =
0 $\longleftrightarrow$ ($\forall$ x $\in$ A . $\neg$ (x $\in$ B))
by auto

lemma MMI˙syl5ibr: assumes A1: φ $\longrightarrow$ (ψ $\longrightarrow$ ch) and
A2: ψ $\longleftrightarrow$ ϑ
shows φ $\longrightarrow$ (ϑ $\longrightarrow$ ch)
using assms by auto

lemma MMI˙con3d: assumes A1: φ $\longrightarrow$ (ψ $\longrightarrow$ ch)
shows φ $\longrightarrow$ ($\neg$ (ch) $\longrightarrow$ $\neg$ (ψ))
using assms by auto

lemma MMI˙dfrex2:
shows ($\exists$ x $\in$ A . φ(x)) $\longleftrightarrow$ $\neg$ (($\forall$ x $\in$ A . $\neg$ φ(x)))
by auto

lemma MMI˙visset:
shows x isASet
by auto

lemma MMI˙elpr: assumes A1: A isASet
shows A ∈ − B , C ″ ⟷ (A = B ∨ A = C)
using assms by auto

lemma MMI˙rexbii: assumes A1: ∀x. φ(x) ⟷ ψ(x)
shows (∃ x ∈ A . φ(x)) ⟷ (∃ x ∈ A . ψ(x))
using assms by auto

lemma MMI˙r19˙43:
shows (∃ x ∈ A . (φ(x) ∨ ψ(x))) ⟷
((∃ x ∈ A . φ(x) ∨ (∃ x ∈ A . ψ(x))))
by auto

lemma MMI˙exancom:
shows (∃ x . (φ(x) ∧ ψ(x))) ⟷
(∃ x . (ψ(x) ∧ φ(x)))
by auto

lemma MMI˙ceqsexv: assumes A1: A isASet and
A2: ∀x. x = A ⟶ (φ(x) ⟷ ψ(x))
shows (∃ x . (x = A ∧ φ(x))) ⟷ ψ(A)
using assms by auto

lemma MMI˙orbi12i˙orig: assumes A1: φ ⟷ ψ and
A2: ch ⟷ ϑ
shows (φ ∨ ch) ⟷ (ψ ∨ ϑ)
using assms by auto

lemma MMI˙orbi12i: assumes A1: (∃x. φ(x)) ⟷ ψ and
A2: (∃x. ch(x)) ⟷ ϑ
shows (∃x. φ(x)) ∨ (∃x. ch(x)) ⟷ (ψ ∨ ϑ)
using assms by auto

lemma MMI˙syl6ib: assumes A1: φ ⟶ (ψ ⟶ ch) and
A2: ch ⟷ ϑ
shows φ ⟶ (ψ ⟶ ϑ)
using assms by auto

lemma MMI˙intnan: assumes A1: ¬ (φ)
shows ¬ ((ψ ∧ φ))
using assms by auto

lemma MMI˙intnanr: assumes A1: ¬ (φ)
shows ¬ ((φ ∧ ψ))
using assms by auto

lemma MMI˙pm3˙2ni: assumes A1: ¬ (φ) and
A2: ¬ (ψ)
shows ¬ ((φ ∨ ψ))

using assms by auto

lemma (in MMIsar0) MMI˙breq12:
shows
(A = B ∧ C = D) ⟶ (A ¡ C ⟷ B ¡ D)
(A = B ∧ C = D) ⟶ (A ≤ C ⟷ B ≤ D)
by auto

lemma MMI˙necom:
shows A ≠ B ⟷ B ≠ A
by auto

lemma MMI˙3jaoi: assumes A1: φ ⟶ ψ and
A2: ch ⟶ ψ and
A3: ϑ ⟶ ψ
shows (φ ∨ ch ∨ ϑ) ⟶ ψ
using assms by auto

lemma MMI˙jctr: assumes A1: ψ
shows φ ⟶ (φ ∧ ψ)
using assms by auto

lemma MMI˙olc:
shows φ ⟶ (ψ ∨ φ)
by auto

lemma MMI˙3syl: assumes A1: φ ⟶ ψ and
A2: ψ ⟶ ch and
A3: ch ⟶ ϑ
shows φ ⟶ ϑ
using assms by auto

lemma MMI˙mtbird: assumes Amin: φ ⟶ ¬ (ch) and
Amaj: φ ⟶ (ψ ⟷ ch)
shows φ ⟶ ¬ (ψ)
using assms by auto

lemma MMI˙pm2˙21d: assumes A1: φ ⟶ ¬ (ψ)
shows φ ⟶ (ψ ⟶ ch)
using assms by auto

lemma MMI˙3jaodan: assumes A1: (φ ∧ ψ) ⟶ ch and
A2: (φ ∧ ϑ) ⟶ ch and
A3: (φ ∧ τ) ⟶ ch
shows (φ ∧ (ψ ∨ ϑ ∨ τ)) ⟶ ch
using assms by auto

lemma MMI˙sylan2br: assumes A1: $(\varphi \wedge \psi) \longrightarrow$ ch and
A2: $\psi \longleftrightarrow \vartheta$
shows $(\varphi \wedge \vartheta) \longrightarrow$ ch
using assms by auto

lemma MMI˙3jaoian: assumes A1: $(\varphi \wedge \psi) \longrightarrow$ ch and
A2: $(\vartheta \wedge \psi) \longrightarrow$ ch and
A3: $(\tau \wedge \psi) \longrightarrow$ ch
shows $((\varphi \vee \vartheta \vee \tau) \wedge \psi) \longrightarrow$ ch
using assms by auto

lemma MMI˙mtbid: assumes Amin: $\varphi \longrightarrow \neg (\psi)$ and
Amaj: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch $)$
shows $\varphi \longrightarrow \neg ($ ch $)$
using assms by auto

lemma MMI˙con1d: assumes A1: $\varphi \longrightarrow (\neg (\psi) \longrightarrow$ ch $)$
shows $\varphi \longrightarrow (\neg ($ ch $) \longrightarrow \psi)$
using assms by auto

lemma MMI˙pm2˙21nd: assumes A1: $\varphi \longrightarrow \psi$
shows $\varphi \longrightarrow (\neg (\psi) \longrightarrow$ ch $)$
using assms by auto

lemma MMI˙syl3an1b: assumes A1: $(\varphi \wedge \psi \wedge$ ch $) \longrightarrow \vartheta$ and
A2: $\tau \longleftrightarrow \varphi$
shows $(\tau \wedge \psi \wedge$ ch $) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙adantld: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow$ ch $)$
shows $\varphi \longrightarrow ((\vartheta \wedge \psi) \longrightarrow$ ch $)$
using assms by auto

lemma MMI˙adantrd: assumes A1: $\varphi \longrightarrow (\psi \longrightarrow$ ch $)$
shows $\varphi \longrightarrow ((\psi \wedge \vartheta) \longrightarrow$ ch $)$
using assms by auto

lemma MMI˙anasss: assumes A1: $((\varphi \wedge \psi) \wedge$ ch $) \longrightarrow \vartheta$
shows $(\varphi \wedge (\psi \wedge$ ch $)) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙syl3an3b: assumes A1: $(\varphi \wedge \psi \wedge$ ch $) \longrightarrow \vartheta$ and
A2: $\tau \longleftrightarrow$ ch
shows $(\varphi \wedge \psi \wedge \tau) \longrightarrow \vartheta$
using assms by auto

lemma MMI˙mpbid: assumes Amin: $\varphi \longrightarrow \psi$ and
Amaj: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$
shows $\varphi \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙orbi12d: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$ and
A2: $\varphi \longrightarrow (\vartheta \longleftrightarrow \tau)$
shows $\varphi \longrightarrow ((\psi \vee \vartheta) \longleftrightarrow (\mathrm{ch} \vee \tau))$
using assms by auto

lemma MMI˙ianor:
shows $\neg (\varphi \wedge \psi) \longleftrightarrow \neg \varphi \vee \neg \psi$
by auto

lemma MMI˙bitr2: assumes A1: $\varphi \longleftrightarrow \psi$ and
A2: $\psi \longleftrightarrow \mathrm{ch}$
shows $\mathrm{ch} \longleftrightarrow \varphi$
using assms by auto

lemma MMI˙biimp: assumes A1: $\varphi \longleftrightarrow \psi$
shows $\varphi \longrightarrow \psi$
using assms by auto

lemma MMI˙mpan2d: assumes A1: $\varphi \longrightarrow \mathrm{ch}$ and
A2: $\varphi \longrightarrow ((\psi \wedge \mathrm{ch}) \longrightarrow \vartheta)$
shows $\varphi \longrightarrow (\psi \longrightarrow \vartheta)$
using assms by auto

lemma MMI˙ad2antrr: assumes A1: $\varphi \longrightarrow \psi$
shows $((\varphi \wedge \mathrm{ch}) \wedge \vartheta) \longrightarrow \psi$
using assms by auto

lemma MMI˙biimpac: assumes A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$
shows $(\psi \wedge \varphi) \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙con2bii: assumes A1: $\varphi \longleftrightarrow \neg (\psi)$
shows $\psi \longleftrightarrow \neg (\varphi)$
using assms by auto

lemma MMI˙pm3˙26bd: assumes A1: $\varphi \longleftrightarrow (\psi \wedge \mathrm{ch})$
shows $\varphi \longrightarrow \psi$
using assms by auto

lemma MMI˙biimpr: assumes A1: $\varphi \longleftrightarrow \psi$

shows $\psi \longrightarrow \varphi$
using assms by auto

lemma (in MMIsar0) MMI˙3brtr3g: assumes A1: $\varphi \longrightarrow$ A ¡ B and
A2: A = C and
A3: B = D
shows $\varphi \longrightarrow$ C ¡ D
using assms by auto

lemma (in MMIsar0) MMI˙breq12i: assumes A1: A = B and
A2: C = D
shows
A ¡ C $\longleftrightarrow$ B ¡ D
A $\leq$ C $\longleftrightarrow$ B $\leq$ D
using assms by auto

lemma MMI˙negbii: assumes Aa: $\varphi \longleftrightarrow \psi$
shows $\neg\varphi \longleftrightarrow \neg\psi$
using assms by auto

lemma (in MMIsar0) MMI˙breq1i: assumes A1: A = B
shows
A ¡ C $\longleftrightarrow$ B ¡ C
A $\leq$ C $\longleftrightarrow$ B $\leq$ C
using assms by auto

lemma MMI˙syl5eqr: assumes A1: $\varphi \longrightarrow$ A = B and
A2: A = C
shows $\varphi \longrightarrow$ C = B
using assms by auto

lemma (in MMIsar0) MMI˙breq2d: assumes A1: $\varphi \longrightarrow$ A = B
shows
$\varphi \longrightarrow$ C ¡ A $\longleftrightarrow$ C ¡ B
$\varphi \longrightarrow$ C $\leq$ A $\longleftrightarrow$ C $\leq$ B
using assms by auto

lemma MMI˙ccase: assumes A1: $\varphi \wedge \psi \longrightarrow \tau$ and
A2: ch $\wedge \psi \longrightarrow \tau$ and
A3: $\varphi \wedge \vartheta \longrightarrow \tau$ and
A4: ch $\wedge \vartheta \longrightarrow \tau$
shows $(\varphi \vee$ ch$) \wedge (\psi \vee \vartheta) \longrightarrow \tau$
using assms by auto

lemma MMI˙pm3˙27bd: assumes A1: $\varphi \longleftrightarrow \psi \wedge$ ch

shows $\varphi \longrightarrow$ ch
using assms by auto

lemma MMI˙nsyl3: assumes A1: $\varphi \longrightarrow \neg\psi$ and
A2: ch $\longrightarrow \psi$
shows ch $\longrightarrow \neg\varphi$
using assms by auto

lemma MMI˙jctild: assumes A1: $\varphi \longrightarrow \psi \longrightarrow$ ch and
A2: $\varphi \longrightarrow \vartheta$
shows $\varphi \longrightarrow$
$\psi \longrightarrow \vartheta \wedge$ ch
using assms by auto

lemma MMI˙jctird: assumes A1: $\varphi \longrightarrow \psi \longrightarrow$ ch and
A2: $\varphi \longrightarrow \vartheta$
shows $\varphi \longrightarrow$
$\psi \longrightarrow$ ch $\wedge \vartheta$
using assms by auto

lemma MMI˙ccase2: assumes A1: $\varphi \wedge \psi \longrightarrow \tau$ and
A2: ch $\longrightarrow \tau$ and
A3: $\vartheta \longrightarrow \tau$
shows $(\varphi \vee$ ch$) \wedge (\psi \vee \vartheta) \longrightarrow \tau$
using assms by auto

lemma MMI˙3bitr3r: assumes A1: $\varphi \longleftrightarrow \psi$ and
A2: $\varphi \longleftrightarrow$ ch and
A3: $\psi \longleftrightarrow \vartheta$
shows $\vartheta \longleftrightarrow$ ch
using assms by auto

lemma (in MMIsar0) MMI˙syl6breq: assumes A1: $\varphi \longrightarrow$ A ¡ B and
A2: B = C
shows
$\varphi \longrightarrow$ A ¡ C
using assms by auto

lemma MMI˙pm2˙61i: assumes A1: $\varphi \longrightarrow \psi$ and
A2: $\neg\varphi \longrightarrow \psi$
shows ψ
using assms by auto

lemma MMI˙syl6req: assumes A1: $\varphi \longrightarrow$ A = B and
A2: B = C

shows $\varphi \longrightarrow$ C = A
using assms by auto

lemma MMI˙pm2˙61d: assumes A1: $\varphi \longrightarrow \psi \longrightarrow$ ch and
A2: $\varphi \longrightarrow$
$\neg\psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
using assms by auto

lemma MMI˙orim1d: assumes A1: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \vee \vartheta \longrightarrow$ ch $\vee \vartheta$
using assms by auto

lemma (in MMIsar0) MMI˙breq1d: assumes A1: $\varphi \longrightarrow$ A = B
shows
$\varphi \longrightarrow$ A ¡ C $\longleftrightarrow$ B ¡ C
$\varphi \longrightarrow$ A $\leq$ C $\longleftrightarrow$ B $\leq$ C
using assms by auto

lemma (in MMIsar0) MMI˙breq12d: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\varphi \longrightarrow$ C = D
shows
$\varphi \longrightarrow$ A ¡ C $\longleftrightarrow$ B ¡ D
$\varphi \longrightarrow$ A $\leq$ C $\longleftrightarrow$ B $\leq$ D
using assms by auto

lemma MMI˙bibi2d: assumes A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
$(\vartheta \longleftrightarrow \psi) \longleftrightarrow$
$\vartheta \longleftrightarrow$ ch
using assms by auto

lemma MMI˙con4bid: assumes A1: $\varphi \longrightarrow$
$\neg\psi \longleftrightarrow \neg$ch
shows $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
using assms by auto

lemma MMI˙3com13: assumes A1: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
shows ch $\wedge \psi \wedge \varphi \longrightarrow \vartheta$
using assms by auto

lemma MMI˙3bitr3rd: assumes A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch and
A2: $\varphi \longrightarrow$

$\psi \longleftrightarrow \vartheta$ and
A3: $\varphi \longrightarrow$
ch $\longleftrightarrow \tau$
shows $\varphi \longrightarrow$
$\tau \longleftrightarrow \vartheta$
using assms by auto

lemma MMI˙3imtr4g: assumes A1: $\varphi \longrightarrow \psi \longrightarrow$ ch and
A2: $\vartheta \longleftrightarrow \psi$ and
A3: $\tau \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
$\vartheta \longrightarrow \tau$
using assms by auto

lemma MMI˙expcom: assumes A1: $\varphi \wedge \psi \longrightarrow$ ch
shows $\psi \longrightarrow \varphi \longrightarrow$ ch
using assms by auto

lemma (in MMIsar0) MMI˙breq2i: assumes A1: A = B
shows
C ¡ A $\longleftrightarrow$ C ¡ B
C $\leq$ A $\longleftrightarrow$ C $\leq$ B
using assms by auto

lemma MMI˙3bitr2r: assumes A1: $\varphi \longleftrightarrow \psi$ and
A2: ch $\longleftrightarrow \psi$ and
A3: ch $\longleftrightarrow \vartheta$
shows $\vartheta \longleftrightarrow \varphi$
using assms by auto

lemma MMI˙dedth4h: assumes A1: A = if(φ, A, R) $\longrightarrow$
$\tau \longleftrightarrow \eta$ and
A2: B = if(ψ, B, S) $\longrightarrow$
$\eta \longleftrightarrow \zeta$ and
A3: C = if(ch, C, F) $\longrightarrow$
$\zeta \longleftrightarrow$ si and
A4: D = if(ϑ, D, G) $\longrightarrow$ si $\longleftrightarrow$ rh and
A5: rh
shows $(\varphi \wedge \psi) \wedge$ ch $\wedge \vartheta \longrightarrow \tau$
using assms by auto

lemma MMI˙anbi1d: assumes A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \wedge \vartheta \longleftrightarrow$ ch $\wedge \vartheta$
using assms by auto

lemma (in MMIsar0) MMI˙breqtrrd: assumes A1: $\varphi \longrightarrow$ A ¡ B and
A2: $\varphi \longrightarrow$ C = B
shows $\varphi \longrightarrow$ A ¡ C
using assms by auto

lemma MMI˙syl3an: assumes A1: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$ and
A2: $\tau \longrightarrow \varphi$ and
A3: $\eta \longrightarrow \psi$ and
A4: $\zeta \longrightarrow$ ch
shows $\tau \wedge \eta \wedge \zeta \longrightarrow \vartheta$
using assms by auto

lemma MMI˙3bitrd: assumes A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch and
A2: $\varphi \longrightarrow$
ch $\longleftrightarrow \vartheta$ and
A3: $\varphi \longrightarrow$
$\vartheta \longleftrightarrow \tau$
shows $\varphi \longrightarrow$
$\psi \longleftrightarrow \tau$
using assms by auto

lemma (in MMIsar0) MMI˙breqtr: assumes A1: A ¡ B and
A2: B = C
shows A ¡ C
using assms by auto

lemma MMI˙mpi: assumes A1: ψ and
A2: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
using assms by auto

lemma MMI˙eqtr2: assumes A1: A = B and
A2: B = C
shows C = A
using assms by auto

lemma MMI˙eqneqi: assumes A1: A = B $\longleftrightarrow$ C = D
shows A $\neq$ B $\longleftrightarrow$ C $\neq$ D
using assms by auto

lemma (in MMIsar0) MMI˙eqbrtrrd: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\varphi \longrightarrow$ A ¡ C
shows $\varphi \longrightarrow$ B ¡ C
using assms by auto

lemma MMI˙mpd: assumes A1: $\varphi \longrightarrow \psi$ and
A2: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
using assms by auto

lemma MMI˙mpdan: assumes A1: $\varphi \longrightarrow \psi$ and
A2: $\varphi \wedge \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
using assms by auto

lemma (in MMIsar0) MMI˙breqtrd: assumes A1: $\varphi \longrightarrow$ A ¡ B and
A2: $\varphi \longrightarrow$ B = C
shows $\varphi \longrightarrow$ A ¡ C
using assms by auto

lemma MMI˙mpand: assumes A1: $\varphi \longrightarrow \psi$ and
A2: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \longrightarrow$ ch $\longrightarrow \vartheta$
using assms by auto

lemma MMI˙imbi1d: assumes A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
$(\psi \longrightarrow \vartheta) \longleftrightarrow$
(ch $\longrightarrow \vartheta$)
using assms by auto

lemma MMI˙mtbii: assumes Amin: $\neg\psi$ and
Amaj: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow \neg$ch
using assms by auto

lemma MMI˙sylan2d: assumes A1: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$ and
A2: $\varphi \longrightarrow \tau \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \wedge \tau \longrightarrow \vartheta$
using assms by auto

lemma MMI˙imp32: assumes A1: $\varphi \longrightarrow$
$\psi \longrightarrow$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
using assms by auto

lemma (in MMIsar0) MMI˙breqan12d: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\psi \longrightarrow$ C = D
shows
$\varphi \wedge \psi \longrightarrow$ A ¡ C $\longleftrightarrow$ B ¡ D
$\varphi \wedge \psi \longrightarrow$ A $\leq$ C $\longleftrightarrow$ B $\leq$ D
using assms by auto

lemma MMI˙a1dd: assumes A1: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \longrightarrow \vartheta \longrightarrow$ ch
using assms by auto

lemma (in MMIsar0) MMI˙3brtr3d: assumes A1: $\varphi \longrightarrow$ A $\leq$ B and
A2: $\varphi \longrightarrow$ A = C and
A3: $\varphi \longrightarrow$ B = D
shows $\varphi \longrightarrow$ C $\leq$ D
using assms by auto

lemma MMI˙ad2antll: assumes A1: $\varphi \longrightarrow \psi$
shows ch $\wedge \vartheta \wedge \varphi \longrightarrow \psi$
using assms by auto

lemma MMI˙adantrrl: assumes A1: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge \psi \wedge \tau \wedge$ ch $\longrightarrow \vartheta$
using assms by auto

lemma MMI˙syl2ani: assumes A1: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$ and
A2: $\tau \longrightarrow \psi$ and
A3: $\eta \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\tau \wedge \eta \longrightarrow \vartheta$
using assms by auto

lemma MMI˙im2anan9: assumes A1: $\varphi \longrightarrow \psi \longrightarrow$ ch and
A2: $\vartheta \longrightarrow$
$\tau \longrightarrow \eta$
shows $\varphi \wedge \vartheta \longrightarrow$
$\psi \wedge \tau \longrightarrow$ ch $\wedge \eta$
using assms by auto

lemma MMI˙ancomsd: assumes A1: $\varphi \longrightarrow$

$\psi \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $\varphi \longrightarrow$
$\mathrm{ch} \wedge \psi \longrightarrow \vartheta$
using assms by auto

lemma MMI˙mpani: assumes A1: ψ and
A2: $\varphi \longrightarrow$
$\psi \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $\varphi \longrightarrow \mathrm{ch} \longrightarrow \vartheta$
using assms by auto

lemma MMI˙syldan: assumes A1: $\varphi \wedge \psi \longrightarrow \mathrm{ch}$ and
A2: $\varphi \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $\varphi \wedge \psi \longrightarrow \vartheta$
using assms by auto

lemma MMI˙mp3anl1: assumes A1: φ and
A2: $(\varphi \wedge \psi \wedge \mathrm{ch}) \wedge \vartheta \longrightarrow \tau$
shows $(\psi \wedge \mathrm{ch}) \wedge \vartheta \longrightarrow \tau$
using assms by auto

lemma MMI˙3ad2ant1: assumes A1: $\varphi \longrightarrow \mathrm{ch}$
shows $\varphi \wedge \psi \wedge \vartheta \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙pm3˙2:
shows $\varphi \longrightarrow$
$\psi \longrightarrow \varphi \wedge \psi$
by auto

lemma MMI˙pm2˙43i: assumes A1: $\varphi \longrightarrow$
$\varphi \longrightarrow \psi$
shows $\varphi \longrightarrow \psi$
using assms by auto

lemma MMI˙jctil: assumes A1: $\varphi \longrightarrow \psi$ and
A2: ch
shows $\varphi \longrightarrow \mathrm{ch} \wedge \psi$
using assms by auto

lemma MMI˙mpanl12: assumes A1: φ and
A2: ψ and
A3: $(\varphi \wedge \psi) \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $\mathrm{ch} \longrightarrow \vartheta$
using assms by auto

lemma MMI˙mpanr1: assumes A1: ψ and
A2: $\varphi \wedge \psi \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $\varphi \wedge \mathrm{ch} \longrightarrow \vartheta$
using assms by auto

lemma MMI˙ad2antrl: assumes A1: $\varphi \longrightarrow \psi$
shows $\mathrm{ch} \wedge \varphi \wedge \vartheta \longrightarrow \psi$
using assms by auto

lemma MMI˙3adant3r: assumes A1: $\varphi \wedge \psi \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $\varphi \wedge \psi \wedge \mathrm{ch} \wedge \tau \longrightarrow \vartheta$
using assms by auto

lemma MMI˙3adant1l: assumes A1: $\varphi \wedge \psi \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $(\tau \wedge \varphi) \wedge \psi \wedge \mathrm{ch} \longrightarrow \vartheta$
using assms by auto

lemma MMI˙3adant2r: assumes A1: $\varphi \wedge \psi \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $\varphi \wedge (\psi \wedge \tau) \wedge \mathrm{ch} \longrightarrow \vartheta$
using assms by auto

lemma MMI˙3bitr4rd: assumes A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow \mathrm{ch}$ and
A2: $\varphi \longrightarrow$
$\vartheta \longleftrightarrow \psi$ and
A3: $\varphi \longrightarrow$
$\tau \longleftrightarrow \mathrm{ch}$
shows $\varphi \longrightarrow$
$\tau \longleftrightarrow \vartheta$
using assms by auto

lemma MMI˙3anrev:
shows $\varphi \wedge \psi \wedge \mathrm{ch} \longleftrightarrow \mathrm{ch} \wedge \psi \wedge \varphi$
by auto

lemma MMI˙eqtr4: assumes A1: $\mathrm{A} = \mathrm{B}$ and
A2: $\mathrm{C} = \mathrm{B}$
shows $\mathrm{A} = \mathrm{C}$
using assms by auto

lemma MMI˙anidm:
shows $\varphi \wedge \varphi \longleftrightarrow \varphi$
by auto

lemma MMI˙bi2anan9r: assumes A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow \mathrm{ch}$ and

A2: $\vartheta \longrightarrow$
$\tau \longleftrightarrow \eta$
shows $\vartheta \wedge \varphi \longrightarrow$
$\psi \wedge \tau \longleftrightarrow$ ch $\wedge \eta$
using assms by auto

lemma MMI˙3imtr3g: assumes A1: $\varphi \longrightarrow \psi \longrightarrow$ ch and
A2: $\psi \longleftrightarrow \vartheta$ and
A3: ch $\longleftrightarrow \tau$
shows $\varphi \longrightarrow$
$\vartheta \longrightarrow \tau$
using assms by auto

lemma MMI˙a3d: assumes A1: $\varphi \longrightarrow$
$\neg\psi \longrightarrow \neg$ch
shows $\varphi \longrightarrow$ ch $\longrightarrow \psi$
using assms by auto

lemma MMI˙sylan9bbr: assumes A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch and
A2: $\vartheta \longrightarrow$
ch $\longleftrightarrow \tau$
shows $\vartheta \wedge \varphi \longrightarrow$
$\psi \longleftrightarrow \tau$
using assms by auto

lemma MMI˙sylan9bb: assumes A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch and
A2: $\vartheta \longrightarrow$
ch $\longleftrightarrow \tau$
shows $\varphi \wedge \vartheta \longrightarrow$
$\psi \longleftrightarrow \tau$
using assms by auto

lemma MMI˙3bitr3g: assumes A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch and
A2: $\psi \longleftrightarrow \vartheta$ and
A3: ch $\longleftrightarrow \tau$
shows $\varphi \longrightarrow$
$\vartheta \longleftrightarrow \tau$
using assms by auto

lemma MMI˙pm5˙21:
shows $\neg\varphi \wedge \neg\psi \longrightarrow$
$\varphi \longleftrightarrow \psi$
by auto

lemma MMI˙an6:
shows $(\varphi \wedge \psi \wedge \mathrm{ch}) \wedge \vartheta \wedge \tau \wedge \eta \longleftrightarrow$
$(\varphi \wedge \vartheta) \wedge (\psi \wedge \tau) \wedge \mathrm{ch} \wedge \eta$
by auto

lemma MMI˙syl3anl1: assumes A1: $(\varphi \wedge \psi \wedge \mathrm{ch}) \wedge \vartheta \longrightarrow \tau$ and
A2: $\eta \longrightarrow \varphi$
shows $(\eta \wedge \psi \wedge \mathrm{ch}) \wedge \vartheta \longrightarrow \tau$
using assms by auto

lemma MMI˙imp4a: assumes A1: $\varphi \longrightarrow$
$\psi \longrightarrow$
$\mathrm{ch} \longrightarrow$
$\vartheta \longrightarrow \tau$
shows $\varphi \longrightarrow$
$\psi \longrightarrow$
$\mathrm{ch} \wedge \vartheta \longrightarrow \tau$
using assms by auto

lemma (in MMIsar0) MMI˙breqan12rd: assumes A1: $\varphi \longrightarrow \mathrm{A} = \mathrm{B}$ and
A2: $\psi \longrightarrow \mathrm{C} = \mathrm{D}$
shows
$\psi \wedge \varphi \longrightarrow \mathrm{A} \text{ ¡ } \mathrm{C} \longleftrightarrow \mathrm{B} \text{ ¡ } \mathrm{D}$
$\psi \wedge \varphi \longrightarrow \mathrm{A} \leq \mathrm{C} \longleftrightarrow \mathrm{B} \leq \mathrm{D}$
using assms by auto

lemma (in MMIsar0) MMI˙3brtr4d: assumes A1: $\varphi \longrightarrow \mathrm{A} \text{ ¡ } \mathrm{B}$ and
A2: $\varphi \longrightarrow \mathrm{C} = \mathrm{A}$ and
A3: $\varphi \longrightarrow \mathrm{D} = \mathrm{B}$
shows $\varphi \longrightarrow \mathrm{C} \text{ ¡ } \mathrm{D}$
using assms by auto

lemma MMI˙adantrrr: assumes A1: $\varphi \wedge \psi \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $\varphi \wedge \psi \wedge \mathrm{ch} \wedge \tau \longrightarrow \vartheta$
using assms by auto

lemma MMI˙adantrlr: assumes A1: $\varphi \wedge \psi \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $\varphi \wedge (\psi \wedge \tau) \wedge \mathrm{ch} \longrightarrow \vartheta$
using assms by auto

lemma MMI˙imdistani: assumes A1: $\varphi \longrightarrow \psi \longrightarrow \mathrm{ch}$
shows $\varphi \wedge \psi \longrightarrow \varphi \wedge \mathrm{ch}$
using assms by auto

lemma MMI˙anabss3: assumes A1: $(\varphi \wedge \psi) \wedge \psi \longrightarrow \mathrm{ch}$
shows $\varphi \wedge \psi \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙mp3anl2: assumes A1: ψ and
A2: $(\varphi \wedge \psi \wedge \mathrm{ch}) \wedge \vartheta \longrightarrow \tau$
shows $(\varphi \wedge \mathrm{ch}) \wedge \vartheta \longrightarrow \tau$
using assms by auto

lemma MMI˙mpanl2: assumes A1: ψ and
A2: $(\varphi \wedge \psi) \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $\varphi \wedge \mathrm{ch} \longrightarrow \vartheta$
using assms by auto

lemma MMI˙mpancom: assumes A1: $\psi \longrightarrow \varphi$ and
A2: $\varphi \wedge \psi \longrightarrow \mathrm{ch}$
shows $\psi \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙or12:
shows $\varphi \vee \psi \vee \mathrm{ch} \longleftrightarrow \psi \vee \varphi \vee \mathrm{ch}$
by auto

lemma MMI˙rcla4ev: assumes A1: $\forall \mathrm{x}.\ \mathrm{x} = \mathrm{A} \longrightarrow\ \varphi(\mathrm{x}) \longleftrightarrow \psi$
shows $\mathrm{A} \in \mathrm{B} \wedge \psi \longrightarrow (\ \exists \mathrm{x} \in \mathrm{B}.\ \varphi(\mathrm{x})\ )$
using assms by auto

lemma MMI˙jctir: assumes A1: $\varphi \longrightarrow \psi$ and
A2: ch
shows $\varphi \longrightarrow \psi \wedge \mathrm{ch}$
using assms by auto

lemma MMI˙iffalse:
shows $\neg\varphi \longrightarrow\ \mathrm{if}(\varphi, \mathrm{A}, \mathrm{B}) = \mathrm{B}$
by auto

lemma MMI˙iftrue:
shows $\varphi \longrightarrow\ \mathrm{if}(\varphi, \mathrm{A}, \mathrm{B}) = \mathrm{A}$
by auto

lemma MMI˙pm2˙61d2: assumes A1: $\varphi \longrightarrow$
$\neg\psi \longrightarrow \mathrm{ch}$ and
A2: $\psi \longrightarrow \mathrm{ch}$
shows $\varphi \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙pm2˙61dan: assumes A1: $\varphi \wedge \psi \longrightarrow \mathrm{ch}$ and
A2: $\varphi \wedge \neg\psi \longrightarrow \mathrm{ch}$
shows $\varphi \longrightarrow \mathrm{ch}$
using assms by auto

lemma MMI˙orcanai: assumes A1: $\varphi \longrightarrow \psi \vee$ ch
shows $\varphi \wedge \neg\psi \longrightarrow$ ch
using assms by auto

lemma MMI˙ifcl:
shows $A \in C \wedge B \in C \longrightarrow$ if(φ, A, B) $\in$ C
by auto

lemma MMI˙imim2i: assumes A1: $\varphi \longrightarrow \psi$
shows (ch $\longrightarrow \varphi$) $\longrightarrow$ ch $\longrightarrow \psi$
using assms by auto

lemma MMI˙com13: assumes A1: $\varphi \longrightarrow$
$\psi \longrightarrow$ ch $\longrightarrow \vartheta$
shows ch $\longrightarrow$
$\psi \longrightarrow$
$\varphi \longrightarrow \vartheta$
using assms by auto

lemma MMI˙rcla4v: assumes A1: $\forall$x. x = A $\longrightarrow$ φ(x) $\longleftrightarrow \psi$
shows $A \in B \longrightarrow$ ($\forall$x$\in$B. φ(x)) $\longrightarrow \psi$
using assms by auto

lemma MMI˙syl5d: assumes A1: $\varphi \longrightarrow$
$\psi \longrightarrow$ ch $\longrightarrow \vartheta$ and
A2: $\varphi \longrightarrow \tau \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \longrightarrow$
$\tau \longrightarrow \vartheta$
using assms by auto

lemma MMI˙eqcoms: assumes A1: A = B $\longrightarrow \varphi$
shows B = A $\longrightarrow \varphi$
using assms by auto

lemma MMI˙rgen: assumes A1: $\forall$x. x $\in$ A $\longrightarrow \varphi$(x)
shows $\forall$x$\in$A. φ(x)
using assms by auto

lemma (in MMIsar0) MMI˙reex:
shows $\mathbb{R} = \mathbb{R}$
by auto

lemma MMI˙sstri: assumes A1: A $\subseteq$B and
A2: B $\subseteq$C
shows A $\subseteq$C

using assms by auto

lemma MMI˙ssexi: assumes A1: B = B and
A2: A ⊆B
shows A = A
using assms by auto

end

79 Complex numbers in Metamatah - introduction

theory MMI˙Complex˙ZF imports MMI˙logic˙and˙sets

begin

This theory contains theorems (with proofs) about complex numbers imported from the Metamath's set.mm database. The original Metamath proofs were mostly written by Norman Megill, see the Metamath Proof Explorer pages for full atribution. This theory contains about 200 theorems from "recnt" to "div11t".

lemma (in MMIsar0) MMI˙recnt:
shows A ∈ ℝ ⟶ A ∈ ℂ
proof -
have S1: ℝ ⊆ ℂ by (rule MMI˙axresscn)
from S1 show A ∈ ℝ ⟶ A ∈ ℂ by (rule MMI˙sseli)
qed

lemma (in MMIsar0) MMI˙recn: assumes A1: A ∈ ℝ
shows A ∈ ℂ
proof -
have S1: ℝ ⊆ ℂ by (rule MMI˙axresscn)
from A1 have S2: A ∈ ℝ.
from S1 S2 show A ∈ ℂ by (rule MMI˙sselii)
qed

lemma (in MMIsar0) MMI˙recnd: assumes A1: φ ⟶ A ∈ ℝ
shows φ ⟶ A ∈ ℂ
proof -
from A1 have S1: φ ⟶ A ∈ ℝ.
have S2: A ∈ ℝ ⟶ A ∈ ℂ by (rule MMI˙recnt)
from S1 S2 show φ ⟶ A ∈ ℂ by (rule MMI˙syl)
qed

lemma (in MMIsar0) MMI˙elimne0:
shows if (A ≠ **0** , A , **1**) ≠ **0**
proof -
have S1: A = if (A ≠ **0** , A , **1**) ⟶

(A ≠ **0** ⟷ if (A ≠ **0** , A , **1**) ≠ **0**) by (rule MMI˙neeq1)
have S2: **1** = if (A ≠ **0** , A , **1**) ⟶
(**1** ≠ **0** ⟷ if (A ≠ **0** , A , **1**) ≠ **0**) by (rule MMI˙neeq1)
have S3: **1** ≠ **0** by (rule MMI˙ax1ne0)
from S1 S2 S3 show if (A ≠ **0** , A , **1**) ≠ **0** by (rule MMI˙elimhyp)
qed

lemma (in MMIsar0) MMI˙addex:
shows + isASet
proof -
have S1: ℂ isASet by (rule MMI˙axcnex)
have S2: ℂ isASet by (rule MMI˙axcnex)
from S1 S2 have S3: (ℂ × ℂ) isASet by (rule MMI˙xpex)
have S4: + : (ℂ × ℂ) → ℂ by (rule MMI˙axaddopr)
have S5: (ℂ × ℂ) isASet ⟶
(+ : (ℂ × ℂ) → ℂ ⟶ + isASet) by (rule MMI˙fex)
from S3 S4 S5 show + isASet by (rule MMI˙mp2)
qed

lemma (in MMIsar0) MMI˙mulex:
shows · isASet
proof -
have S1: ℂ isASet by (rule MMI˙axcnex)
have S2: ℂ isASet by (rule MMI˙axcnex)
from S1 S2 have S3: (ℂ × ℂ) isASet by (rule MMI˙xpex)
have S4: · : (ℂ × ℂ) → ℂ by (rule MMI˙axmulopr)
have S5: (ℂ × ℂ) isASet ⟶
(· : (ℂ × ℂ) → ℂ ⟶ · isASet) by (rule MMI˙fex)
from S3 S4 S5 show · isASet by (rule MMI˙mp2)
qed

lemma (in MMIsar0) MMI˙adddirt:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) · C) = ((A · C) + (B · C))
proof -
have S1: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶
(C · (A + B)) = ((C · A) + (C · B))
by (rule MMI˙axdistr)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(C · (A + B)) = ((C · A) + (C · B)) by (rule MMI˙3coml)
have S3: ((A + B) ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) · C) = (C · (A + B)) by (rule MMI˙axmulcom)
have S4: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + B) ∈ ℂ by (rule MMI˙axaddcl)
from S3 S4 have S5: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ C ∈ ℂ) ⟶
((A + B) · C) = (C · (A + B)) by (rule MMI˙sylan)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) · C) = (C · (A + B)) by (rule MMI˙3impa)
have S7: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) = (C · A)
by (rule MMI˙axmulcom)

from S7 have S8: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) = (C · A)
by (rule MMI˙3adant2)
have S9: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B · C) = (C · B)
by (rule MMI˙axmulcom)
from S9 have S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B · C) = (C · B)
by (rule MMI˙3adant1)
from S8 S10 have S11: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · C) + (B · C)) = ((C · A) + (C · B))
by (rule MMI˙opreq12d)
from S2 S6 S11 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) · C) = ((A · C) + (B · C))
by (rule MMI˙3eqtr4d)
qed

lemma (in MMIsar0) MMI˙addcl: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows (A + B) ∈ ℂ
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
have S3: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + B) ∈ ℂ by (rule MMI˙axaddcl)
from S1 S2 S3 show (A + B) ∈ ℂ by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙mulcl: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows (A · B) ∈ ℂ
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
have S3: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A · B) ∈ ℂ by (rule MMI˙axmulcl)
from S1 S2 S3 show (A · B) ∈ ℂ by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙addcom: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows (A + B) = (B + A)
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
have S3: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + B) = (B + A)
by (rule MMI˙axaddcom)
from S1 S2 S3 show (A + B) = (B + A) by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙mulcom: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows (A · B) = (B · A)
proof -

```
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A · B ) = ( B · A )
    by (rule MMI'axmulcom)
  from S1 S2 S3 show ( A · B ) = ( B · A ) by (rule MMI'mp2an)
qed

lemma (in MMIsar0) MMI'addass: assumes A1: A ∈ ℂ and
   A2: B ∈ ℂ and
   A3: C ∈ ℂ
  shows ( ( A + B ) + C ) = ( A + ( B + C ) )
proof -
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  from A3 have S3: C ∈ ℂ.
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + B ) + C ) =
    ( A + ( B + C ) ) by (rule MMI'axaddass)
  from S1 S2 S3 S4 show ( ( A + B ) + C ) =
    ( A + ( B + C ) ) by (rule MMI'mp3an)
qed

lemma (in MMIsar0) MMI'mulass: assumes A1: A ∈ ℂ and
   A2: B ∈ ℂ and
   A3: C ∈ ℂ
  shows ( ( A · B ) · C ) = ( A · ( B · C ) )
proof -
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  from A3 have S3: C ∈ ℂ.
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A · B ) · C ) =
    ( A · ( B · C ) ) by (rule MMI'axmulass)
  from S1 S2 S3 S4 show ( ( A · B ) · C ) = ( A · ( B · C ) )
    by (rule MMI'mp3an)
qed

lemma (in MMIsar0) MMI'adddi: assumes A1: A ∈ ℂ and
   A2: B ∈ ℂ and
   A3: C ∈ ℂ
  shows ( A · ( B + C ) ) = ( ( A · B ) + ( A · C ) )
proof -
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  from A3 have S3: C ∈ ℂ.
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A · ( B + C ) ) =
    ( ( A · B ) + ( A · C ) ) by (rule MMI'axdistr)
  from S1 S2 S3 S4 show ( A · ( B + C ) ) =
    ( ( A · B ) + ( A · C ) ) by (rule MMI'mp3an)
qed
```

lemma (in MMIsar0) MMI˙adddir: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows ((A + B) · C) = ((A · C) + (B · C))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + B) · C) =
((A · C) + (B · C)) by (rule MMI˙adddirt)
from S1 S2 S3 S4 show ((A + B) · C) =
((A · C) + (B · C)) by (rule MMI˙mp3an)
qed

lemma (in MMIsar0) MMI˙1cn:
shows **1** ∈ ℂ
proof -
have S1: **1** ∈ ℝ by (rule MMI˙ax1re)
from S1 show **1** ∈ ℂ by (rule MMI˙recn)
qed

lemma (in MMIsar0) MMI˙0cn:
shows **0** ∈ ℂ
proof -
have S1: ((i · i) + **1**) = **0** by (rule MMI˙axi2m1)
have S2: i ∈ ℂ by (rule MMI˙axicn)
have S3: i ∈ ℂ by (rule MMI˙axicn)
from S2 S3 have S4: (i · i) ∈ ℂ by (rule MMI˙mulcl)
have S5: **1** ∈ ℂ by (rule MMI˙1cn)
from S4 S5 have S6: ((i · i) + **1**) ∈ ℂ by (rule MMI˙addcl)
from S1 S6 show **0** ∈ ℂ by (rule MMI˙eqeltrr)
qed

lemma (in MMIsar0) MMI˙addid1: assumes A1: A ∈ ℂ
shows (A + **0**) = A
proof -
from A1 have S1: A ∈ ℂ.
have S2: A ∈ ℂ ⟶ (A + **0**) = A by (rule MMI˙ax0id)
from S1 S2 show (A + **0**) = A by (rule MMI˙ax˙mp)
qed

lemma (in MMIsar0) MMI˙addid2: assumes A1: A ∈ ℂ
shows (**0** + A) = A
proof -
have S1: **0** ∈ ℂ by (rule MMI˙0cn)
from A1 have S2: A ∈ ℂ.
from S1 S2 have S3: (**0** + A) = (A + **0**) by (rule MMI˙addcom)
from A1 have S4: A ∈ ℂ.
from S4 have S5: (A + **0**) = A by (rule MMI˙addid1)

from S3 S5 show ($\mathbf{0}$ + A) = A by (rule MMI˙eqtr)
qed

lemma (in MMIsar0) MMI˙mulid1: assumes A1: A ∈ ℂ
shows (A · $\mathbf{1}$) = A
proof -
from A1 have S1: A ∈ ℂ.
have S2: A ∈ ℂ ⟶ (A · $\mathbf{1}$) = A by (rule MMI˙ax1id)
from S1 S2 show (A · $\mathbf{1}$) = A by (rule MMI˙ax˙mp)
qed

lemma (in MMIsar0) MMI˙mulid2: assumes A1: A ∈ ℂ
shows ($\mathbf{1}$ · A) = A
proof -
have S1: $\mathbf{1}$ ∈ ℂ by (rule MMI˙1cn)
from A1 have S2: A ∈ ℂ.
from S1 S2 have S3: ($\mathbf{1}$ · A) = (A · $\mathbf{1}$) by (rule MMI˙mulcom)
from A1 have S4: A ∈ ℂ.
from S4 have S5: (A · $\mathbf{1}$) = A by (rule MMI˙mulid1)
from S3 S5 show ($\mathbf{1}$ · A) = A by (rule MMI˙eqtr)
qed

lemma (in MMIsar0) MMI˙negex: assumes A1: A ∈ ℂ
shows ∃ x ∈ ℂ . (A + x) = $\mathbf{0}$
proof -
from A1 have S1: A ∈ ℂ.
have S2: A ∈ ℂ ⟶ (∃ x ∈ ℂ . (A + x) = $\mathbf{0}$) by (rule MMI˙axnegex)
from S1 S2 show ∃ x ∈ ℂ . (A + x) = $\mathbf{0}$ by (rule MMI˙ax˙mp)
qed

lemma (in MMIsar0) MMI˙recex: assumes A1: A ∈ ℂ and
A2: A ≠ $\mathbf{0}$
shows ∃ x ∈ ℂ . (A · x) = $\mathbf{1}$
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: A ≠ $\mathbf{0}$.
have S3: (A ∈ ℂ ∧ A ≠ $\mathbf{0}$) ⟶ (∃ x ∈ ℂ . (A · x) = $\mathbf{1}$)
by (rule MMI˙axrecex)
from S1 S2 S3 show ∃ x ∈ ℂ . (A · x) = $\mathbf{1}$ by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙readdcl: assumes A1: A ∈ ℝ and
A2: B ∈ ℝ
shows (A + B) ∈ ℝ
proof -
from A1 have S1: A ∈ ℝ.

from A2 have S2: B ∈ ℝ.
have S3: (A ∈ ℝ ∧ B ∈ ℝ) ⟶ (A + B) ∈ ℝ by (rule MMI˙axaddrcl)
from S1 S2 S3 show (A + B) ∈ ℝ by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙remulcl: assumes A1: A ∈ ℝ and
A2: B ∈ ℝ
shows (A · B) ∈ ℝ
proof -
from A1 have S1: A ∈ ℝ.
from A2 have S2: B ∈ ℝ.
have S3: (A ∈ ℝ ∧ B ∈ ℝ) ⟶ (A · B) ∈ ℝ by (rule MMI˙axmulrcl)
from S1 S2 S3 show (A · B) ∈ ℝ by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙addcan: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows (A + B) = (A + C) ⟷ B = C
proof -
from A1 have S1: A ∈ ℂ.
from S1 have S2: ∃ x ∈ ℂ . (A + x) = **0** by (rule MMI˙negex)
from A1 have S3: A ∈ ℂ.
from A2 have S4: B ∈ ℂ.
{ fix x
have S5: (x ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((x + A) + B) =
(x + (A + B)) by (rule MMI˙axaddass)
from S4 S5 have S6: (x ∈ ℂ ∧ A ∈ ℂ) ⟶ ((x + A) + B) =
(x + (A + B)) by (rule MMI˙mp3an3)
from A3 have S7: C ∈ ℂ.
have S8: (x ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ) ⟶ ((x + A) + C) =
(x + (A + C)) by (rule MMI˙axaddass)
from S7 S8 have S9: (x ∈ ℂ ∧ A ∈ ℂ) ⟶ ((x + A) + C) =
(x + (A + C)) by (rule MMI˙mp3an3)
from S6 S9 have S10: (x ∈ ℂ ∧ A ∈ ℂ) ⟶
(((x + A) + B) = ((x + A) + C) ⟷
(x + (A + B)) = (x + (A + C)))
by (rule MMI˙eqeq12d)
from S3 S10 have S11: x ∈ ℂ ⟶ (((x + A) + B) =
((x + A) + C) ⟷ (x + (A + B)) =
(x + (A + C))) by (rule MMI˙mpan2)
have S12: (A + B) = (A + C) ⟶ (x + (A + B)) =
(x + (A + C)) by (rule MMI˙opreq2)
from S11 S12 have S13: x ∈ ℂ ⟶ ((A + B) = (A + C) ⟶
((x + A) + B) = ((x + A) + C))
by (rule MMI˙syl5bir)
from S13 have S14: (x ∈ ℂ ∧ (A + x) = **0**) ⟶ ((A + B) =
(A + C) ⟶ ((x + A) + B) =
((x + A) + C)) by (rule MMI˙adantr)

```
    from A1 have S15: A ∈ ℂ.
    have S16: ( A ∈ ℂ ∧ x ∈ ℂ ) ⟶ ( A + x ) = ( x + A )
      by (rule MMI˙axaddcom)
    from S15 S16 have S17: x ∈ ℂ ⟶ ( A + x ) = ( x + A )
      by (rule MMI˙mpan)
    from S17 have S18: x ∈ ℂ ⟶ ( ( A + x ) = 𝟬 ⟷
      ( x + A ) = 𝟬 ) by (rule MMI˙eqeq1d)
    have S19: ( x + A ) = 𝟬 ⟶ ( ( x + A ) + B ) =
      ( 𝟬 + B ) by (rule MMI˙opreq1)
    from A2 have S20: B ∈ ℂ.
    from S20 have S21: ( 𝟬 + B ) = B by (rule MMI˙addid2)
    from S19 S21 have S22: ( x + A ) = 𝟬 ⟶
      ( ( x + A ) + B ) = B by (rule MMI˙syl6eq)
    have S23: ( x + A ) = 𝟬 ⟶ ( ( x + A ) + C ) =
      ( 𝟬 + C ) by (rule MMI˙opreq1)
    from A3 have S24: C ∈ ℂ.
    from S24 have S25: ( 𝟬 + C ) = C by (rule MMI˙addid2)
    from S23 S25 have S26: ( x + A ) = 𝟬 ⟶
      ( ( x + A ) + C ) = C by (rule MMI˙syl6eq)
    from S22 S26 have S27: ( x + A ) = 𝟬 ⟶
      ( ( ( x + A ) + B ) = ( ( x + A ) + C ) ⟷ B = C )
      by (rule MMI˙eqeq12d)
    from S18 S27 have S28: x ∈ ℂ ⟶ ( ( A + x ) = 𝟬 ⟶
      ( ( ( x + A ) + B ) = ( ( x + A ) + C ) ⟷ B = C ) )
      by (rule MMI˙syl6bi)
    from S28 have S29: ( x ∈ ℂ ∧ ( A + x ) = 𝟬 ) ⟶
      ( ( ( x + A ) + B ) = ( ( x + A ) + C ) ⟷ B = C )
      by (rule MMI˙imp)
    from S14 S29 have S30: ( x ∈ ℂ ∧ ( A + x ) = 𝟬 ) ⟶
      ( ( A + B ) = ( A + C ) ⟶ B = C ) by (rule MMI˙sylibd)
    from S30 have x ∈ ℂ ⟶ ( ( A + x ) = 𝟬 ⟶
      ( ( A + B ) = ( A + C ) ⟶ B = C ) ) by (rule MMI˙ex)
  } then have S31: ∀ x. (x ∈ ℂ ⟶ ( ( A + x ) = 𝟬 ⟶
      ( ( A + B ) = ( A + C ) ⟶ B = C ) )) by auto
  from S31 have S32: ( ∃ x ∈ ℂ . ( A + x ) = 𝟬 ) ⟶
    ( ( A + B ) = ( A + C ) ⟶ B = C ) by (rule MMI˙r19˙23aiv)
  from S2 S32 have S33: ( A + B ) = ( A + C ) ⟶ B = C
    by (rule MMI˙ax˙mp)
  have S34: B = C ⟶ ( A + B ) = ( A + C ) by (rule MMI˙opreq2)
  from S33 S34 show ( A + B ) = ( A + C ) ⟷ B = C
    by (rule MMI˙impbi)
qed

lemma (in MMIsar0) MMI˙addcan2: assumes A1: A ∈ ℂ and
   A2: B ∈ ℂ and
   A3: C ∈ ℂ
  shows ( A + C ) = ( B + C ) ⟷ A = B
proof -
  from A1 have S1: A ∈ ℂ.
```

from A3 have S2: C ∈ ℂ.
from S1 S2 have S3: (A + C) = (C + A) by (rule MMI˙addcom)
from A2 have S4: B ∈ ℂ.
from A3 have S5: C ∈ ℂ.
from S4 S5 have S6: (B + C) = (C + B) by (rule MMI˙addcom)
from S3 S6 have S7: (A + C) = (B + C) ⟷
(C + A) = (C + B) by (rule MMI˙eqeq12i)
from A3 have S8: C ∈ ℂ.
from A1 have S9: A ∈ ℂ.
from A2 have S10: B ∈ ℂ.
from S8 S9 S10 have S11: (C + A) = (C + B) ⟷ A = B
by (rule MMI˙addcan)
from S7 S11 show (A + C) = (B + C) ⟷ A = B by (rule MMI˙bitr)
qed

lemma (in MMIsar0) MMI˙addcant:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) = (A + C) ⟷ B = C)
proof -
have S1: A = if (A ∈ ℂ , A , **0**) ⟶ (A + B) = (if (A ∈ ℂ , A , **0**) + B)
by (rule MMI˙opreq1)
have S2: A = if (A ∈ ℂ , A , **0**) ⟶
(A + C) = (if (A ∈ ℂ , A , **0**) + C) by (rule MMI˙opreq1)
from S1 S2 have S3: A = if (A ∈ ℂ , A , **0**) ⟶
((A + B) = (A + C) ⟷
(if (A ∈ ℂ , A , **0**) + B) = (if (A ∈ ℂ , A , **0**) + C))
by (rule MMI˙eqeq12d)
from S3 have S4: A = if (A ∈ ℂ , A , **0**) ⟶
(((A + B) = (A + C) ⟷ B = C) ⟷
((if (A ∈ ℂ , A , **0**) + B) = (if (A ∈ ℂ , A , **0**) + C)
⟷ B = C)) by (rule MMI˙bibi1d)
have S5: B = if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) + B) =
(if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
from S5 have S6: B = if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) + B) = (if (A ∈ ℂ , A , **0**) + C)
⟷ (if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) =
(if (A ∈ ℂ , A , **0**) + C)) by (rule MMI˙eqeq1d)
have S7: B = if (B ∈ ℂ , B , **0**) ⟶ (B = C ⟷
if (B ∈ ℂ , B , **0**) = C) by (rule MMI˙eqeq1)
from S6 S7 have S8: B = if (B ∈ ℂ , B , **0**) ⟶
(((if (A ∈ ℂ , A , **0**) + B) =
(if (A ∈ ℂ , A , **0**) + C) ⟷ B = C) ⟷
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) =
(if (A ∈ ℂ , A , **0**) + C) ⟷ if (B ∈ ℂ , B , **0**) = C))
by (rule MMI˙bibi12d)
have S9: C = if (C ∈ ℂ , C , **0**) ⟶ (if (A ∈ ℂ , A , **0**) + C) =
(if (A ∈ ℂ , A , **0**) + if (C ∈ ℂ , C , **0**))
by (rule MMI˙opreq2)

from S9 have S10: C = if (C ∈ ℂ , C , **0**) ⟶
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) =
(if (A ∈ ℂ , A , **0**) + C) ⟷
(if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) =
(if (A ∈ ℂ , A , **0**) + if (C ∈ ℂ , C , **0**)))
by (rule MMI˙eqeq2d)
have S11: C = if (C ∈ ℂ , C , **0**) ⟶ (if (B ∈ ℂ , B , **0**) = C ⟷
if (B ∈ ℂ , B , **0**) = if (C ∈ ℂ , C , **0**)) by (rule MMI˙eqeq2)
from S10 S11 have S12: C = if (C ∈ ℂ , C , **0**) ⟶
(((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) =
(if (A ∈ ℂ , A , **0**) + C) ⟷ if (B ∈ ℂ , B , **0**) = C) ⟷
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) =
(if (A ∈ ℂ , A , **0**) + if (C ∈ ℂ , C , **0**)) ⟷
if (B ∈ ℂ , B , **0**) = if (C ∈ ℂ , C , **0**))) by (rule MMI˙bibi12d)
have S13: **0** ∈ ℂ by (rule MMI˙0cn)
from S13 have S14: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S15: **0** ∈ ℂ by (rule MMI˙0cn)
from S15 have S16: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
have S17: **0** ∈ ℂ by (rule MMI˙0cn)
from S17 have S18: if (C ∈ ℂ , C , **0**) ∈ ℂ by (rule MMI˙elimel)
from S14 S16 S18 have S19:
(if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) =
(if (A ∈ ℂ , A , **0**) + if (C ∈ ℂ , C , **0**)) ⟷
if (B ∈ ℂ , B , **0**) = if (C ∈ ℂ , C , **0**) by (rule MMI˙addcan)
from S4 S8 S12 S19 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) = (A + C) ⟷ B = C) by (rule MMI˙dedth3h)
qed

lemma (in MMIsar0) MMI˙addcan2t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + C) = (B + C) ⟷
A = B)
proof -
have S1: (C ∈ ℂ ∧ A ∈ ℂ) ⟶ (C + A) = (A + C)
by (rule MMI˙axaddcom)
from S1 have S2: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶ (C + A) =
(A + C) by (rule MMI˙3adant3)
have S3: (C ∈ ℂ ∧ B ∈ ℂ) ⟶ (C + B) = (B + C)
by (rule MMI˙axaddcom)
from S3 have S4: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶ (C + B) =
(B + C) by (rule MMI˙3adant2)
from S2 S4 have S5: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶
((C + A) = (C + B) ⟷ (A + C) = (B + C))
by (rule MMI˙eqeq12d)
have S6: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((C + A) =
(C + B) ⟷ A = B) by (rule MMI˙addcant)
from S5 S6 have S7: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A + C) =
(B + C) ⟷ A = B) by (rule MMI˙bitr3d)
from S7 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + C) =
(B + C) ⟷ A = B) by (rule MMI˙3coml)

qed

lemma (in MMIsar0) MMI˙add12t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A + (B + C)) =
(B + (A + C))
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + B) = (B + A)
by (rule MMI˙axaddcom)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A + B) + C) =
((B + A) + C) by (rule MMI˙opreq1d)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) + C) = ((B + A) + C)
by (rule MMI˙3adant3)
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + B) + C) =
(A + (B + C)) by (rule MMI˙axaddass)
have S5: (B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ) ⟶ ((B + A) + C) =
(B + (A + C)) by (rule MMI˙axaddass)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((B + A) + C) = (B + (A + C)) by (rule MMI˙3com12)
from S3 S4 S6 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (B + C)) = (B + (A + C))
by (rule MMI˙3eqtr3d)
qed

lemma (in MMIsar0) MMI˙add23t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + B) + C) =
((A + C) + B)
proof -
have S1: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B + C) = (C + B)
by (rule MMI˙axaddcom)
from S1 have S2: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A + (B + C)) =
(A + (C + B)) by (rule MMI˙opreq2d)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (B + C)) = (A + (C + B))
by (rule MMI˙3adant1)
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + B) + C) =
(A + (B + C)) by (rule MMI˙axaddass)
have S5: (A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A + C) + B) =
(A + (C + B)) by (rule MMI˙axaddass)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + C) + B) = (A + (C + B)) by (rule MMI˙3com23)
from S3 S4 S6 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) + C) = ((A + C) + B)
by (rule MMI˙3eqtr4d)
qed

lemma (in MMIsar0) MMI˙add4t:

shows ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) + (C + D)) = ((A + C) + (B + D))
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) + C) = ((A + C) + B) by (rule MMI˙add23t)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(((A + B) + C) + D) =
(((A + C) + B) + D) by (rule MMI˙opreq1d)
from S2 have S3: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ C ∈ ℂ) ⟶
(((A + B) + C) + D) =
(((A + C) + B) + D) by (rule MMI˙3expa)
from S3 have S4: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A + B) + C) + D) =
(((A + C) + B) + D) by (rule MMI˙adantrr)
have S5: ((A + B) ∈ ℂ ∧ C ∈ ℂ ∧ D ∈ ℂ) ⟶
(((A + B) + C) + D) =
((A + B) + (C + D)) by (rule MMI˙axaddass)
from S5 have S6: ((A + B) ∈ ℂ ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A + B) + C) + D) =
((A + B) + (C + D)) by (rule MMI˙3expb)
have S7: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + B) ∈ ℂ by (rule MMI˙axaddcl)
from S6 S7 have S8: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A + B) + C) + D) =
((A + B) + (C + D)) by (rule MMI˙sylan)
have S9: ((A + C) ∈ ℂ ∧ B ∈ ℂ ∧ D ∈ ℂ) ⟶
(((A + C) + B) + D) =
((A + C) + (B + D)) by (rule MMI˙axaddass)
from S9 have S10: ((A + C) ∈ ℂ ∧ (B ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A + C) + B) + D) =
((A + C) + (B + D)) by (rule MMI˙3expb)
have S11: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A + C) ∈ ℂ by (rule MMI˙axaddcl)
from S10 S11 have S12: ((A ∈ ℂ ∧ C ∈ ℂ) ∧ (B ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A + C) + B) + D) =
((A + C) + (B + D)) by (rule MMI˙sylan)
from S12 have S13: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A + C) + B) + D) =
((A + C) + (B + D)) by (rule MMI˙an4s)
from S4 S8 S13 show ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) + (C + D)) =
((A + C) + (B + D)) by (rule MMI˙3eqtr3d)
qed

lemma (in MMIsar0) MMI˙add42t:
shows ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) + (C + D)) = ((A + C) + (D + B))
proof -
have S1: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) + (C + D)) =
((A + C) + (B + D)) by (rule MMI˙add4t)

have S2: (B ∈ ℂ ∧ D ∈ ℂ) ⟶ (B + D) =
(D + B) by (rule MMI˙axaddcom)
from S2 have S3: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(B + D) = (D + B) by (rule MMI˙ad2ant2l)
from S3 have S4: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + C) + (B + D)) =
((A + C) + (D + B)) by (rule MMI˙opreq2d)
from S1 S4 show ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) + (C + D)) =
((A + C) + (D + B)) by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙add12: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows (A + (B + C)) = (B + (A + C))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A + (B + C)) =
(B + (A + C)) by (rule MMI˙add12t)
from S1 S2 S3 S4 show (A + (B + C)) =
(B + (A + C)) by (rule MMI˙mp3an)
qed

lemma (in MMIsar0) MMI˙add23: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows ((A + B) + C) = ((A + C) + B)
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) + C) = ((A + C) + B) by (rule MMI˙add23t)
from S1 S2 S3 S4 show ((A + B) + C) =
((A + C) + B) by (rule MMI˙mp3an)
qed

lemma (in MMIsar0) MMI˙add4: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ and
A4: D ∈ ℂ
shows ((A + B) + (C + D)) =
((A + C) + (B + D))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.

from S1 S2 have S3: A ∈ ℂ ∧ B ∈ ℂ by (rule MMI˙pm3˙2i)
from A3 have S4: C ∈ ℂ.
from A4 have S5: D ∈ ℂ.
from S4 S5 have S6: C ∈ ℂ ∧ D ∈ ℂ by (rule MMI˙pm3˙2i)
have S7: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) + (C + D)) =
((A + C) + (B + D)) by (rule MMI˙add4t)
from S3 S6 S7 show ((A + B) + (C + D)) =
((A + C) + (B + D)) by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙add42: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ and
A4: D ∈ ℂ
shows ((A + B) + (C + D)) =
((A + C) + (D + B))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
from A4 have S4: D ∈ ℂ.
from S1 S2 S3 S4 have S5: ((A + B) + (C + D)) =
((A + C) + (B + D)) by (rule MMI˙add4)
from A2 have S6: B ∈ ℂ.
from A4 have S7: D ∈ ℂ.
from S6 S7 have S8: (B + D) = (D + B) by (rule MMI˙addcom)
from S8 have S9: ((A + C) + (B + D)) =
((A + C) + (D + B)) by (rule MMI˙opreq2i)
from S5 S9 show ((A + B) + (C + D)) =
((A + C) + (D + B)) by (rule MMI˙eqtr)
qed

lemma (in MMIsar0) MMI˙addid2t:
shows A ∈ ℂ ⟶ (**0** + A) = A
proof -
have S1: **0** ∈ ℂ by (rule MMI˙0cn)
have S2: (**0** ∈ ℂ ∧ A ∈ ℂ) ⟶ (**0** + A) = (A + **0**)
by (rule MMI˙axaddcom)
from S1 S2 have S3: A ∈ ℂ ⟶ (**0** + A) = (A + **0**)
by (rule MMI˙mpan)
have S4: A ∈ ℂ ⟶ (A + **0**) = A by (rule MMI˙ax0id)
from S3 S4 show A ∈ ℂ ⟶ (**0** + A) = A by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙peano2cn:
shows A ∈ ℂ ⟶ (A + **1**) ∈ ℂ
proof -
have S1: **1** ∈ ℂ by (rule MMI˙1cn)

have S2: (A ∈ ℂ ∧ **1** ∈ ℂ) ⟶ (A + **1**) ∈ ℂ by (rule MMI˙axaddcl)
from S1 S2 show A ∈ ℂ ⟶ (A + **1**) ∈ ℂ by (rule MMI˙mpan2)
qed

lemma (in MMIsar0) MMI˙peano2re:
shows A ∈ ℝ ⟶ (A + **1**) ∈ ℝ
proof -
have S1: **1** ∈ ℝ by (rule MMI˙ax1re)
have S2: (A ∈ ℝ ∧ **1** ∈ ℝ) ⟶ (A + **1**) ∈ ℝ by (rule MMI˙axaddrcl)
from S1 S2 show A ∈ ℝ ⟶ (A + **1**) ∈ ℝ by (rule MMI˙mpan2)
qed

lemma (in MMIsar0) MMI˙negeu: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows ∃! x . x ∈ ℂ ∧ (A + x) = B
proof -
{ fix x y
have S1: x = y ⟶ (A + x) = (A + y) by (rule MMI˙opreq2)
from S1 have x = y ⟶ ((A + x) = B ⟷ (A + y) = B)
by (rule MMI˙eqeq1d)
} then have S2: ∀x y. x = y ⟶ ((A + x) = B ⟷
(A + y) = B) by simp
from S2 have S3: (∃! x . x ∈ ℂ ∧ (A + x) = B) ⟷
((∃ x ∈ ℂ . (A + x) = B) ∧
(∀ x ∈ ℂ . ∀ y ∈ ℂ . (((A + x) = B ∧ (A + y) = B) ⟶
x = y))) by (rule MMI˙reu4)
from A1 have S4: A ∈ ℂ.
from S4 have S5: ∃ y ∈ ℂ . (A + y) = **0** by (rule MMI˙negex)
from A2 have S6: B ∈ ℂ.
{ fix y
have S7: (y ∈ ℂ ∧ B ∈ ℂ) ⟶ (y + B) ∈ ℂ by (rule MMI˙axaddcl)
from S6 S7 have S8: y ∈ ℂ ⟶ (y + B) ∈ ℂ by (rule MMI˙mpan2)
have S9: (y + B) ∈ ℂ ⟷ (∃ x ∈ ℂ . x = (y + B))
by (rule MMI˙risset)
from S8 S9 have S10: y ∈ ℂ ⟶ (∃ x ∈ ℂ . x = (y + B))
by (rule MMI˙sylib)
{ fix x
have S11: x = (y + B) ⟶ (A + x) =
(A + (y + B)) by (rule MMI˙opreq2)
from A1 have S12: A ∈ ℂ.
from A2 have S13: B ∈ ℂ.
have S14: (A ∈ ℂ ∧ y ∈ ℂ ∧ B ∈ ℂ) ⟶
((A + y) + B) = (A + (y + B))
by (rule MMI˙axaddass)
from S12 S13 S14 have S15: y ∈ ℂ ⟶ ((A + y) + B) =
(A + (y + B)) by (rule MMI˙mp3an13)
from S15 have S16: y ∈ ℂ ⟶ (A + (y + B)) =

((A + y) + B) by (rule MMI˙eqcomd)
from S11 S16 have S17: (y ∈ ℂ ∧ x = (y + B))
⟶ (A + x) = ((A + y) + B) by (rule MMI˙sylan9eqr)
have S18: (A + y) = **0** ⟶
((A + y) + B) = (**0** + B) by (rule MMI˙opreq1)
from A2 have S19: B ∈ ℂ.
from S19 have S20: (**0** + B) = B by (rule MMI˙addid2)
from S18 S20 have S21: (A + y) = **0** ⟶
((A + y) + B) = B by (rule MMI˙syl6eq)
from S17 S21 have S22: ((A + y) = **0** ∧ (y ∈ ℂ ∧ x =
(y + B))) ⟶ (A + x) = B by (rule MMI˙sylan9eqr)
from S22 have S23: (A + y) = **0** ⟶
(y ∈ ℂ ⟶ (x = (y + B) ⟶ (A + x) = B))
by (rule MMI˙exp32)
from S23 have S24: (y ∈ ℂ ∧ (A + y) = **0**) ⟶
(x = (y + B) ⟶ (A + x) = B) by (rule MMI˙impcom)
from S24 have (y ∈ ℂ ∧ (A + y) = **0**) ⟶
(x ∈ ℂ ⟶ (x = (y + B) ⟶ (A + x) = B))
by (rule MMI˙a1d)
} then have S25: ∀ x. (y ∈ ℂ ∧ (A + y) = **0**) ⟶
(x ∈ ℂ ⟶ (x = (y + B) ⟶ (A + x) = B)) by auto
from S25 have S26: (y ∈ ℂ ∧ (A + y) = **0**) ⟶
(∀ x ∈ ℂ . (x = (y + B) ⟶ (A + x) = B))
by (rule MMI˙r19˙21aiv)
from S26 have S27: y ∈ ℂ ⟶ ((A + y) = **0** ⟶
(∀ x ∈ ℂ . (x = (y + B) ⟶ (A + x) = B)))
by (rule MMI˙ex)
have S28: (∀ x ∈ ℂ . (x = (y + B) ⟶ (A + x) = B))
⟶ ((∃ x ∈ ℂ . x = (y + B)) ⟶
(∃ x ∈ ℂ . (A + x) = B)) by (rule MMI˙r19˙22)
from S27 S28 have S29: y ∈ ℂ ⟶ ((A + y) = **0** ⟶
((∃ x ∈ ℂ . x = (y + B)) ⟶
(∃ x ∈ ℂ . (A + x) = B))) by (rule MMI˙syl6)
from S10 S29 have y ∈ ℂ ⟶ ((A + y) = **0** ⟶
(∃ x ∈ ℂ . (A + x) = B)) by (rule MMI˙mpid)
} then have S30: ∀ y. y ∈ ℂ ⟶ ((A + y) = **0** ⟶
(∃ x ∈ ℂ . (A + x) = B)) by simp
from S30 have S31: (∃ y ∈ ℂ . (A + y) = **0**) ⟶
(∃ x ∈ ℂ . (A + x) = B) by (rule MMI˙r19˙23aiv)
from S5 S31 have S32: ∃ x ∈ ℂ . (A + x) = B by (rule MMI˙ax˙mp)
from A1 have S33: A ∈ ℂ.
{ fix x y
have S34: (A ∈ ℂ ∧ x ∈ ℂ ∧ y ∈ ℂ) ⟶
((A + x) = (A + y) ⟷ x = y) by (rule MMI˙addcant)
have S35: ((A + x) = B ∧ (A + y) = B) ⟶
(A + x) = (A + y) by (rule MMI˙eqtr3t)
from S34 S35 have S36: (A ∈ ℂ ∧ x ∈ ℂ ∧ y ∈ ℂ) ⟶
(((A + x) = B ∧ (A + y) = B) ⟶ x = y)
by (rule MMI˙syl5bi)

```
    from S33 S36 have ( x ∈ ℂ ∧ y ∈ ℂ ) ⟶
      ( ( ( A + x ) = B ∧ ( A + y ) = B ) ⟶ x = y )
      by (rule MMI˙mp3an1)
  } then have S37: ∀x y . ( x ∈ ℂ ∧ y ∈ ℂ ) ⟶
      ( ( ( A + x ) = B ∧ ( A + y ) = B ) ⟶ x = y ) by auto
  from S37 have S38: ∀ x ∈ ℂ . ∀ y ∈ ℂ . ( ( ( A + x ) = B ∧
    ( A + y ) = B ) ⟶ x = y ) by (rule MMI˙rgen2)
  from S3 S32 S38 show ∃! x . x ∈ ℂ ∧ ( A + x ) = B
    by (rule MMI˙mpbir2an)
qed
```

```
lemma (in MMIsar0) MMI˙subval: assumes A ∈ ℂ  B ∈ ℂ
  shows A - B = ⋃ − x ∈ ℂ . B + x = A ″
  using sub˙def by simp
```

```
lemma (in MMIsar0) MMI˙df˙neg: shows (- A) = 𝟬 - A
  using cneg˙def by simp
```

```
lemma (in MMIsar0) MMI˙negeq:
  shows A = B ⟶ (-A) = (- B)
proof -
  have S1: A = B ⟶ ( 𝟬 - A ) = ( 𝟬 - B ) by (rule MMI˙opreq2)
  have S2: (-A) = ( 𝟬 - A ) by (rule MMI˙df˙neg)
  have S3: (-B) = ( 𝟬 - B ) by (rule MMI˙df˙neg)
  from S1 S2 S3 show A = B ⟶ (-A) = (-B) by (rule MMI˙3eqtr4g)
qed

lemma (in MMIsar0) MMI˙negeqi: assumes A1: A = B
  shows (-  A) = (-B)
proof -
  from A1 have S1: A = B.
  have S2: A = B ⟶ (-A) = (-B) by (rule MMI˙negeq)
  from S1 S2 show (-A) = (-B) by (rule MMI˙ax˙mp)
qed

lemma (in MMIsar0) MMI˙negeqd: assumes A1: φ ⟶ A = B
  shows φ ⟶ (-A) = (-B)
proof -
  from A1 have S1: φ ⟶ A = B.
  have S2: A = B ⟶ (-A) = (-B) by (rule MMI˙negeq)
  from S1 S2 show φ ⟶ (-A) = (-B) by (rule MMI˙syl)
qed
```

lemma (in MMIsar0) MMI˙hbneg: assumes A1: y ∈ A ⟶ (∀ x . y ∈ A)
shows y ∈ ((- A)) ⟶ (∀ x . (y ∈ ((- A))))
using assms by auto

lemma (in MMIsar0) MMI˙minusex:
shows ((- A)) isASet by auto

lemma (in MMIsar0) MMI˙subcl: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows (A - B) ∈ ℂ
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from S1 S2 have S3: (A - B) = ⋃ − x ∈ ℂ . (B + x) = A ″
by (rule MMI˙subval)
from A2 have S4: B ∈ ℂ.
from A1 have S5: A ∈ ℂ.
from S4 S5 have S6: ∃! x . x ∈ ℂ ∧ (B + x) = A by (rule MMI˙negeu)
have S7: (∃! x . x ∈ ℂ ∧ (B + x) = A) ⟶
⋃ − x ∈ ℂ . (B + x) = A ″ ∈ ℂ by (rule MMI˙reucl)
from S6 S7 have S8: ⋃ − x ∈ ℂ . (B + x) = A ″ ∈ ℂ
by (rule MMI˙ax˙mp)
from S3 S8 show (A - B) ∈ ℂ by simp
qed

lemma (in MMIsar0) MMI˙subclt:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A - B) ∈ ℂ
proof -
have S1: A = if (A ∈ ℂ , A , **0**) ⟶ (A - B) =
(if (A ∈ ℂ , A , **0**) - B) by (rule MMI˙opreq1)
from S1 have S2: A = if (A ∈ ℂ , A , **0**) ⟶ ((A - B) ∈ ℂ ⟷
(if (A ∈ ℂ , A , **0**) - B) ∈ ℂ) by (rule MMI˙eleq1d)
have S3: B = if (B ∈ ℂ , B , **0**) ⟶ (if (A ∈ ℂ , A , **0**) - B) =
(if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
from S3 have S4: B = if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) - B) ∈ ℂ ⟷
(if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**)) ∈ ℂ)
by (rule MMI˙eleq1d)
have S5: **0** ∈ ℂ by (rule MMI˙0cn)
from S5 have S6: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S7: **0** ∈ ℂ by (rule MMI˙0cn)
from S7 have S8: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
from S6 S8 have S9:
(if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**)) ∈ ℂ
by (rule MMI˙subcl)

```
  from S2 S4 S9 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A - B ) ∈ ℂ
    by (rule MMI˙dedth2h)
qed

lemma (in MMIsar0) MMI˙negclt:
  shows A ∈ ℂ ⟶ ( (-  A) ) ∈ ℂ
proof -
  have S1: 𝟬 ∈ ℂ by (rule MMI˙0cn)
  have S2: ( 𝟬 ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( 𝟬 - A ) ∈ ℂ by (rule MMI˙subclt)
  from S1 S2 have S3: A ∈ ℂ ⟶ ( 𝟬 - A ) ∈ ℂ by (rule MMI˙mpan)
  have S4: ( (-  A) ) = ( 𝟬 - A ) by (rule MMI˙df˙neg)
  from S3 S4 show A ∈ ℂ ⟶ ( (-  A) ) ∈ ℂ by (rule MMI˙syl5eqel)
qed

lemma (in MMIsar0) MMI˙negcl: assumes A1: A ∈ ℂ
  shows ( (-  A) ) ∈ ℂ
proof -
  from A1 have S1: A ∈ ℂ.
  have S2: A ∈ ℂ ⟶ ( (-  A) ) ∈ ℂ by (rule MMI˙negclt)
  from S1 S2 show ( (-  A) ) ∈ ℂ by (rule MMI˙ax˙mp)
qed

lemma (in MMIsar0) MMI˙subadd: assumes A1: A ∈ ℂ and
   A2: B ∈ ℂ and
   A3: C ∈ ℂ
  shows ( A - B ) = C ⟷ ( B + C ) = A
proof -
  from A3 have S1: C ∈ ℂ.
  { fix x
    have S2: x = C ⟶ ( ( A - B ) = x ⟷ ( A - B ) = C )
      by (rule MMI˙eqeq2)
    have S3: x = C ⟶ ( B + x ) = ( B + C ) by (rule MMI˙opreq2)
    from S3 have S4: x = C ⟶ ( ( B + x ) = A ⟷ ( B + C ) = A )
      by (rule MMI˙eqeq1d)
    from S2 S4 have x = C ⟶ ( ( ( A - B ) = x ⟷
      ( B + x ) = A ) ⟷ ( ( A - B ) = C ⟷ ( B + C ) = A ) )
      by (rule MMI˙bibi12d)
  } then have S5: ∀x. x = C ⟶ ( ( ( A - B ) = x ⟷
      ( B + x ) = A ) ⟷ ( ( A - B ) = C ⟷
      ( B + C ) = A ) ) by simp
  from A2 have S6: B ∈ ℂ.
  from A1 have S7: A ∈ ℂ.
  from S6 S7 have S8: ∃! x . x ∈ ℂ ∧ ( B + x ) = A by (rule MMI˙negeu)
  { fix x
    have S9: ( x ∈ ℂ ∧ ( ∃! x . x ∈ ℂ ∧ ( B + x ) = A ) ⟶
      ( ( B + x ) = A ) ⟷ ⋃ − x ∈ ℂ . ( B + x ) = A ″ = x )
      by (rule MMI˙reuuni1)
    from S8 S9 have x ∈ ℂ ⟶ ( ( B + x ) = A ⟷
      ⋃ − x ∈ ℂ . ( B + x ) = A ″ = x ) by (rule MMI˙mpan2)
```

} then have S10: ∀ x. x ∈ ℂ ⟶ ((B + x) = A ⟷
⋃ − x ∈ ℂ . (B + x) = A ″ = x) by blast
from A1 have S11: A ∈ ℂ.
from A2 have S12: B ∈ ℂ.
from S11 S12 have S13: (A - B) = ⋃ − x ∈ ℂ . (B + x) = A ″
by (rule MMI˙subval)
from S13 have S14: ∀x. (A - B) = x ⟷
⋃ − x ∈ ℂ . (B + x) = A ″ = x by simp
from S10 S14 have S15: ∀x. x ∈ ℂ ⟶ ((A - B) = x ⟷
(B + x) = A) by (rule MMI˙syl6rbbr)
from S5 S15 have S16: C ∈ ℂ ⟶ ((A - B) = C ⟷
(B + C) = A) by (rule MMI˙vtoclga)
from S1 S16 show (A - B) = C ⟷ (B + C) = A
by (rule MMI˙ax˙mp)
qed

lemma (in MMIsar0) MMI˙subsub23: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows (A - B) = C ⟷ (A - C) = B
proof -
from A2 have S1: B ∈ ℂ.
from A3 have S2: C ∈ ℂ.
from S1 S2 have S3: (B + C) = (C + B) by (rule MMI˙addcom)
from S3 have S4: (B + C) = A ⟷ (C + B) = A
by (rule MMI˙eqeq1i)
from A1 have S5: A ∈ ℂ.
from A2 have S6: B ∈ ℂ.
from A3 have S7: C ∈ ℂ.
from S5 S6 S7 have S8: (A - B) = C ⟷ (B + C) = A
by (rule MMI˙subadd)
from A1 have S9: A ∈ ℂ.
from A3 have S10: C ∈ ℂ.
from A2 have S11: B ∈ ℂ.
from S9 S10 S11 have S12: (A - C) = B ⟷ (C + B) = A
by (rule MMI˙subadd)
from S4 S8 S12 show (A - B) = C ⟷ (A - C) = B
by (rule MMI˙3bitr4)
qed

lemma (in MMIsar0) MMI˙subaddt:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A - B) = C ⟷
(B + C) = A)
proof -
have S1: A = if (A ∈ ℂ , A , **0**) ⟶ (A - B) =
(if (A ∈ ℂ , A , **0**) - B) by (rule MMI˙opreq1)

from S1 have S2: A = if (A ∈ ℂ , A , **0**) ⟶ ((A - B) = C ⟷
(if (A ∈ ℂ , A , **0**) - B) = C) by (rule MMI˙eqeq1d)
have S3: A = if (A ∈ ℂ , A , **0**) ⟶ ((B + C) = A ⟷
(B + C) = if (A ∈ ℂ , A , **0**)) by (rule MMI˙eqeq2)
from S2 S3 have S4: A = if (A ∈ ℂ , A , **0**) ⟶
(((A - B) = C ⟷ (B + C) = A) ⟷
((if (A ∈ ℂ , A , **0**) - B) = C ⟷ (B + C) =
if (A ∈ ℂ , A , **0**))) by (rule MMI˙bibi12d)
have S5: B = if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) - B) =
(if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
from S5 have S6: B = if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) - B) = C ⟷
(if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**)) = C)
by (rule MMI˙eqeq1d)
have S7: B = if (B ∈ ℂ , B , **0**) ⟶ (B + C) =
(if (B ∈ ℂ , B , **0**) + C) by (rule MMI˙opreq1)
from S7 have S8: B = if (B ∈ ℂ , B , **0**) ⟶
((B + C) = if (A ∈ ℂ , A , **0**) ⟷
(if (B ∈ ℂ , B , **0**) + C) = if (A ∈ ℂ , A , **0**))
by (rule MMI˙eqeq1d)
from S6 S8 have S9: B = if (B ∈ ℂ , B , **0**) ⟶
(((if (A ∈ ℂ , A , **0**) - B) = C ⟷
(B + C) = if (A ∈ ℂ , A , **0**)) ⟷
((if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**)) = C ⟷
(if (B ∈ ℂ , B , **0**) + C) = if (A ∈ ℂ , A , **0**)))
by (rule MMI˙bibi12d)
have S10: C = if (C ∈ ℂ , C , **0**) ⟶
((if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**)) = C ⟷
(if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**)) =
if (C ∈ ℂ , C , **0**)) by (rule MMI˙eqeq2)
have S11: C = if (C ∈ ℂ , C , **0**) ⟶
(if (B ∈ ℂ , B , **0**) + C) =
(if (B ∈ ℂ , B , **0**) + if (C ∈ ℂ , C , **0**)) by (rule MMI˙opreq2)
from S11 have S12: C = if (C ∈ ℂ , C , **0**) ⟶
((if (B ∈ ℂ , B , **0**) + C) = if (A ∈ ℂ , A , **0**) ⟷
(if (B ∈ ℂ , B , **0**) + if (C ∈ ℂ , C , **0**)) =
if (A ∈ ℂ , A , **0**)) by (rule MMI˙eqeq1d)
from S10 S12 have S13: C = if (C ∈ ℂ , C , **0**) ⟶
(((if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**)) = C ⟷
(if (B ∈ ℂ , B , **0**) + C) = if (A ∈ ℂ , A , **0**)) ⟷
((if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**)) =
if (C ∈ ℂ , C , **0**) ⟷
(if (B ∈ ℂ , B , **0**) + if (C ∈ ℂ , C , **0**)) =
if (A ∈ ℂ , A , **0**))) by (rule MMI˙bibi12d)
have S14: **0** ∈ ℂ by (rule MMI˙0cn)
from S14 have S15: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S16: **0** ∈ ℂ by (rule MMI˙0cn)
from S16 have S17: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)

have S18: $\mathbf{0} \in \mathbb{C}$ by (rule MMI˙0cn)
from S18 have S19: if ($C \in \mathbb{C}$, C , $\mathbf{0}$) $\in \mathbb{C}$ by (rule MMI˙elimel)
from S15 S17 S19 have S20:
(if ($A \in \mathbb{C}$, A , $\mathbf{0}$) - if ($B \in \mathbb{C}$, B , $\mathbf{0}$)) =
if ($C \in \mathbb{C}$, C , $\mathbf{0}$) $\longleftrightarrow$
(if ($B \in \mathbb{C}$, B , $\mathbf{0}$) + if ($C \in \mathbb{C}$, C , $\mathbf{0}$)) =
if ($A \in \mathbb{C}$, A , $\mathbf{0}$) by (rule MMI˙subadd)
from S4 S9 S13 S20 show ($A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}$) $\longrightarrow$
((A - B) = C $\longleftrightarrow$ (B + C) = A) by (rule MMI˙dedth3h)
qed

lemma (in MMIsar0) MMI˙pncan3t:
shows ($A \in \mathbb{C} \wedge B \in \mathbb{C}$) $\longrightarrow$ (A + (B - A)) = B
proof -
have S1: (B - A) = (B - A) by (rule MMI˙eqid)
have S2: ($B \in \mathbb{C} \wedge A \in \mathbb{C} \wedge$ (B - A) $\in \mathbb{C}$) $\longrightarrow$
((B - A) = (B - A) $\longleftrightarrow$ (A + (B - A)) = B)
by (rule MMI˙subaddt)
have S3: ($A \in \mathbb{C} \wedge B \in \mathbb{C}$) $\longrightarrow$ $B \in \mathbb{C}$ by (rule MMI˙pm3˙27)
have S4: ($A \in \mathbb{C} \wedge B \in \mathbb{C}$) $\longrightarrow$ $A \in \mathbb{C}$ by (rule MMI˙pm3˙26)
have S5: ($B \in \mathbb{C} \wedge A \in \mathbb{C}$) $\longrightarrow$ (B - A) $\in \mathbb{C}$ by (rule MMI˙subclt)
from S5 have S6: ($A \in \mathbb{C} \wedge B \in \mathbb{C}$) $\longrightarrow$ (B - A) $\in \mathbb{C}$
by (rule MMI˙ancoms)
from S2 S3 S4 S6 have S7: ($A \in \mathbb{C} \wedge B \in \mathbb{C}$) $\longrightarrow$ ((B - A) =
(B - A) $\longleftrightarrow$ (A + (B - A)) = B) by (rule MMI˙syl3anc)
from S1 S7 show ($A \in \mathbb{C} \wedge B \in \mathbb{C}$) $\longrightarrow$ (A + (B - A)) = B
by (rule MMI˙mpbii)
qed

lemma (in MMIsar0) MMI˙pncan3: assumes A1: $A \in \mathbb{C}$ and
A2: $B \in \mathbb{C}$
shows (A + (B - A)) = B
proof -
from A1 have S1: $A \in \mathbb{C}$.
from A2 have S2: $B \in \mathbb{C}$.
have S3: ($A \in \mathbb{C} \wedge B \in \mathbb{C}$) $\longrightarrow$ (A + (B - A)) = B
by (rule MMI˙pncan3t)
from S1 S2 S3 show (A + (B - A)) = B by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙negidt:
shows $A \in \mathbb{C}$ $\longrightarrow$ (A + ((- A))) = $\mathbf{0}$
proof -
have S1: $\mathbf{0} \in \mathbb{C}$ by (rule MMI˙0cn)
have S2: ($A \in \mathbb{C} \wedge \mathbf{0} \in \mathbb{C}$) $\longrightarrow$ (A + ($\mathbf{0}$ - A)) = $\mathbf{0}$
by (rule MMI˙pncan3t)
from S1 S2 have S3: $A \in \mathbb{C}$ $\longrightarrow$ (A + ($\mathbf{0}$ - A)) = $\mathbf{0}$
by (rule MMI˙mpan2)
have S4: ((- A)) = ($\mathbf{0}$ - A) by (rule MMI˙df˙neg)

from S4 have S5: (A + ((- A))) = (A + (**0** - A))
by (rule MMI˙opreq2i)
from S3 S5 show A ∈ ℂ ⟶ (A + ((- A))) = **0** by (rule MMI˙syl5eq)
qed

lemma (in MMIsar0) MMI˙negid: assumes A1: A ∈ ℂ
shows (A + ((- A))) = **0**
proof -
from A1 have S1: A ∈ ℂ.
have S2: A ∈ ℂ ⟶ (A + ((- A))) = **0** by (rule MMI˙negidt)
from S1 S2 show (A + ((- A))) = **0** by (rule MMI˙ax˙mp)
qed

lemma (in MMIsar0) MMI˙negsub: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows (A + ((- B))) = (A - B)
proof -
from A2 have S1: B ∈ ℂ.
from A1 have S2: A ∈ ℂ.
from A2 have S3: B ∈ ℂ.
from S3 have S4: ((- B)) ∈ ℂ by (rule MMI˙negcl)
from S2 S4 have S5: (A + ((- B))) ∈ ℂ by (rule MMI˙addcl)
from S1 S5 have S6: (B + (A + ((- B)))) =
((A + ((- B))) + B) by (rule MMI˙addcom)
from A1 have S7: A ∈ ℂ.
from S4 have S8: ((- B)) ∈ ℂ .
from A2 have S9: B ∈ ℂ.
from S7 S8 S9 have S10: ((A + ((- B))) + B) =
(A + (((- B)) + B)) by (rule MMI˙addass)
from S4 have S11: ((- B)) ∈ ℂ .
from A2 have S12: B ∈ ℂ.
from S11 S12 have S13: (((- B)) + B) = (B + ((- B)))
by (rule MMI˙addcom)
from A2 have S14: B ∈ ℂ.
from S14 have S15: (B + ((- B))) = **0** by (rule MMI˙negid)
from S13 S15 have S16: (((- B)) + B) = **0** by (rule MMI˙eqtr)
from S16 have S17: (A + (((- B)) + B)) = (A + **0**)
by (rule MMI˙opreq2i)
from A1 have S18: A ∈ ℂ.
from S18 have S19: (A + **0**) = A by (rule MMI˙addid1)
from S10 S17 S19 have S20: ((A + ((- B))) + B) = A
by (rule MMI˙3eqtr)
from S6 S20 have S21: (B + (A + ((- B)))) = A
by (rule MMI˙eqtr)
from A1 have S22: A ∈ ℂ.
from A2 have S23: B ∈ ℂ.
from S5 have S24: (A + ((- B))) ∈ ℂ .
from S22 S23 S24 have S25: (A - B) = (A + ((- B))) ⟷
(B + (A + ((- B)))) = A by (rule MMI˙subadd)

from S21 S25 have S26: (A - B) = (A + ((- B)))
by (rule MMI˙mpbir)
from S26 show (A + ((- B))) = (A - B) by (rule MMI˙eqcomi)
qed

lemma (in MMIsar0) MMI˙negsubt:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + ((- B))) = (A - B)
proof -
have S1: A = if (A ∈ ℂ , A , **0**) ⟶ (A + ((- B))) =
(if (A ∈ ℂ , A , **0**) + ((- B))) by (rule MMI˙opreq1)
have S2: A = if (A ∈ ℂ , A , **0**) ⟶ (A - B) =
(if (A ∈ ℂ , A , **0**) - B) by (rule MMI˙opreq1)
from S1 S2 have S3: A = if (A ∈ ℂ , A , **0**) ⟶
((A + ((- B))) = (A - B) ⟷
(if (A ∈ ℂ , A , **0**) + ((- B))) =
(if (A ∈ ℂ , A , **0**) - B)) by (rule MMI˙eqeq12d)
have S4: B = if (B ∈ ℂ , B , **0**) ⟶
((- B)) = (- if (B ∈ ℂ , B , **0**)) by (rule MMI˙negeq)
from S4 have S5: B = if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) + ((- B))) =
(if (A ∈ ℂ , A , **0**) + (- if (B ∈ ℂ , B , **0**)))
by (rule MMI˙opreq2d)
have S6: B = if (B ∈ ℂ , B , **0**) ⟶ (if (A ∈ ℂ , A , **0**) - B) =
(if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**))
by (rule MMI˙opreq2)
from S5 S6 have S7: B = if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) + ((- B))) =
(if (A ∈ ℂ , A , **0**) - B) ⟷
(if (A ∈ ℂ , A , **0**) + (- if (B ∈ ℂ , B , **0**))) =
(if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**)))
by (rule MMI˙eqeq12d)
have S8: **0** ∈ ℂ by (rule MMI˙0cn)
from S8 have S9: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S10: **0** ∈ ℂ by (rule MMI˙0cn)
from S10 have S11: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
from S9 S11 have S12:
(if (A ∈ ℂ , A , **0**) + (- if (B ∈ ℂ , B , **0**))) =
(if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**))
by (rule MMI˙negsub)
from S3 S7 S12 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + ((- B))) =
(A - B) by (rule MMI˙dedth2h)
qed

lemma (in MMIsar0) MMI˙addsubasst:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + B) - C) =
(A + (B - C))
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ ∧ (- C) ∈ ℂ) ⟶
((A + B) + (- C)) =

(A + (B + (- C))) by (rule MMI˙axaddass)
have S2: C ∈ ℂ ⟶ (- C) ∈ ℂ by (rule MMI˙negclt)
from S1 S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) + (- C)) =
(A + (B + (- C))) by (rule MMI˙syl3an3)
have S4: ((A + B) ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) + (- C)) = ((A + B) - C)
by (rule MMI˙negsubt)
have S5: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + B) ∈ ℂ by (rule MMI˙axaddcl)
from S4 S5 have S6: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ C ∈ ℂ) ⟶
((A + B) + (- C)) = ((A + B) - C)
by (rule MMI˙sylan)
from S6 have S7: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) + (- C)) = ((A + B) - C)
by (rule MMI˙3impa)
have S8: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B + (- C)) = (B - C)
by (rule MMI˙negsubt)
from S8 have S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B + (- C)) = (B - C) by (rule MMI˙3adant1)
from S9 have S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (B + (- C))) = (A + (B - C))
by (rule MMI˙opreq2d)
from S3 S7 S10 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - C) = (A + (B - C))
by (rule MMI˙3eqtr3d)
qed

lemma (in MMIsar0) MMI˙addsubt:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + B) - C) =
((A - C) + B)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + B) = (B + A)
by (rule MMI˙axaddcom)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A + B) - C) =
((B + A) - C) by (rule MMI˙opreq1d)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - C) = ((B + A) - C)
by (rule MMI˙3adant3)
have S4: (B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ) ⟶ ((B + A) - C) =
(B + (A - C)) by (rule MMI˙addsubasst)
from S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((B + A) - C) = (B + (A - C)) by (rule MMI˙3com12)
have S6: (B ∈ ℂ ∧ (A - C) ∈ ℂ) ⟶ (B + (A - C)) =
((A - C) + B) by (rule MMI˙axaddcom)
from S6 have S7: B ∈ ℂ ⟶ ((A - C) ∈ ℂ ⟶
(B + (A - C)) = ((A - C) + B)) by (rule MMI˙ex)
have S8: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A - C) ∈ ℂ by (rule MMI˙subclt)
from S7 S8 have S9: B ∈ ℂ ⟶ ((A ∈ ℂ ∧ C ∈ ℂ) ⟶
(B + (A - C)) = ((A - C) + B)) by (rule MMI˙syl5)

from S9 have S10: B ∈ ℂ ⟶ (A ∈ ℂ ⟶ (C ∈ ℂ ⟶
(B + (A - C)) = ((A - C) + B)))
by (rule MMI˙exp3a)
from S10 have S11: A ∈ ℂ ⟶ (B ∈ ℂ ⟶ (C ∈ ℂ ⟶
(B + (A - C)) = ((A - C) + B)))
by (rule MMI˙com12)
from S11 have S12: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B + (A - C)) = ((A - C) + B) by (rule MMI˙3imp)
from S3 S5 S12 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - C) = ((A - C) + B) by (rule MMI˙3eqtrd)
qed

lemma (in MMIsar0) MMI˙addsub12t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A + (B - C)) =
(B + (A - C))
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + B) = (B + A)
by (rule MMI˙axaddcom)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A + B) - C) =
((B + A) - C) by (rule MMI˙opreq1d)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - C) = ((B + A) - C)
by (rule MMI˙3adant3)
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + B) - C) =
(A + (B - C)) by (rule MMI˙addsubasst)
have S5: (B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ) ⟶ ((B + A) - C) =
(B + (A - C)) by (rule MMI˙addsubasst)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((B + A) - C) = (B + (A - C)) by (rule MMI˙3com12)
from S3 S4 S6 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (B - C)) = (B + (A - C))
by (rule MMI˙3eqtr3d)
qed

lemma (in MMIsar0) MMI˙addsubass: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows ((A + B) - C) = (A + (B - C))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + B) - C) =
(A + (B - C)) by (rule MMI˙addsubasst)
from S1 S2 S3 S4 show ((A + B) - C) =
(A + (B - C)) by (rule MMI˙mp3an)
qed

lemma (in MMIsar0) MMI˙addsub: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows ((A + B) - C) = ((A - C) + B)
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + B) - C) =
((A - C) + B) by (rule MMI˙addsubt)
from S1 S2 S3 S4 show ((A + B) - C) =
((A - C) + B) by (rule MMI˙mp3an)
qed

lemma (in MMIsar0) MMI˙2addsubt:
shows ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A + B) + C) - D) = (((A + C) - D) + B)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + B) + C) =
((A + C) + B) by (rule MMI˙add23t)
from S1 have S2: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ C ∈ ℂ) ⟶
((A + B) + C) = ((A + C) + B) by (rule MMI˙3expa)
from S2 have S3: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) + C) = ((A + C) + B)
by (rule MMI˙adantrr)
from S3 have S4: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A + B) + C) - D) =
(((A + C) + B) - D) by (rule MMI˙opreq1d)
have S5: ((A + C) ∈ ℂ ∧ B ∈ ℂ ∧ D ∈ ℂ) ⟶
(((A + C) + B) - D) =
(((A + C) - D) + B) by (rule MMI˙addsubt)
from S5 have S6: ((A + C) ∈ ℂ ∧ (B ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A + C) + B) - D) =
(((A + C) - D) + B) by (rule MMI˙3expb)
have S7: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A + C) ∈ ℂ by (rule MMI˙axaddcl)
from S6 S7 have S8: ((A ∈ ℂ ∧ C ∈ ℂ) ∧ (B ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A + C) + B) - D) =
(((A + C) - D) + B) by (rule MMI˙sylan)
from S8 have S9: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A + C) + B) - D) =
(((A + C) - D) + B) by (rule MMI˙an4s)
from S4 S9 show ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A + B) + C) - D) =
(((A + C) - D) + B) by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙negneg: assumes A1: A ∈ ℂ
shows (- ((- A))) = A

```
proof -
  from A1 have S1: A ∈ ℂ.
  from S1 have S2: ( (-  A) ) ∈ ℂ by (rule MMI˙negcl)
  from S2 have S3: ( ( (-  A) ) + ( - ( (-  A) ) ) ) = 𝟬
    by (rule MMI˙negid)
  from S3 have S4: ( A + ( ( (-  A) ) + ( - ( (-  A) ) ) ) ) =
    ( A + 𝟬 ) by (rule MMI˙opreq2i)
  from A1 have S5: A ∈ ℂ.
  from S5 have S6: ( A + ( (-  A) ) ) = 𝟬 by (rule MMI˙negid)
  from S6 have S7: ( ( A + ( (-  A) ) ) + ( - ( (-  A) ) ) ) =
    ( 𝟬 + ( - ( (-  A) ) ) ) by (rule MMI˙opreq1i)
  from A1 have S8: A ∈ ℂ.
  from S2 have S9: ( (-  A) ) ∈ ℂ .
  from S2 have S10: ( (-  A) ) ∈ ℂ .
  from S10 have S11: ( - ( (-  A) ) ) ∈ ℂ by (rule MMI˙negcl)
  from S8 S9 S11 have S12:
    ( ( A + ( (-  A) ) ) + ( - ( (-  A) ) ) ) =
    ( A + ( ( (-  A) ) + ( - ( (-  A) ) ) ) )
    by (rule MMI˙addass)
  from S11 have S13: ( - ( (-  A) ) ) ∈ ℂ .
  from S13 have S14: ( 𝟬 + ( - ( (-  A) ) ) ) =
    ( - ( (-  A) ) ) by (rule MMI˙addid2)
  from S7 S12 S14 have S15:
    ( A + ( ( (-  A) ) + ( - ( (-  A) ) ) ) ) =
    ( - ( (-  A) ) ) by (rule MMI˙3eqtr3)
  from A1 have S16: A ∈ ℂ.
  from S16 have S17: ( A + 𝟬 ) = A by (rule MMI˙addid1)
  from S4 S15 S17 show ( - ( (-  A) ) ) = A by (rule MMI˙3eqtr3)
qed

lemma (in MMIsar0) MMI˙subid: assumes A1: A ∈ ℂ
  shows ( A - A ) = 𝟬
proof -
  from A1 have S1: A ∈ ℂ.
  from A1 have S2: A ∈ ℂ.
  from S1 S2 have S3: ( A + ( (-  A) ) ) = ( A - A )
    by (rule MMI˙negsub)
  from A1 have S4: A ∈ ℂ.
  from S4 have S5: ( A + ( (-  A) ) ) = 𝟬 by (rule MMI˙negid)
  from S3 S5 show ( A - A ) = 𝟬 by (rule MMI˙eqtr3)
qed

lemma (in MMIsar0) MMI˙subid1: assumes A1: A ∈ ℂ
  shows ( A - 𝟬 ) = A
proof -
  from A1 have S1: A ∈ ℂ.
  from S1 have S2: ( 𝟬 + A ) = A by (rule MMI˙addid2)
  from A1 have S3: A ∈ ℂ.
  have S4: 𝟬 ∈ ℂ by (rule MMI˙0cn)
```

from A1 have S5: A ∈ ℂ.
from S3 S4 S5 have S6: (A - **0**) = A ⟷ (**0** + A) = A
by (rule MMI˙subadd)
from S2 S6 show (A - **0**) = A by (rule MMI˙mpbir)
qed

lemma (in MMIsar0) MMI˙negnegt:
shows A ∈ ℂ ⟶ (- ((- A))) = A
proof -
have S1: A = if (A ∈ ℂ , A , **0**) ⟶ ((- A)) =
(- if (A ∈ ℂ , A , **0**)) by (rule MMI˙negeq)
from S1 have S2: A = if (A ∈ ℂ , A , **0**) ⟶ (- ((- A))) =
(- (- if (A ∈ ℂ , A , **0**))) by (rule MMI˙negeqd)
have S3: A = if (A ∈ ℂ , A , **0**) ⟶ A = if (A ∈ ℂ , A , **0**)
by (rule MMI˙id)
from S2 S3 have S4: A = if (A ∈ ℂ , A , **0**) ⟶
((- ((- A))) = A ⟷
(- (- if (A ∈ ℂ , A , **0**))) = if (A ∈ ℂ , A , **0**))
by (rule MMI˙eqeq12d)
have S5: **0** ∈ ℂ by (rule MMI˙0cn)
from S5 have S6: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
from S6 have S7: (- (- if (A ∈ ℂ , A , **0**))) =
if (A ∈ ℂ , A , **0**) by (rule MMI˙negneg)
from S4 S7 show A ∈ ℂ ⟶ (- ((- A))) = A by (rule MMI˙dedth)
qed

lemma (in MMIsar0) MMI˙subnegt:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A - ((- B))) = (A + B)
proof -
have S1: (A ∈ ℂ ∧ ((- B)) ∈ ℂ) ⟶
(A + (- ((- B)))) = (A - ((- B)))
by (rule MMI˙negsubt)
have S2: B ∈ ℂ ⟶ ((- B)) ∈ ℂ by (rule MMI˙negclt)
from S1 S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A + (- ((- B)))) = (A - ((- B)))
by (rule MMI˙sylan2)
have S4: B ∈ ℂ ⟶ (- ((- B))) = B by (rule MMI˙negnegt)
from S4 have S5: B ∈ ℂ ⟶ (A + (- ((- B)))) =
(A + B) by (rule MMI˙opreq2d)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A + (- ((- B)))) = (A + B) by (rule MMI˙adantl)
from S3 S6 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A - ((- B))) =
(A + B) by (rule MMI˙eqtr3d)
qed

lemma (in MMIsar0) MMI˙subidt:
shows A ∈ ℂ ⟶ (A - A) = **0**
proof -
have S1: (A = if (A ∈ ℂ , A , **0**) ∧ A = if (A ∈ ℂ , A , **0**)) ⟶

(A - A) = (if (A ∈ ℂ , A , **0**) - if (A ∈ ℂ , A , **0**))
by (rule MMI˙opreq12)
from S1 have S2: A = if (A ∈ ℂ , A , **0**) ⟶
(A - A) = (if (A ∈ ℂ , A , **0**) - if (A ∈ ℂ , A , **0**))
by (rule MMI˙anidms)
from S2 have S3: A = if (A ∈ ℂ , A , **0**) ⟶
((A - A) = **0** ⟷
(if (A ∈ ℂ , A , **0**) - if (A ∈ ℂ , A , **0**)) = **0**)
by (rule MMI˙eqeq1d)
have S4: **0** ∈ ℂ by (rule MMI˙0cn)
from S4 have S5: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
from S5 have S6:
(if (A ∈ ℂ , A , **0**) - if (A ∈ ℂ , A , **0**)) = **0**
by (rule MMI˙subid)
from S3 S6 show A ∈ ℂ ⟶ (A - A) = **0** by (rule MMI˙dedth)
qed

lemma (in MMIsar0) MMI˙subid1t:
shows A ∈ ℂ ⟶ (A - **0**) = A
proof -
have S1: A = if (A ∈ ℂ , A , **0**) ⟶ (A - **0**) =
(if (A ∈ ℂ , A , **0**) - **0**) by (rule MMI˙opreq1)
have S2: A = if (A ∈ ℂ , A , **0**) ⟶
A = if (A ∈ ℂ , A , **0**) by (rule MMI˙id)
from S1 S2 have S3: A = if (A ∈ ℂ , A , **0**) ⟶
((A - **0**) = A ⟷ (if (A ∈ ℂ , A , **0**) - **0**) =
if (A ∈ ℂ , A , **0**)) by (rule MMI˙eqeq12d)
have S4: **0** ∈ ℂ by (rule MMI˙0cn)
from S4 have S5: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
from S5 have S6: (if (A ∈ ℂ , A , **0**) - **0**) =
if (A ∈ ℂ , A , **0**) by (rule MMI˙subid1)
from S3 S6 show A ∈ ℂ ⟶ (A - **0**) = A by (rule MMI˙dedth)
qed

lemma (in MMIsar0) MMI˙pncant:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A + B) - B) = A
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A + B) - B) =
(A + (B - B)) by (rule MMI˙addsubasst)
from S1 have S2: (A ∈ ℂ ∧ (B ∈ ℂ ∧ B ∈ ℂ)) ⟶
((A + B) - B) = (A + (B - B)) by (rule MMI˙3expb)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A + B) - B) =
(A + (B - B)) by (rule MMI˙anabsan2)
have S4: B ∈ ℂ ⟶ (B - B) = **0** by (rule MMI˙subidt)
from S4 have S5: B ∈ ℂ ⟶ (A + (B - B)) = (A + **0**)
by (rule MMI˙opreq2d)
have S6: A ∈ ℂ ⟶ (A + **0**) = A by (rule MMI˙ax0id)
from S5 S6 have S7: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + (B - B)) = A

```
    by (rule MMI˙sylan9eqr)
  from S3 S7 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - B ) = A
    by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙pncan2t:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - A ) = B
proof -
  have S1: ( B ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( B + A ) = ( A + B )
    by (rule MMI˙axaddcom)
  from S1 have S2: ( B ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( ( B + A ) - A ) =
    ( ( A + B ) - A ) by (rule MMI˙opreq1d)
  have S3: ( B ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( ( B + A ) - A ) = B
    by (rule MMI˙pncant)
  from S2 S3 have S4: ( B ∈ ℂ ∧ A ∈ ℂ ) ⟶
    ( ( A + B ) - A ) = B by (rule MMI˙eqtr3d)
  from S4 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - A ) = B
    by (rule MMI˙ancoms)
qed

lemma (in MMIsar0) MMI˙npcant:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A - B ) + B ) = A
proof -
  have S1: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ∈ ℂ ) ⟶
    ( ( A + B ) - B ) = ( ( A - B ) + B )
    by (rule MMI˙addsubt)
  from S1 have S2: ( A ∈ ℂ ∧ ( B ∈ ℂ ∧ B ∈ ℂ ) ) ⟶
    ( ( A + B ) - B ) = ( ( A - B ) + B ) by (rule MMI˙3expb)
  from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
    ( ( A + B ) - B ) = ( ( A - B ) + B )
    by (rule MMI˙anabsan2)
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - B ) = A
    by (rule MMI˙pncant)
  from S3 S4 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A - B ) + B ) = A
    by (rule MMI˙eqtr3d)
qed

lemma (in MMIsar0) MMI˙npncant:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A - B ) + ( B - C ) ) = ( A - C )
proof -
  have S1: ( ( A - B ) ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( ( A - B ) + B ) - C ) =
    ( ( A - B ) + ( B - C ) ) by (rule MMI˙addsubasst)
  have S2: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A - B ) ∈ ℂ by (rule MMI˙subclt)
  from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( A - B ) ∈ ℂ by (rule MMI˙3adant3)
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ B ∈ ℂ by (rule MMI˙3simp2)
  have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ C ∈ ℂ by (rule MMI˙3simp3)
```

from S1 S3 S4 S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(((A - B) + B) - C) =
((A - B) + (B - C)) by (rule MMI˙syl3anc)
have S7: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A - B) + B) = A
by (rule MMI˙npcant)
from S7 have S8: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((A - B) + B) - C) = (A - C)
by (rule MMI˙opreq1d)
from S8 have S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(((A - B) + B) - C) = (A - C)
by (rule MMI˙3adant3)
from S6 S9 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) + (B - C)) = (A - C)
by (rule MMI˙eqtr3d)
qed

lemma (in MMIsar0) MMI˙nppcant:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(((A - B) + C) + B) = (A + C)
proof -
have S1: ((A - B) ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ) ⟶
(((A - B) + C) + B) =
(((A - B) + B) + C) by (rule MMI˙add23t)
have S2: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A - B) ∈ ℂ by (rule MMI˙subclt)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A - B) ∈ ℂ
by (rule MMI˙3adant3)
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ C ∈ ℂ by (rule MMI˙3simp3)
have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ B ∈ ℂ by (rule MMI˙3simp2)
from S1 S3 S4 S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(((A - B) + C) + B) =
(((A - B) + B) + C) by (rule MMI˙syl3anc)
have S7: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A - B) + B) = A
by (rule MMI˙npcant)
from S7 have S8: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((A - B) + B) + C) = (A + C)
by (rule MMI˙opreq1d)
from S8 have S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(((A - B) + B) + C) = (A + C)
by (rule MMI˙3adant3)
from S6 S9 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(((A - B) + C) + B) = (A + C) by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙subneg: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows (A - ((- B))) = (A + B)
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.

```
  have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A - ( (-  B) ) ) = ( A + B )
    by (rule MMI˙subnegt)
  from S1 S2 S3 show ( A - ( (-  B) ) ) = ( A + B )
    by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙subeq0: assumes A1: A ∈ ℂ and
   A2: B ∈ ℂ
  shows ( A - B ) = 𝟬 ⟷ A = B
proof -
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  from S1 S2 have S3: ( A + ( (-  B) ) ) = ( A - B )
    by (rule MMI˙negsub)
  from S3 have S4: ( A + ( (-  B) ) ) = 𝟬 ⟷ ( A - B ) = 𝟬
    by (rule MMI˙eqeq1i)
  have S5: ( A + ( (-  B) ) ) = 𝟬 ⟶
    ( ( A + ( (-  B) ) ) + B ) = ( 𝟬 + B ) by (rule MMI˙opreq1)
  from S4 S5 have S6: ( A - B ) = 𝟬 ⟶
    ( ( A + ( (-  B) ) ) + B ) = ( 𝟬 + B ) by (rule MMI˙sylbir)
  from A1 have S7: A ∈ ℂ.
  from A2 have S8: B ∈ ℂ.
  from S8 have S9: ( (-  B) ) ∈ ℂ by (rule MMI˙negcl)
  from A2 have S10: B ∈ ℂ.
  from S7 S9 S10 have S11: ( ( A + ( (-  B) ) ) + B ) =
    ( ( A + B ) + ( (-  B) ) ) by (rule MMI˙add23)
  from A1 have S12: A ∈ ℂ.
  from A2 have S13: B ∈ ℂ.
  from S9 have S14: ( (-  B) ) ∈ ℂ .
  from S12 S13 S14 have S15: ( ( A + B ) + ( (-  B) ) ) =
    ( A + ( B + ( (-  B) ) ) ) by (rule MMI˙addass)
  from A2 have S16: B ∈ ℂ.
  from S16 have S17: ( B + ( (-  B) ) ) = 𝟬 by (rule MMI˙negid)
  from S17 have S18: ( A + ( B + ( (-  B) ) ) ) = ( A + 𝟬 )
    by (rule MMI˙opreq2i)
  from A1 have S19: A ∈ ℂ.
  from S19 have S20: ( A + 𝟬 ) = A by (rule MMI˙addid1)
  from S18 S20 have S21: ( A + ( B + ( (-  B) ) ) ) = A
    by (rule MMI˙eqtr)
  from S11 S15 S21 have S22: ( ( A + ( (-  B) ) ) + B ) = A
    by (rule MMI˙3eqtr)
  from A2 have S23: B ∈ ℂ.
  from S23 have S24: ( 𝟬 + B ) = B by (rule MMI˙addid2)
  from S6 S22 S24 have S25: ( A - B ) = 𝟬 ⟶ A = B
    by (rule MMI˙3eqtr3g)
  have S26: A = B ⟶ ( A - B ) = ( B - B ) by (rule MMI˙opreq1)
  from A2 have S27: B ∈ ℂ.
  from S27 have S28: ( B - B ) = 𝟬 by (rule MMI˙subid)
  from S26 S28 have S29: A = B ⟶ ( A - B ) = 𝟬 by (rule MMI˙syl6eq)
```

from S25 S29 show (A - B) = **0** ⟷ A = B by (rule MMI˙impbi)
qed

lemma (in MMIsar0) MMI˙neg11: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows ((- A)) = ((- B)) ⟷ A = B
proof -
have S1: ((- A)) = (**0** - A) by (rule MMI˙df˙neg)
have S2: ((- B)) = (**0** - B) by (rule MMI˙df˙neg)
from S1 S2 have S3: ((- A)) = ((- B)) ⟷ (**0** - A) =
(**0** - B) by (rule MMI˙eqeq12i)
have S4: **0** ∈ ℂ by (rule MMI˙0cn)
from A1 have S5: A ∈ ℂ.
have S6: **0** ∈ ℂ by (rule MMI˙0cn)
from A2 have S7: B ∈ ℂ.
from S6 S7 have S8: (**0** - B) ∈ ℂ by (rule MMI˙subcl)
from S4 S5 S8 have S9: (**0** - A) = (**0** - B) ⟷
(A + (**0** - B)) = **0** by (rule MMI˙subadd)
from S2 have S10: ((- B)) = (**0** - B) .
from S10 have S11: (A + ((- B))) = (A + (**0** - B))
by (rule MMI˙opreq2i)
from A1 have S12: A ∈ ℂ.
from A2 have S13: B ∈ ℂ.
from S12 S13 have S14: (A + ((- B))) = (A - B)
by (rule MMI˙negsub)
from S11 S14 have S15: (A + (**0** - B)) = (A - B)
by (rule MMI˙eqtr3)
from S15 have S16: (A + (**0** - B)) = **0** ⟷ (A - B) = **0**
by (rule MMI˙eqeq1i)
from A1 have S17: A ∈ ℂ.
from A2 have S18: B ∈ ℂ.
from S17 S18 have S19: (A - B) = **0** ⟷ A = B by (rule MMI˙subeq0)
from S16 S19 have S20: (A + (**0** - B)) = **0** ⟷ A = B
by (rule MMI˙bitr)
from S3 S9 S20 show ((- A)) = ((- B)) ⟷ A = B by (rule MMI˙3bitr)
qed

lemma (in MMIsar0) MMI˙negcon1: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows ((- A)) = B ⟷ ((- B)) = A
proof -
from A1 have S1: A ∈ ℂ.
from S1 have S2: (- ((- A))) = A by (rule MMI˙negneg)
from S2 have S3: (- ((- A))) = ((- B)) ⟷ A = ((- B))
by (rule MMI˙eqeq1i)
from A1 have S4: A ∈ ℂ.
from S4 have S5: ((- A)) ∈ ℂ by (rule MMI˙negcl)

from A2 have S6: B ∈ ℂ.
from S5 S6 have S7: (- ((- A))) =
((- B)) ⟷ ((- A)) = B by (rule MMI˙neg11)
have S8: A = ((- B)) ⟷ ((- B)) = A by (rule MMI˙eqcom)
from S3 S7 S8 show ((- A)) = B ⟷ ((- B)) = A by (rule MMI˙3bitr3)
qed

lemma (in MMIsar0) MMI˙negcon2: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows A = ((- B)) ⟷ B = ((- A))
proof -
from A2 have S1: B ∈ ℂ.
from A1 have S2: A ∈ ℂ.
from S1 S2 have S3: ((- B)) = A ⟷ ((- A)) = B
by (rule MMI˙negcon1)
have S4: A = ((- B)) ⟷ ((- B)) = A by (rule MMI˙eqcom)
have S5: B = ((- A)) ⟷ ((- A)) = B by (rule MMI˙eqcom)
from S3 S4 S5 show A = ((- B)) ⟷ B = ((- A)) by (rule MMI˙3bitr4)
qed

lemma (in MMIsar0) MMI˙neg11t:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (((- A)) = ((- B)) ⟷ A = B)
proof -
have S1: A = if (A ∈ ℂ , A , **0**) ⟶ ((- A)) =
(- if (A ∈ ℂ , A , **0**)) by (rule MMI˙negeq)
from S1 have S2: A = if (A ∈ ℂ , A , **0**) ⟶ (((- A)) =
((- B)) ⟷ (- if (A ∈ ℂ , A , **0**)) = ((- B)))
by (rule MMI˙eqeq1d)
have S3: A = if (A ∈ ℂ , A , **0**) ⟶ (A = B ⟷
if (A ∈ ℂ , A , **0**) = B) by (rule MMI˙eqeq1)
from S2 S3 have S4: A = if (A ∈ ℂ , A , **0**) ⟶
((((- A)) = ((- B)) ⟷ A = B) ⟷
((- if (A ∈ ℂ , A , **0**)) = ((- B)) ⟷
if (A ∈ ℂ , A , **0**) = B)) by (rule MMI˙bibi12d)
have S5: B = if (B ∈ ℂ , B , **0**) ⟶ ((- B)) =
(- if (B ∈ ℂ , B , **0**)) by (rule MMI˙negeq)
from S5 have S6: B = if (B ∈ ℂ , B , **0**) ⟶
((- if (A ∈ ℂ , A , **0**)) = ((- B)) ⟷
(- if (A ∈ ℂ , A , **0**)) = (- if (B ∈ ℂ , B , **0**)))
by (rule MMI˙eqeq2d)
have S7: B = if (B ∈ ℂ , B , **0**) ⟶ (if (A ∈ ℂ , A , **0**) = B ⟷
if (A ∈ ℂ , A , **0**) = if (B ∈ ℂ , B , **0**)) by (rule MMI˙eqeq2)
from S6 S7 have S8: B = if (B ∈ ℂ , B , **0**) ⟶
(((- if (A ∈ ℂ , A , **0**)) = ((- B)) ⟷
if (A ∈ ℂ , A , **0**) = B) ⟷ ((- if (A ∈ ℂ , A , **0**)) =
(- if (B ∈ ℂ , B , **0**)) ⟷ if (A ∈ ℂ , A , **0**) =
if (B ∈ ℂ , B , **0**))) by (rule MMI˙bibi12d)
have S9: **0** ∈ ℂ by (rule MMI˙0cn)
from S9 have S10: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)

have S11: $\mathbf{0} \in \mathbb{C}$ by (rule MMI˙0cn)
from S11 have S12: if (B $\in \mathbb{C}$, B , $\mathbf{0}$) $\in \mathbb{C}$ by (rule MMI˙elimel)
from S10 S12 have S13: (- if (A $\in \mathbb{C}$, A , $\mathbf{0}$)) =
(- if (B $\in \mathbb{C}$, B , $\mathbf{0}$)) $\longleftrightarrow$ if (A $\in \mathbb{C}$, A , $\mathbf{0}$) =
if (B $\in \mathbb{C}$, B , $\mathbf{0}$) by (rule MMI˙neg11)
from S4 S8 S13 show (A $\in \mathbb{C} \wedge$ B $\in \mathbb{C}$) $\longrightarrow$ (((- A)) =
((- B)) $\longleftrightarrow$ A = B) by (rule MMI˙dedth2h)
qed

lemma (in MMIsar0) MMI˙negcon1t:
shows (A $\in \mathbb{C} \wedge$ B $\in \mathbb{C}$) $\longrightarrow$ (((- A)) = B $\longleftrightarrow$ ((- B)) = A)
proof -
have S1: (((- A)) $\in \mathbb{C} \wedge$ B $\in \mathbb{C}$) $\longrightarrow$ ((- ((- A))) =
((- B)) $\longleftrightarrow$ ((- A)) = B) by (rule MMI˙neg11t)
have S2: A $\in \mathbb{C}$ $\longrightarrow$ ((- A)) $\in \mathbb{C}$ by (rule MMI˙negclt)
from S1 S2 have S3: (A $\in \mathbb{C} \wedge$ B $\in \mathbb{C}$) $\longrightarrow$ ((- ((- A))) =
((- B)) $\longleftrightarrow$ ((- A)) = B) by (rule MMI˙sylan)
have S4: A $\in \mathbb{C}$ $\longrightarrow$ (- ((- A))) = A by (rule MMI˙negnegt)
from S4 have S5: (A $\in \mathbb{C} \wedge$ B $\in \mathbb{C}$) $\longrightarrow$ (- ((- A))) = A
by (rule MMI˙adantr)
from S5 have S6: (A $\in \mathbb{C} \wedge$ B $\in \mathbb{C}$) $\longrightarrow$ ((- ((- A))) =
((- B)) $\longleftrightarrow$ A = ((- B))) by (rule MMI˙eqeq1d)
from S3 S6 have S7: (A $\in \mathbb{C} \wedge$ B $\in \mathbb{C}$) $\longrightarrow$ (((- A)) = B $\longleftrightarrow$ A =
((- B))) by (rule MMI˙bitr3d)
have S8: A = ((- B)) $\longleftrightarrow$ ((- B)) = A by (rule MMI˙eqcom)
from S7 S8 show (A $\in \mathbb{C} \wedge$ B $\in \mathbb{C}$) $\longrightarrow$ (((- A)) = B $\longleftrightarrow$
((- B)) = A) by (rule MMI˙syl6bb)
qed

lemma (in MMIsar0) MMI˙negcon2t:
shows (A $\in \mathbb{C} \wedge$ B $\in \mathbb{C}$) $\longrightarrow$ (A = ((- B)) $\longleftrightarrow$ B = ((- A)))
proof -
have S1: (A $\in \mathbb{C} \wedge$ B $\in \mathbb{C}$) $\longrightarrow$ (((- A)) = B $\longleftrightarrow$ ((- B)) = A)
by (rule MMI˙negcon1t)
have S2: A = ((- B)) $\longleftrightarrow$ ((- B)) = A by (rule MMI˙eqcom)
from S1 S2 have S3: (A $\in \mathbb{C} \wedge$ B $\in \mathbb{C}$) $\longrightarrow$ (A = ((- B)) $\longleftrightarrow$
((- A)) = B) by (rule MMI˙syl6rbbrA)
have S4: ((- A)) = B $\longleftrightarrow$ B = ((- A)) by (rule MMI˙eqcom)
from S3 S4 show (A $\in \mathbb{C} \wedge$ B $\in \mathbb{C}$) $\longrightarrow$ (A = ((- B)) $\longleftrightarrow$ B =
((- A))) by (rule MMI˙syl6bb)
qed

lemma (in MMIsar0) MMI˙subcant:
shows (A $\in \mathbb{C} \wedge$ B $\in \mathbb{C} \wedge$ C $\in \mathbb{C}$) $\longrightarrow$ ((A - B) =
(A - C) $\longleftrightarrow$ B = C)
proof -
have S1: (A $\in \mathbb{C} \wedge$ ((- B)) $\in \mathbb{C} \wedge$ (- C) $\in \mathbb{C}$) $\longrightarrow$
((A + ((- B))) = (A + (- C)) $\longleftrightarrow$
((- B)) = (- C)) by (rule MMI˙addcant)

have S2: C ∈ ℂ ⟶ (- C) ∈ ℂ by (rule MMI˙negclt)
from S1 S2 have S3: (A ∈ ℂ ∧ ((- B)) ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + ((- B))) = (A + (- C)) ⟷
((- B)) = (- C)) by (rule MMI˙syl3an3)
have S4: B ∈ ℂ ⟶ ((- B)) ∈ ℂ by (rule MMI˙negclt)
from S3 S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + ((- B))) = (A + (- C)) ⟷
((- B)) = (- C)) by (rule MMI˙syl3an2)
have S6: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + ((- B))) = (A - B)
by (rule MMI˙negsubt)
from S6 have S7: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + ((- B))) = (A - B) by (rule MMI˙3adant3)
have S8: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A + (- C)) = (A - C)
by (rule MMI˙negsubt)
from S8 have S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (- C)) = (A - C) by (rule MMI˙3adant2)
from S7 S9 have S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + ((- B))) = (A + (- C)) ⟷
(A - B) = (A - C)) by (rule MMI˙eqeq12d)
have S11: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (((- B)) = (- C) ⟷ B = C)
by (rule MMI˙neg11t)
from S11 have S12: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(((- B)) = (- C) ⟷ B = C) by (rule MMI˙3adant1)
from S5 S10 S12 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) = (A - C) ⟷ B = C) by (rule MMI˙3bitr3d)
qed

lemma (in MMIsar0) MMI˙subcan2t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - C) = (B - C) ⟷ A = B)
proof -
have S1: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A + (- C)) = (A - C)
by (rule MMI˙negsubt)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (- C)) = (A - C) by (rule MMI˙3adant2)
have S3: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B + (- C)) = (B - C)
by (rule MMI˙negsubt)
from S3 have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B + (- C)) = (B - C) by (rule MMI˙3adant1)
from S2 S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + (- C)) = (B + (- C)) ⟷ (A - C) =
(B - C)) by (rule MMI˙eqeq12d)
have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ (- C) ∈ ℂ) ⟶
((A + (- C)) = (B + (- C)) ⟷ A = B)
by (rule MMI˙addcan2t)
have S7: C ∈ ℂ ⟶ (- C) ∈ ℂ by (rule MMI˙negclt)
from S6 S7 have S8: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + (- C)) = (B + (- C)) ⟷ A = B)
by (rule MMI˙syl3an3)

from S5 S8 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - C) = (B - C) ⟷ A = B) by (rule MMI˙bitr3d)
qed

lemma (in MMIsar0) MMI˙subcan: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows (A - B) = (A - C) ⟷ B = C
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A - B) = (A - C) ⟷ B = C)
by (rule MMI˙subcant)
from S1 S2 S3 S4 show (A - B) = (A - C) ⟷ B = C
by (rule MMI˙mp3an)
qed

lemma (in MMIsar0) MMI˙subcan2: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows (A - C) = (B - C) ⟷ A = B
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - C) = (B - C) ⟷ A = B) by (rule MMI˙subcan2t)
from S1 S2 S3 S4 show (A - C) = (B - C) ⟷ A = B
by (rule MMI˙mp3an)
qed

lemma (in MMIsar0) MMI˙subeq0t:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A - B) = **0** ⟷ A = B)
proof -
have S1: A = if (A ∈ ℂ , A , **0**) ⟶ (A - B) =
(if (A ∈ ℂ , A , **0**) - B) by (rule MMI˙opreq1)
from S1 have S2: A = if (A ∈ ℂ , A , **0**) ⟶ ((A - B) = **0** ⟷
(if (A ∈ ℂ , A , **0**) - B) = **0**) by (rule MMI˙eqeq1d)
have S3: A = if (A ∈ ℂ , A , **0**) ⟶ (A = B ⟷
if (A ∈ ℂ , A , **0**) = B) by (rule MMI˙eqeq1)
from S2 S3 have S4: A = if (A ∈ ℂ , A , **0**) ⟶
(((A - B) = **0** ⟷ A = B) ⟷
((if (A ∈ ℂ , A , **0**) - B) = **0** ⟷
if (A ∈ ℂ , A , **0**) = B)) by (rule MMI˙bibi12d)
have S5: B = if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) - B) =
(if (A ∈ ℂ , A , **0**) - if (B ∈ ℂ , B , **0**))
by (rule MMI˙opreq2)

```
  from S5 have S6: B = if ( B ∈ ℂ , B , 𝟬 ) ⟶
    ( ( if ( A ∈ ℂ , A , 𝟬 ) - B ) = 𝟬 ⟷
    ( if ( A ∈ ℂ , A , 𝟬 ) - if ( B ∈ ℂ , B , 𝟬 ) ) = 𝟬 )
    by (rule MMI˙eqeq1d)
  have S7: B = if ( B ∈ ℂ , B , 𝟬 ) ⟶ ( if ( A ∈ ℂ , A , 𝟬 ) = B ⟷
    if ( A ∈ ℂ , A , 𝟬 ) = if ( B ∈ ℂ , B , 𝟬 ) ) by (rule MMI˙eqeq2)
  from S6 S7 have S8: B = if ( B ∈ ℂ , B , 𝟬 ) ⟶
    ( ( ( if ( A ∈ ℂ , A , 𝟬 ) - B ) = 𝟬 ⟷
    if ( A ∈ ℂ , A , 𝟬 ) = B ) ⟷
    ( ( if ( A ∈ ℂ , A , 𝟬 ) - if ( B ∈ ℂ , B , 𝟬 ) ) = 𝟬 ⟷
    if ( A ∈ ℂ , A , 𝟬 ) = if ( B ∈ ℂ , B , 𝟬 ) ) )
    by (rule MMI˙bibi12d)
  have S9: 𝟬 ∈ ℂ by (rule MMI˙0cn)
  from S9 have S10: if ( A ∈ ℂ , A , 𝟬 ) ∈ ℂ by (rule MMI˙elimel)
  have S11: 𝟬 ∈ ℂ by (rule MMI˙0cn)
  from S11 have S12: if ( B ∈ ℂ , B , 𝟬 ) ∈ ℂ by (rule MMI˙elimel)
  from S10 S12 have S13:
    ( if ( A ∈ ℂ , A , 𝟬 ) - if ( B ∈ ℂ , B , 𝟬 ) ) = 𝟬 ⟷
    if ( A ∈ ℂ , A , 𝟬 ) = if ( B ∈ ℂ , B , 𝟬 )
    by (rule MMI˙subeq0)
  from S4 S8 S13 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
    ( ( A - B ) = 𝟬 ⟷ A = B ) by (rule MMI˙dedth2h)
qed



lemma (in MMIsar0) MMI˙neg0:
  shows ( - 𝟬 ) = 𝟬
proof -
  have S1: ( - 𝟬 ) = ( 𝟬 - 𝟬 ) by (rule MMI˙df˙neg)
  have S2: 𝟬 ∈ ℂ by (rule MMI˙0cn)
  from S2 have S3: ( 𝟬 - 𝟬 ) = 𝟬 by (rule MMI˙subid)
  from S1 S3 show ( - 𝟬 ) = 𝟬 by (rule MMI˙eqtr)
qed

lemma (in MMIsar0) MMI˙renegcl: assumes A1: A ∈ ℝ
  shows ( (-  A) ) ∈ ℝ
proof -
  from A1 have S1: A ∈ ℝ.
  have S2: A ∈ ℝ ⟶ ( ∃ x ∈ ℝ . ( A + x ) = 𝟬 ) by (rule MMI˙axrnegex)
  from S1 S2 have S3: ∃ x ∈ ℝ . ( A + x ) = 𝟬 by (rule MMI˙ax˙mp)
  have S4: ( ∃ x ∈ ℝ . ( A + x ) = 𝟬 ) ⟷
    ( ∃ x . ( x ∈ ℝ ∧ ( A + x ) = 𝟬 ) ) by (rule MMI˙df˙rex)
  from S3 S4 have S5: ∃ x . ( x ∈ ℝ ∧ ( A + x ) = 𝟬 )
    by (rule MMI˙mpbi)
  { fix x
    have S6: x ∈ ℝ ⟶ x ∈ ℂ by (rule MMI˙recnt)
    have S7: 𝟬 ∈ ℂ by (rule MMI˙0cn)
    from A1 have S8: A ∈ ℝ.
```

from S8 have S9: A ∈ ℂ by (rule MMI˙recn)
have S10: (**0** ∈ ℂ ∧ A ∈ ℂ ∧ x ∈ ℂ) ⟶ ((**0** - A) = x ⟷
(A + x) = **0**) by (rule MMI˙subaddt)
from S7 S9 S10 have S11: x ∈ ℂ ⟶ ((**0** - A) = x ⟷
(A + x) = **0**) by (rule MMI˙mp3an12)
from S6 S11 have S12: x ∈ ℝ ⟶ ((**0** - A) = x ⟷
(A + x) = **0**) by (rule MMI˙syl)
have S13: ((- A)) = (**0** - A) by (rule MMI˙df˙neg)
from S13 have S14: ((- A)) = x ⟷ (**0** - A) = x
by (rule MMI˙eqeq1i)
from S12 S14 have S15: x ∈ ℝ ⟶ (((- A)) = x ⟷
(A + x) = **0**) by (rule MMI˙syl5bb)
have S16: x ∈ ℝ ⟶ (((- A)) = x ⟶ ((- A)) ∈ ℝ)
by (rule MMI˙eleq1a)
from S15 S16 have S17: x ∈ ℝ ⟶ ((A + x) = **0** ⟶
((- A)) ∈ ℝ) by (rule MMI˙sylbird)
from S17 have (x ∈ ℝ ∧ (A + x) = **0**) ⟶ ((- A)) ∈ ℝ
by (rule MMI˙imp)
} then have S18:
∀x . (x ∈ ℝ ∧ (A + x) = **0**) ⟶ ((- A)) ∈ ℝ
by auto
from S18 have S19: (∃ x . (x ∈ ℝ ∧ (A + x) = **0**)) ⟶
((- A)) ∈ ℝ by (rule MMI˙19˙23aiv)
from S5 S19 show ((- A)) ∈ ℝ by (rule MMI˙ax˙mp)
qed

lemma (in MMIsar0) MMI˙renegclt:
shows A ∈ ℝ ⟶ ((- A)) ∈ ℝ
proof -
have S1: A = if (A ∈ ℝ , A , **1**) ⟶ ((- A)) =
(- if (A ∈ ℝ , A , **1**)) by (rule MMI˙negeq)
from S1 have S2: A = if (A ∈ ℝ , A , **1**) ⟶ (((- A)) ∈ ℝ ⟷
(- if (A ∈ ℝ , A , **1**)) ∈ ℝ) by (rule MMI˙eleq1d)
have S3: **1** ∈ ℝ by (rule MMI˙ax1re)
from S3 have S4: if (A ∈ ℝ , A , **1**) ∈ ℝ by (rule MMI˙elimel)
from S4 have S5: (- if (A ∈ ℝ , A , **1**)) ∈ ℝ by (rule MMI˙renegcl)
from S2 S5 show A ∈ ℝ ⟶ ((- A)) ∈ ℝ by (rule MMI˙dedth)
qed

lemma (in MMIsar0) MMI˙resubclt:
shows (A ∈ ℝ ∧ B ∈ ℝ) ⟶ (A - B) ∈ ℝ
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + ((- B))) = (A - B)
by (rule MMI˙negsubt)
have S2: A ∈ ℝ ⟶ A ∈ ℂ by (rule MMI˙recnt)
have S3: B ∈ ℝ ⟶ B ∈ ℂ by (rule MMI˙recnt)
from S1 S2 S3 have S4: (A ∈ ℝ ∧ B ∈ ℝ) ⟶ (A + ((- B))) =
(A - B) by (rule MMI˙syl2an)
have S5: (A ∈ ℝ ∧ ((- B)) ∈ ℝ) ⟶ (A + ((- B))) ∈ ℝ

by (rule MMI˙axaddrcl)
have S6: B ∈ ℝ ⟶ ((- B)) ∈ ℝ by (rule MMI˙renegclt)
from S5 S6 have S7: (A ∈ ℝ ∧ B ∈ ℝ) ⟶ (A + ((- B))) ∈ ℝ
by (rule MMI˙sylan2)
from S4 S7 show (A ∈ ℝ ∧ B ∈ ℝ) ⟶ (A - B) ∈ ℝ
by (rule MMI˙eqeltrrd)
qed

lemma (in MMIsar0) MMI˙resubcl: assumes A1: A ∈ ℝ and
A2: B ∈ ℝ
shows (A - B) ∈ ℝ
proof -
from A1 have S1: A ∈ ℝ.
from A2 have S2: B ∈ ℝ.
have S3: (A ∈ ℝ ∧ B ∈ ℝ) ⟶ (A - B) ∈ ℝ by (rule MMI˙resubclt)
from S1 S2 S3 show (A - B) ∈ ℝ by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙0re:
shows **0** ∈ ℝ
proof -
have S1: **1** ∈ ℂ by (rule MMI˙1cn)
from S1 have S2: (**1** - **1**) = **0** by (rule MMI˙subid)
have S3: **1** ∈ ℝ by (rule MMI˙ax1re)
have S4: **1** ∈ ℝ by (rule MMI˙ax1re)
from S3 S4 have S5: (**1** - **1**) ∈ ℝ by (rule MMI˙resubcl)
from S2 S5 show **0** ∈ ℝ by (rule MMI˙eqeltrr)
qed

lemma (in MMIsar0) MMI˙mulid2t:
shows A ∈ ℂ ⟶ (**1** · A) = A
proof -
have S1: **1** ∈ ℂ by (rule MMI˙1cn)
have S2: (**1** ∈ ℂ ∧ A ∈ ℂ) ⟶ (**1** · A) = (A · **1**)
by (rule MMI˙axmulcom)
from S1 S2 have S3: A ∈ ℂ ⟶ (**1** · A) = (A · **1**) by (rule MMI˙mpan)
have S4: A ∈ ℂ ⟶ (A · **1**) = A by (rule MMI˙ax1id)
from S3 S4 show A ∈ ℂ ⟶ (**1** · A) = A by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙mul12t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · (B · C)) =
(B · (A · C))
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A · B) = (B · A)
by (rule MMI˙axmulcom)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ) ⟶

((A · B) · C) = ((B · A) · C) by (rule MMI˙opreq1d)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · B) · C) = ((B · A) · C) by (rule MMI˙3adant3)
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · B) · C) = (A · (B · C)) by (rule MMI˙axmulass)
have S5: (B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ) ⟶
((B · A) · C) = (B · (A · C)) by (rule MMI˙axmulass)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((B · A) · C) = (B · (A · C)) by (rule MMI˙3com12)
from S3 S4 S6 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A · (B · C)) = (B · (A · C)) by (rule MMI˙3eqtr3d)
qed

lemma (in MMIsar0) MMI˙mul23t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A · B) · C) =
((A · C) · B)
proof -
have S1: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B · C) = (C · B)
by (rule MMI˙axmulcom)
from S1 have S2: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · (B · C)) =
(A · (C · B)) by (rule MMI˙opreq2d)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · (B · C)) =
(A · (C · B)) by (rule MMI˙3adant1)
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A · B) · C) =
(A · (B · C)) by (rule MMI˙axmulass)
have S5: (A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A · C) · B) =
(A · (C · B)) by (rule MMI˙axmulass)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · C) · B) = (A · (C · B)) by (rule MMI˙3com23)
from S3 S4 S6 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · B) · C) = ((A · C) · B) by (rule MMI˙3eqtr4d)
qed

lemma (in MMIsar0) MMI˙mul4t:
shows ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · B) · (C · D)) = ((A · C) · (B · D))
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · B) · C) = ((A · C) · B) by (rule MMI˙mul23t)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(((A · B) · C) · D) = (((A · C) · B) · D)
by (rule MMI˙opreq1d)
from S2 have S3: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ C ∈ ℂ) ⟶
(((A · B) · C) · D) = (((A · C) · B) · D)
by (rule MMI˙3expa)
from S3 have S4: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A · B) · C) · D) = (((A · C) · B) · D)
by (rule MMI˙adantrr)
have S5: ((A · B) ∈ ℂ ∧ C ∈ ℂ ∧ D ∈ ℂ) ⟶

(((A · B) · C) · D) = ((A · B) · (C · D))
by (rule MMI˙axmulass)
from S5 have S6: ((A · B) ∈ ℂ ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A · B) · C) · D) = ((A · B) · (C · D)) by (rule MMI˙3expb)
have S7: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A · B) ∈ ℂ by (rule MMI˙axmulcl)
from S6 S7 have S8: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A · B) · C) · D) = ((A · B) · (C · D)) by (rule MMI˙sylan)
have S9: ((A · C) ∈ ℂ ∧ B ∈ ℂ ∧ D ∈ ℂ) ⟶
(((A · C) · B) · D) = ((A · C) · (B · D))
by (rule MMI˙axmulass)
from S9 have S10: ((A · C) ∈ ℂ ∧ (B ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A · C) · B) · D) = ((A · C) · (B · D))
by (rule MMI˙3expb)
have S11: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) ∈ ℂ by (rule MMI˙axmulcl)
from S10 S11 have S12: ((A ∈ ℂ ∧ C ∈ ℂ) ∧ (B ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A · C) · B) · D) = ((A · C) · (B · D))
by (rule MMI˙sylan)
from S12 have S13: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A · C) · B) · D) = ((A · C) · (B · D))
by (rule MMI˙an4s)
from S4 S8 S13 show ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · B) · (C · D)) = ((A · C) · (B · D))
by (rule MMI˙3eqtr3d)
qed

lemma (in MMIsar0) MMI˙muladdt:
shows ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) · (C + D)) =
(((A · C) + (D · B)) + ((A · D) + (C · B)))
proof -
have S1: ((A + B) ∈ ℂ ∧ C ∈ ℂ ∧ D ∈ ℂ) ⟶
((A + B) · (C + D)) =
(((A + B) · C) + ((A + B) · D))
by (rule MMI˙axdistr)
have S2: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + B) ∈ ℂ by (rule MMI˙axaddcl)
from S2 have S3: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(A + B) ∈ ℂ by (rule MMI˙adantr)
have S4: (C ∈ ℂ ∧ D ∈ ℂ) ⟶ C ∈ ℂ by (rule MMI˙pm3˙26)
from S4 have S5: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶ C ∈ ℂ
by (rule MMI˙adantl)
have S6: (C ∈ ℂ ∧ D ∈ ℂ) ⟶ D ∈ ℂ by (rule MMI˙pm3˙27)
from S6 have S7: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶ D ∈ ℂ
by (rule MMI˙adantl)
from S1 S3 S5 S7 have S8:
((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) · (C + D)) =
(((A + B) · C) + ((A + B) · D))
by (rule MMI˙syl3anc)
have S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶

$((A + B) \cdot C) = ((A \cdot C) + (B \cdot C))$
by (rule MMI˙adddirt)
from S9 have S10: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge C \in \mathbb{C}) \longrightarrow$
$((A + B) \cdot C) = ((A \cdot C) + (B \cdot C))$
by (rule MMI˙3expa)
from S10 have S11: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$((A + B) \cdot C) = ((A \cdot C) + (B \cdot C))$
by (rule MMI˙adantrr)
have S12: $(A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge D \in \mathbb{C}) \longrightarrow$
$((A + B) \cdot D) = ((A \cdot D) + (B \cdot D))$
by (rule MMI˙adddirt)
from S12 have S13: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge D \in \mathbb{C}) \longrightarrow$
$((A + B) \cdot D) = ((A \cdot D) + (B \cdot D))$
by (rule MMI˙3expa)
from S13 have S14: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$((A + B) \cdot D) = ((A \cdot D) + (B \cdot D))$
by (rule MMI˙adantrl)
from S11 S14 have S15: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(((A + B) \cdot C) + ((A + B) \cdot D)) =$
$(((A \cdot C) + (B \cdot C)) + ((A \cdot D) + (B \cdot D)))$
by (rule MMI˙opreq12d)
have S16:
$((A \cdot C) \in \mathbb{C} \wedge (B \cdot C) \in \mathbb{C} \wedge$
$((A \cdot D) + (B \cdot D)) \in \mathbb{C}) \longrightarrow$
$(((A \cdot C) + (B \cdot C)) + ((A \cdot D) + (B \cdot D))) =$
$(((A \cdot C) + ((A \cdot D) + (B \cdot D))) + (B \cdot C))$
by (rule MMI˙add23t)
have S17: $(A \in \mathbb{C} \wedge C \in \mathbb{C}) \longrightarrow (A \cdot C) \in \mathbb{C}$ by (rule MMI˙axmulcl)
from S17 have S18: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(A \cdot C) \in \mathbb{C}$ by (rule MMI˙ad2ant2r)
have S19: $(B \in \mathbb{C} \wedge C \in \mathbb{C}) \longrightarrow (B \cdot C) \in \mathbb{C}$ by (rule MMI˙axmulcl)
from S19 have S20: $(B \in \mathbb{C} \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(B \cdot C) \in \mathbb{C}$ by (rule MMI˙adantrr)
from S20 have S21: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(B \cdot C) \in \mathbb{C}$ by (rule MMI˙adantll)
have S22: $((A \cdot D) \in \mathbb{C} \wedge (B \cdot D) \in \mathbb{C}) \longrightarrow$
$((A \cdot D) + (B \cdot D)) \in \mathbb{C}$ by (rule MMI˙axaddcl)
have S23: $(A \in \mathbb{C} \wedge D \in \mathbb{C}) \longrightarrow (A \cdot D) \in \mathbb{C}$ by (rule MMI˙axmulcl)
have S24: $(B \in \mathbb{C} \wedge D \in \mathbb{C}) \longrightarrow (B \cdot D) \in \mathbb{C}$ by (rule MMI˙axmulcl)
from S22 S23 S24 have S25:
$((A \in \mathbb{C} \wedge D \in \mathbb{C}) \wedge (B \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$((A \cdot D) + (B \cdot D)) \in \mathbb{C}$ by (rule MMI˙syl2an)
from S25 have S26: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge D \in \mathbb{C}) \longrightarrow$
$((A \cdot D) + (B \cdot D)) \in \mathbb{C}$ by (rule MMI˙anandirs)
from S26 have S27: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$((A \cdot D) + (B \cdot D)) \in \mathbb{C}$ by (rule MMI˙adantrl)
from S16 S18 S21 S27 have S28:
$((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(((A \cdot C) + (B \cdot C)) + ((A \cdot D) + (B \cdot D))) =$

$(((A \cdot C) + ((A \cdot D) + (B \cdot D))) + (B \cdot C))$
by (rule MMI˙syl3anc)
have S29: $(B \in \mathbb{C} \wedge D \in \mathbb{C}) \longrightarrow (B \cdot D) = (D \cdot B)$
by (rule MMI˙axmulcom)
from S29 have S30: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(B \cdot D) = (D \cdot B)$ by (rule MMI˙ad2ant2l)
from S30 have S31: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(((A \cdot C) + (A \cdot D)) + (B \cdot D)) =$
$(((A \cdot C) + (A \cdot D)) + (D \cdot B))$
by (rule MMI˙opreq2d)
have S32: $((A \cdot C) \in \mathbb{C} \wedge (A \cdot D) \in \mathbb{C} \wedge (B \cdot D) \in \mathbb{C}) \longrightarrow$
$(((A \cdot C) + (A \cdot D)) + (B \cdot D)) =$
$((A \cdot C) + ((A \cdot D) + (B \cdot D)))$
by (rule MMI˙axaddass)
from S18 have S33:
$((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow (A \cdot C) \in \mathbb{C}$.
from S23 have S34: $(A \in \mathbb{C} \wedge D \in \mathbb{C}) \longrightarrow (A \cdot D) \in \mathbb{C}$.
from S34 have S35: $(A \in \mathbb{C} \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(A \cdot D) \in \mathbb{C}$ by (rule MMI˙adantrl)
from S35 have S36: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(A \cdot D) \in \mathbb{C}$ by (rule MMI˙adantlr)
from S24 have S37: $(B \in \mathbb{C} \wedge D \in \mathbb{C}) \longrightarrow (B \cdot D) \in \mathbb{C}$.
from S37 have S38: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(B \cdot D) \in \mathbb{C}$ by (rule MMI˙ad2ant2l)
from S32 S33 S36 S38 have S39:
$((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(((A \cdot C) + (A \cdot D)) + (B \cdot D)) =$
$((A \cdot C) + ((A \cdot D) + (B \cdot D)))$ by (rule MMI˙syl3anc)
have S40: $((A \cdot C) \in \mathbb{C} \wedge (A \cdot D) \in \mathbb{C} \wedge (D \cdot B) \in \mathbb{C}) \longrightarrow$
$(((A \cdot C) + (A \cdot D)) + (D \cdot B)) =$
$(((A \cdot C) + (D \cdot B)) + (A \cdot D))$ by (rule MMI˙add23t)
from S18 have S41:
$((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow (A \cdot C) \in \mathbb{C}$.
from S36 have S42: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(A \cdot D) \in \mathbb{C}$.
have S43: $(D \in \mathbb{C} \wedge B \in \mathbb{C}) \longrightarrow (D \cdot B) \in \mathbb{C}$ by (rule MMI˙axmulcl)
from S43 have S44: $(B \in \mathbb{C} \wedge D \in \mathbb{C}) \longrightarrow (D \cdot B) \in \mathbb{C}$
by (rule MMI˙ancoms)
from S44 have S45: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(D \cdot B) \in \mathbb{C}$ by (rule MMI˙ad2ant2l)
from S40 S41 S42 S45 have S46:
$((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(((A \cdot C) + (A \cdot D)) + (D \cdot B)) =$
$(((A \cdot C) + (D \cdot B)) + (A \cdot D))$ by (rule MMI˙syl3anc)
from S31 S39 S46 have S47:
$((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$((A \cdot C) + ((A \cdot D) + (B \cdot D))) =$
$(((A \cdot C) + (D \cdot B)) + (A \cdot D))$ by (rule MMI˙3eqtr3d)
have S48: $(B \in \mathbb{C} \wedge C \in \mathbb{C}) \longrightarrow (B \cdot C) = (C \cdot B)$

by (rule MMI˙axmulcom)
from S48 have S49: $((A \in \mathbb{C} \wedge D \in \mathbb{C}) \wedge (B \in \mathbb{C} \wedge C \in \mathbb{C})) \longrightarrow$
$(B \cdot C) = (C \cdot B)$ by (rule MMI˙adantl)
from S49 have S50: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(B \cdot C) = (C \cdot B)$ by (rule MMI˙an42s)
from S47 S50 have S51: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(((A \cdot C) + ((A \cdot D) + (B \cdot D))) + (B \cdot C)) =$
$((((A \cdot C) + (D \cdot B)) + (A \cdot D)) + (C \cdot B))$
by (rule MMI˙opreq12d)
have S52:
$(((A \cdot C) + (D \cdot B)) \in \mathbb{C} \wedge (A \cdot D) \in \mathbb{C} \wedge$
$(C \cdot B) \in \mathbb{C}) \longrightarrow$
$((((A \cdot C) + (D \cdot B)) + (A \cdot D)) + (C \cdot B)) =$
$(((A \cdot C) + (D \cdot B)) + ((A \cdot D) + (C \cdot B)))$
by (rule MMI˙axaddass)
have S53: $((A \cdot C) \in \mathbb{C} \wedge (D \cdot B) \in \mathbb{C}) \longrightarrow$
$((A \cdot C) + (D \cdot B)) \in \mathbb{C}$ by (rule MMI˙axaddcl)
from S17 have S54: $(A \in \mathbb{C} \wedge C \in \mathbb{C}) \longrightarrow (A \cdot C) \in \mathbb{C}$.
from S44 have S55: $(B \in \mathbb{C} \wedge D \in \mathbb{C}) \longrightarrow (D \cdot B) \in \mathbb{C}$.
from S53 S54 S55 have S56:
$((A \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge (B \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$((A \cdot C) + (D \cdot B)) \in \mathbb{C}$ by (rule MMI˙syl2an)
from S56 have S57: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$((A \cdot C) + (D \cdot B)) \in \mathbb{C}$ by (rule MMI˙an4s)
from S36 have S58: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(A \cdot D) \in \mathbb{C}$.
have S59: $(C \in \mathbb{C} \wedge B \in \mathbb{C}) \longrightarrow (C \cdot B) \in \mathbb{C}$ by (rule MMI˙axmulcl)
from S59 have S60: $(B \in \mathbb{C} \wedge C \in \mathbb{C}) \longrightarrow (C \cdot B) \in \mathbb{C}$
by (rule MMI˙ancoms)
from S60 have S61: $((A \in \mathbb{C} \wedge D \in \mathbb{C}) \wedge (B \in \mathbb{C} \wedge C \in \mathbb{C})) \longrightarrow$
$(C \cdot B) \in \mathbb{C}$ by (rule MMI˙adantl)
from S61 have S62: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(C \cdot B) \in \mathbb{C}$ by (rule MMI˙an42s)
from S52 S57 S58 S62 have S63:
$((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$((((A \cdot C) + (D \cdot B)) + (A \cdot D)) + (C \cdot B)) =$
$(((A \cdot C) + (D \cdot B)) + ((A \cdot D) + (C \cdot B)))$
by (rule MMI˙syl3anc)
from S28 S51 S63 have S64:
$((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$(((A \cdot C) + (B \cdot C)) + ((A \cdot D) + (B \cdot D))) =$
$(((A \cdot C) + (D \cdot B)) + ((A \cdot D) + (C \cdot B)))$
by (rule MMI˙3eqtrd)
from S8 S15 S64 show $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge (C \in \mathbb{C} \wedge D \in \mathbb{C})) \longrightarrow$
$((A + B) \cdot (C + D)) =$
$(((A \cdot C) + (D \cdot B)) + ((A \cdot D) + (C \cdot B)))$
by (rule MMI˙3eqtrd)
qed

lemma (in MMIsar0) MMI˙muladd11t:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((**1** + A) · (**1** + B)) =
((**1** + A) + (B + (A · B)))
proof -
have S1: **1** ∈ ℂ by (rule MMI˙1cn)
have S2: ((**1** + A) ∈ ℂ ∧ **1** ∈ ℂ ∧ B ∈ ℂ) ⟶
((**1** + A) · (**1** + B)) =
(((**1** + A) · **1**) + ((**1** + A) · B))
by (rule MMI˙axdistr)
from S1 S2 have S3: ((**1** + A) ∈ ℂ ∧ B ∈ ℂ) ⟶
((**1** + A) · (**1** + B)) =
(((**1** + A) · **1**) + ((**1** + A) · B))
by (rule MMI˙mp3an2)
have S4: **1** ∈ ℂ by (rule MMI˙1cn)
have S5: (**1** ∈ ℂ ∧ A ∈ ℂ) ⟶ (**1** + A) ∈ ℂ by (rule MMI˙axaddcl)
from S4 S5 have S6: A ∈ ℂ ⟶ (**1** + A) ∈ ℂ by (rule MMI˙mpan)
from S3 S6 have S7: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((**1** + A) · (**1** + B)) =
(((**1** + A) · **1**) + ((**1** + A) · B)) by (rule MMI˙sylan)
from S6 have S8: A ∈ ℂ ⟶ (**1** + A) ∈ ℂ .
have S9: (**1** + A) ∈ ℂ ⟶ ((**1** + A) · **1**) = (**1** + A)
by (rule MMI˙ax1id)
from S8 S9 have S10: A ∈ ℂ ⟶ ((**1** + A) · **1**) = (**1** + A)
by (rule MMI˙syl)
from S10 have S11: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((**1** + A) · **1**) = (**1** + A) by (rule MMI˙adantr)
have S12: **1** ∈ ℂ by (rule MMI˙1cn)
have S13: (**1** ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((**1** + A) · B) =
((**1** · B) + (A · B)) by (rule MMI˙adddirt)
from S12 S13 have S14: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((**1** + A) · B) = ((**1** · B)
+ (A · B)) by (rule MMI˙mp3an1)
have S15: B ∈ ℂ ⟶ (**1** · B) = B by (rule MMI˙mulid2t)
from S15 have S16: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (**1** · B) = B
by (rule MMI˙adantl)
from S16 have S17:
(A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((**1** · B) + (A · B)) =
(B + (A · B)) by (rule MMI˙opreq1d)
from S14 S17 have S18:
(A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((**1** + A) · B) =
(B + (A · B)) by (rule MMI˙eqtrd)
from S11 S18 have S19: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((**1** + A) · **1**) + ((**1** + A) · B)) =
((**1** + A) + (B + (A · B))) by (rule MMI˙opreq12d)
from S7 S19 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((**1** + A) · (**1** + B)) =
((**1** + A) + (B + (A · B)))
by (rule MMI˙eqtrd)

qed

lemma (in MMIsar0) MMI˙mul12: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows (A · (B · C)) = (B · (A · C))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from S1 S2 have S3: (A · B) = (B · A) by (rule MMI˙mulcom)
from S3 have S4: ((A · B) · C) = ((B · A) · C)
by (rule MMI˙opreq1i)
from A1 have S5: A ∈ ℂ.
from A2 have S6: B ∈ ℂ.
from A3 have S7: C ∈ ℂ.
from S5 S6 S7 have S8: ((A · B) · C) = (A · (B · C))
by (rule MMI˙mulass)
from A2 have S9: B ∈ ℂ.
from A1 have S10: A ∈ ℂ.
from A3 have S11: C ∈ ℂ.
from S9 S10 S11 have S12: ((B · A) · C) = (B · (A · C))
by (rule MMI˙mulass)
from S4 S8 S12 show (A · (B · C)) = (B · (A · C))
by (rule MMI˙3eqtr3)
qed

lemma (in MMIsar0) MMI˙mul23: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows ((A · B) · C) = ((A · C) · B)
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A · B) · C) =
((A · C) · B) by (rule MMI˙mul23t)
from S1 S2 S3 S4 show ((A · B) · C) = ((A · C) · B)
by (rule MMI˙mp3an)
qed

lemma (in MMIsar0) MMI˙mul4: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ and
A4: D ∈ ℂ
shows ((A · B) · (C · D)) = ((A · C) · (B · D))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from S1 S2 have S3: A ∈ ℂ ∧ B ∈ ℂ by (rule MMI˙pm3˙2i)

from A3 have S4: C ∈ ℂ.
from A4 have S5: D ∈ ℂ.
from S4 S5 have S6: C ∈ ℂ ∧ D ∈ ℂ by (rule MMI˙pm3˙2i)
have S7: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · B) · (C · D)) = ((A · C) · (B · D))
by (rule MMI˙mul4t)
from S3 S6 S7 show ((A · B) · (C · D)) = ((A · C) · (B · D))
by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙muladd: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ and
A4: D ∈ ℂ
shows ((A + B) · (C + D)) =
(((A · C) + (D · B)) + ((A · D) + (C · B)))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from S1 S2 have S3: A ∈ ℂ ∧ B ∈ ℂ by (rule MMI˙pm3˙2i)
from A3 have S4: C ∈ ℂ.
from A4 have S5: D ∈ ℂ.
from S4 S5 have S6: C ∈ ℂ ∧ D ∈ ℂ by (rule MMI˙pm3˙2i)
have S7: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) · (C + D)) =
(((A · C) + (D · B)) + ((A · D) + (C · B)))
by (rule MMI˙muladdt)
from S3 S6 S7 show
((A + B) · (C + D)) =
(((A · C) + (D · B)) + ((A · D) + (C · B)))
by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙subdit:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A · (B - C)) = ((A · B) - (A · C))
proof -
have S1: (A ∈ ℂ ∧ C ∈ ℂ ∧ (B - C) ∈ ℂ) ⟶
(A · (C + (B - C))) =
((A · C) + (A · (B - C))) by (rule MMI˙axdistr)
have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ A ∈ ℂ by (rule MMI˙3simp1)
have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ C ∈ ℂ by (rule MMI˙3simp3)
have S4: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B - C) ∈ ℂ by (rule MMI˙subclt)
from S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B - C) ∈ ℂ
by (rule MMI˙3adant1)
from S1 S2 S3 S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A · (C + (B - C))) =
((A · C) + (A · (B - C))) by (rule MMI˙syl3anc)

have S7: (C ∈ ℂ ∧ B ∈ ℂ) ⟶ (C + (B - C)) = B by (rule MMI˙pncan3t)
from S7 have S8: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (C + (B - C)) = B by (rule MMI˙ancoms)
from S8 have S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (C + (B - C)) = B by (rule MMI˙3adant1)
from S9 have S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A · (C + (B - C))) = (A · B) by (rule MMI˙opreq2d)
from S6 S10 have S11: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · C) + (A · (B - C))) = (A · B) by (rule MMI˙eqtr3d)
have S12: ((A · B) ∈ ℂ ∧ (A · C) ∈ ℂ ∧ (A · (B - C)) ∈ ℂ) ⟶
(((A · B) - (A · C)) = (A · (B - C)) ⟷
((A · C) + (A · (B - C))) = (A · B)) by (rule MMI˙subaddt)
have S13: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A · B) ∈ ℂ by (rule MMI˙axmulcl)
from S13 have S14: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · B) ∈ ℂ
by (rule MMI˙3adant3)
have S15: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) ∈ ℂ by (rule MMI˙axmulcl)
from S15 have S16: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) ∈ ℂ
by (rule MMI˙3adant2)
have S17: (A ∈ ℂ ∧ (B - C) ∈ ℂ) ⟶ (A · (B - C)) ∈ ℂ
by (rule MMI˙axmulcl)
from S4 have S18: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B - C) ∈ ℂ .
from S17 S18 have S19: (A ∈ ℂ ∧ (B ∈ ℂ ∧ C ∈ ℂ)) ⟶
(A · (B - C)) ∈ ℂ by (rule MMI˙sylan2)
from S19 have S20: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A · (B - C)) ∈ ℂ by (rule MMI˙3impb)
from S12 S14 S16 S20 have S21: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(((A · B) - (A · C)) = (A · (B - C)) ⟷
((A · C) + (A · (B - C))) = (A · B)) by (rule MMI˙syl3anc)
from S11 S21 have S22: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · B) - (A · C)) = (A · (B - C)) by (rule MMI˙mpbird)
from S22 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A · (B - C)) = ((A · B) - (A · C)) by (rule MMI˙eqcomd)
qed

lemma (in MMIsar0) MMI˙subdirt:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) · C) = ((A · C) - (B · C))
proof -
have S1: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶
(C · (A - B)) = ((C · A) - (C · B)) by (rule MMI˙subdit)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(C · (A - B)) = ((C · A) - (C · B)) by (rule MMI˙3coml)
have S3: ((A - B) ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) · C) = (C · (A - B)) by (rule MMI˙axmulcom)
have S4: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A - B) ∈ ℂ by (rule MMI˙subclt)
from S3 S4 have S5: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ C ∈ ℂ) ⟶
((A - B) · C) = (C · (A - B)) by (rule MMI˙sylan)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) · C) = (C · (A - B)) by (rule MMI˙3impa)

have S7: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) = (C · A) by (rule MMI˙axmulcom)
from S7 have S8: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) = (C · A)
by (rule MMI˙3adant2)
have S9: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B · C) = (C · B) by (rule MMI˙axmulcom)
from S9 have S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B · C) = (C · B)
by (rule MMI˙3adant1)
from S8 S10 have S11: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · C) - (B · C)) = ((C · A) - (C · B))
by (rule MMI˙opreq12d)
from S2 S6 S11 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) · C) = ((A · C) - (B · C)) by (rule MMI˙3eqtr4d)
qed

lemma (in MMIsar0) MMI˙subdi: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows (A · (B - C)) = ((A · B) - (A · C))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A · (B - C)) = ((A · B) - (A · C)) by (rule MMI˙subdit)
from S1 S2 S3 S4 show (A · (B - C)) = ((A · B) - (A · C))
by (rule MMI˙mp3an)
qed

lemma (in MMIsar0) MMI˙subdir: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows ((A - B) · C) = ((A · C) - (B · C))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) · C) = ((A · C) - (B · C)) by (rule MMI˙subdirt)
from S1 S2 S3 S4 show ((A - B) · C) = ((A · C) - (B · C))
by (rule MMI˙mp3an)
qed

lemma (in MMIsar0) MMI˙mul01: assumes A1: A ∈ ℂ
shows (A · **0**) = **0**
proof -
from A1 have S1: A ∈ ℂ.
have S2: **0** ∈ ℂ by (rule MMI˙0cn)
have S3: **0** ∈ ℂ by (rule MMI˙0cn)
from S1 S2 S3 have S4: (A · (**0** - **0**)) = ((A · **0**) - (A · **0**))
by (rule MMI˙subdi)

have S5: $\mathbf{0} \in \mathbb{C}$ by (rule MMI˙0cn)
from S5 have S6: ($\mathbf{0}$ - $\mathbf{0}$) = $\mathbf{0}$ by (rule MMI˙subid)
from S6 have S7: (A · ($\mathbf{0}$ - $\mathbf{0}$)) = (A · $\mathbf{0}$) by (rule MMI˙opreq2i)
from A1 have S8: A ∈ ℂ.
have S9: $\mathbf{0} \in \mathbb{C}$ by (rule MMI˙0cn)
from S8 S9 have S10: (A · $\mathbf{0}$) ∈ ℂ by (rule MMI˙mulcl)
from S10 have S11: ((A · $\mathbf{0}$) - (A · $\mathbf{0}$)) = $\mathbf{0}$ by (rule MMI˙subid)
from S4 S7 S11 show (A · $\mathbf{0}$) = $\mathbf{0}$ by (rule MMI˙3eqtr3)
qed

lemma (in MMIsar0) MMI˙mul02: assumes A1: A ∈ ℂ
shows ($\mathbf{0}$ · A) = $\mathbf{0}$
proof -
have S1: $\mathbf{0} \in \mathbb{C}$ by (rule MMI˙0cn)
from A1 have S2: A ∈ ℂ.
from S1 S2 have S3: ($\mathbf{0}$ · A) = (A · $\mathbf{0}$) by (rule MMI˙mulcom)
from A1 have S4: A ∈ ℂ.
from S4 have S5: (A · $\mathbf{0}$) = $\mathbf{0}$ by (rule MMI˙mul01)
from S3 S5 show ($\mathbf{0}$ · A) = $\mathbf{0}$ by (rule MMI˙eqtr)
qed

lemma (in MMIsar0) MMI˙1p1times: assumes A1: A ∈ ℂ
shows (($\mathbf{1}$ + $\mathbf{1}$) · A) = (A + A)
proof -
have S1: $\mathbf{1} \in \mathbb{C}$ by (rule MMI˙1cn)
have S2: $\mathbf{1} \in \mathbb{C}$ by (rule MMI˙1cn)
from A1 have S3: A ∈ ℂ.
from S1 S2 S3 have S4: (($\mathbf{1}$ + $\mathbf{1}$) · A) = (($\mathbf{1}$ · A) + ($\mathbf{1}$ · A))
by (rule MMI˙adddir)
from A1 have S5: A ∈ ℂ.
from S5 have S6: ($\mathbf{1}$ · A) = A by (rule MMI˙mulid2)
from S6 have S7: ($\mathbf{1}$ · A) = A .
from S6 S7 have S8: (($\mathbf{1}$ · A) + ($\mathbf{1}$ · A)) = (A + A)
by (rule MMI˙opreq12i)
from S4 S8 show (($\mathbf{1}$ + $\mathbf{1}$) · A) = (A + A)
by (rule MMI˙eqtr)
qed

lemma (in MMIsar0) MMI˙mul01t:
shows A ∈ ℂ ⟶ (A · $\mathbf{0}$) = $\mathbf{0}$
proof -
have S1: A = if (A ∈ ℂ , A , $\mathbf{0}$) ⟶
(A · $\mathbf{0}$) = (if (A ∈ ℂ , A , $\mathbf{0}$) · $\mathbf{0}$) by (rule MMI˙opreq1)
from S1 have S2: A = if (A ∈ ℂ , A , $\mathbf{0}$) ⟶
((A · $\mathbf{0}$) = $\mathbf{0}$ ⟷ (if (A ∈ ℂ , A , $\mathbf{0}$) · $\mathbf{0}$) = $\mathbf{0}$) by (rule MMI˙eqeq1d)
have S3: $\mathbf{0} \in \mathbb{C}$ by (rule MMI˙0cn)
from S3 have S4: if (A ∈ ℂ , A , $\mathbf{0}$) ∈ ℂ by (rule MMI˙elimel)
from S4 have S5: (if (A ∈ ℂ , A , $\mathbf{0}$) · $\mathbf{0}$) = $\mathbf{0}$ by (rule MMI˙mul01)
from S2 S5 show A ∈ ℂ ⟶ (A · $\mathbf{0}$) = $\mathbf{0}$ by (rule MMI˙dedth)

qed

lemma (in MMIsar0) MMI˙mul02t:
  shows A ∈ ℂ ⟶ (**0** · A) = **0**
proof -
  have S1: **0** ∈ ℂ by (rule MMI˙0cn)
  have S2: (**0** ∈ ℂ ∧ A ∈ ℂ) ⟶ (**0** · A) = (A · **0**) by (rule MMI˙axmulcom)
  from S1 S2 have S3: A ∈ ℂ ⟶ (**0** · A) = (A · **0**) by (rule MMI˙mpan)
  have S4: A ∈ ℂ ⟶ (A · **0**) = **0** by (rule MMI˙mul01t)
  from S3 S4 show A ∈ ℂ ⟶ (**0** · A) = **0** by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙mulneg1: assumes A1: A ∈ ℂ and
   A2: B ∈ ℂ
  shows (((- A)) · B) = (- (A · B))
proof -
  from A2 have S1: B ∈ ℂ.
  from S1 have S2: (B · **0**) = **0** by (rule MMI˙mul01)
  from A2 have S3: B ∈ ℂ.
  from A1 have S4: A ∈ ℂ.
  from S3 S4 have S5: (B · A) = (A · B) by (rule MMI˙mulcom)
  from S2 S5 have S6: ((B · **0**) - (B · A)) = (**0** - (A · B))
    by (rule MMI˙opreq12i)
  have S7: ((- A)) = (**0** - A) by (rule MMI˙df˙neg)
  from S7 have S8: (((- A)) · B) = ((**0** - A) · B)
    by (rule MMI˙opreq1i)
  have S9: **0** ∈ ℂ by (rule MMI˙0cn)
  from A1 have S10: A ∈ ℂ.
  from S9 S10 have S11: (**0** - A) ∈ ℂ by (rule MMI˙subcl)
  from A2 have S12: B ∈ ℂ.
  from S11 S12 have S13: ((**0** - A) · B) = (B · (**0** - A))
    by (rule MMI˙mulcom)
  from A2 have S14: B ∈ ℂ.
  have S15: **0** ∈ ℂ by (rule MMI˙0cn)
  from A1 have S16: A ∈ ℂ.
  from S14 S15 S16 have
    S17: (B · (**0** - A)) = ((B · **0**) - (B · A))
    by (rule MMI˙subdi)
  from S8 S13 S17 have
    S18: (((- A)) · B) = ((B · **0**) - (B · A)) by (rule MMI˙3eqtr)
  have S19: (- (A · B)) = (**0** - (A · B)) by (rule MMI˙df˙neg)
  from S6 S18 S19 show (((- A)) · B) = (- (A · B))
    by (rule MMI˙3eqtr4)
qed

lemma (in MMIsar0) MMI˙mulneg2: assumes A1: A ∈ ℂ and
   A2: B ∈ ℂ

shows (A · ((- B))) =
(- (A · B))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from S2 have S3: ((- B)) ∈ ℂ by (rule MMI˙negcl)
from S1 S3 have S4: (A · ((- B))) =
(((- B)) · A) by (rule MMI˙mulcom)
from A2 have S5: B ∈ ℂ.
from A1 have S6: A ∈ ℂ.
from S5 S6 have S7: (((- B)) · A) =
(- (B · A)) by (rule MMI˙mulneg1)
from A2 have S8: B ∈ ℂ.
from A1 have S9: A ∈ ℂ.
from S8 S9 have S10: (B · A) = (A · B) by (rule MMI˙mulcom)
from S10 have S11: (- (B · A)) =
(- (A · B)) by (rule MMI˙negeqi)
from S4 S7 S11 show (A · ((- B))) =
(- (A · B)) by (rule MMI˙3eqtr)
qed

lemma (in MMIsar0) MMI˙mul2neg: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows (((- A)) · ((- B))) =
(A · B)
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from S2 have S3: ((- B)) ∈ ℂ by (rule MMI˙negcl)
from S1 S3 have S4: (((- A)) · ((- B))) =
(- (A · ((- B)))) by (rule MMI˙mulneg1)
from A1 have S5: A ∈ ℂ.
from S3 have S6: ((- B)) ∈ ℂ .
from S5 S6 have S7: (A · ((- B))) =
(((- B)) · A) by (rule MMI˙mulcom)
from A2 have S8: B ∈ ℂ.
from A1 have S9: A ∈ ℂ.
from S8 S9 have S10: (((- B)) · A) =
(- (B · A)) by (rule MMI˙mulneg1)
from S7 S10 have S11: (A · ((- B))) =
(- (B · A)) by (rule MMI˙eqtr)
from S11 have S12: (- (A · ((- B)))) =
(- (- (B · A))) by (rule MMI˙negeqi)
from A2 have S13: B ∈ ℂ.
from A1 have S14: A ∈ ℂ.
from S13 S14 have S15: (B · A) ∈ ℂ by (rule MMI˙mulcl)
from S15 have S16: (- (- (B · A))) =
(B · A) by (rule MMI˙negneg)
from S4 S12 S16 have S17: (((- A)) · ((- B))) =

(B · A) by (rule MMI˙3eqtr)
from A2 have S18: B ∈ ℂ.
from A1 have S19: A ∈ ℂ.
from S18 S19 have S20: (B · A) = (A · B) by (rule MMI˙mulcom)
from S17 S20 show (((- A)) · ((- B))) =
(A · B) by (rule MMI˙eqtr)
qed

lemma (in MMIsar0) MMI˙negdi: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows (- (A + B)) =
(((- A)) + ((- B)))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from S1 S2 have S3: (A + B) ∈ ℂ by (rule MMI˙addcl)
from S3 have S4: (**1** · (A + B)) =
(A + B) by (rule MMI˙mulid2)
from S4 have S5: (- (**1** · (A + B))) =
(- (A + B)) by (rule MMI˙negeqi)
have S6: **1** ∈ ℂ by (rule MMI˙1cn)
from S6 have S7: (- **1**) ∈ ℂ by (rule MMI˙negcl)
from A1 have S8: A ∈ ℂ.
from A2 have S9: B ∈ ℂ.
from S7 S8 S9 have S10: ((- **1**) · (A + B)) =
(((- **1**) · A) + ((- **1**) · B)) by (rule MMI˙adddi)
have S11: **1** ∈ ℂ by (rule MMI˙1cn)
from S3 have S12: (A + B) ∈ ℂ .
from S11 S12 have S13: ((- **1**) · (A + B)) =
(- (**1** · (A + B))) by (rule MMI˙mulneg1)
have S14: **1** ∈ ℂ by (rule MMI˙1cn)
from A1 have S15: A ∈ ℂ.
from S14 S15 have S16: ((- **1**) · A) =
(- (**1** · A)) by (rule MMI˙mulneg1)
from A1 have S17: A ∈ ℂ.
from S17 have S18: (**1** · A) = A by (rule MMI˙mulid2)
from S18 have S19: (- (**1** · A)) = ((- A)) by (rule MMI˙negeqi)
from S16 S19 have S20: ((- **1**) · A) = ((- A)) by (rule MMI˙eqtr)
have S21: **1** ∈ ℂ by (rule MMI˙1cn)
from A2 have S22: B ∈ ℂ.
from S21 S22 have S23: ((- **1**) · B) =
(- (**1** · B)) by (rule MMI˙mulneg1)
from A2 have S24: B ∈ ℂ.
from S24 have S25: (**1** · B) = B by (rule MMI˙mulid2)
from S25 have S26: (- (**1** · B)) = ((- B)) by (rule MMI˙negeqi)
from S23 S26 have S27: ((- **1**) · B) = ((- B)) by (rule MMI˙eqtr)
from S20 S27 have S28: (((- **1**) · A) + ((- **1**) · B)) =
(((- A)) + ((- B))) by (rule MMI˙opreq12i)
from S10 S13 S28 have S29: (- (**1** · (A + B))) =

(((- A)) + ((- B))) by (rule MMI˙3eqtr3)
from S5 S29 show (- (A + B)) =
(((- A)) + ((- B))) by (rule MMI˙eqtr3)
qed

lemma (in MMIsar0) MMI˙negsubdi: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows (- (A - B)) =
(((- A)) + B)
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from S2 have S3: ((- B)) ∈ ℂ by (rule MMI˙negcl)
from S1 S3 have S4: (- (A + ((- B)))) =
(((- A)) + (- ((- B)))) by (rule MMI˙negdi)
from A1 have S5: A ∈ ℂ.
from A2 have S6: B ∈ ℂ.
from S5 S6 have S7: (A + ((- B))) = (A - B) by (rule MMI˙negsub)
from S7 have S8: (- (A + ((- B)))) =
(- (A - B)) by (rule MMI˙negeqi)
from A2 have S9: B ∈ ℂ.
from S9 have S10: (- ((- B))) = B by (rule MMI˙negneg)
from S10 have S11: (((- A)) + (- ((- B)))) =
(((- A)) + B) by (rule MMI˙opreq2i)
from S4 S8 S11 show (- (A - B)) =
(((- A)) + B) by (rule MMI˙3eqtr3)
qed

lemma (in MMIsar0) MMI˙negsubdi2: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows (- (A - B)) = (B - A)
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from S1 S2 have S3: (- (A - B)) =
(((- A)) + B) by (rule MMI˙negsubdi)
from A1 have S4: A ∈ ℂ.
from S4 have S5: ((- A)) ∈ ℂ by (rule MMI˙negcl)
from A2 have S6: B ∈ ℂ.
from S5 S6 have S7: (((- A)) + B) =
(B + ((- A))) by (rule MMI˙addcom)
from A2 have S8: B ∈ ℂ.
from A1 have S9: A ∈ ℂ.
from S8 S9 have S10: (B + ((- A))) = (B - A) by (rule MMI˙negsub)
from S3 S7 S10 show (- (A - B)) = (B - A) by (rule MMI˙3eqtr)
qed

lemma (in MMIsar0) MMI˙mulneg1t:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶

(((- A)) · B) =
(- (A · B))
proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
((- A)) =
(- if (A ∈ ℂ , A , **0**)) by (rule MMI˙negeq)
from S1 have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
(((- A)) · B) =
((- if (A ∈ ℂ , A , **0**)) · B) by (rule MMI˙opreq1d)
have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
(A · B) =
(if (A ∈ ℂ , A , **0**) · B) by (rule MMI˙opreq1)
from S3 have S4: A =
if (A ∈ ℂ , A , **0**) ⟶
(- (A · B)) =
(- (if (A ∈ ℂ , A , **0**) · B)) by (rule MMI˙negeqd)
from S2 S4 have S5: A =
if (A ∈ ℂ , A , **0**) ⟶
((((- A)) · B) =
(- (A · B)) ⟷
((- if (A ∈ ℂ , A , **0**)) · B) =
(- (if (A ∈ ℂ , A , **0**) · B))) by (rule MMI˙eqeq12d)
have S6: B =
if (B ∈ ℂ , B , **0**) ⟶
((- if (A ∈ ℂ , A , **0**)) · B) =
((- if (A ∈ ℂ , A , **0**)) · if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) · B) =
(if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
from S7 have S8: B =
if (B ∈ ℂ , B , **0**) ⟶
(- (if (A ∈ ℂ , A , **0**) · B)) =
(- (if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**))) by (rule MMI˙negeqd)
from S6 S8 have S9: B =
if (B ∈ ℂ , B , **0**) ⟶
(((- if (A ∈ ℂ , A , **0**)) · B) =
(- (if (A ∈ ℂ , A , **0**) · B)) ⟷
((- if (A ∈ ℂ , A , **0**)) · if (B ∈ ℂ , B , **0**)) =
(- (if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)))) by (rule MMI˙eqeq12d)
have S10: **0** ∈ ℂ by (rule MMI˙0cn)
from S10 have S11: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S12: **0** ∈ ℂ by (rule MMI˙0cn)
from S12 have S13: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
from S11 S13 have S14: ((- if (A ∈ ℂ , A , **0**)) · if (B ∈ ℂ , B , **0**)) =
(- (if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**))) by (rule MMI˙mulneg1)

from S5 S9 S14 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) · B) =
(- (A · B)) by (rule MMI˙dedth2h)
qed

lemma (in MMIsar0) MMI˙mulneg2t:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A · ((- B))) =
(- (A · B))
proof -
have S1: (B ∈ ℂ ∧ A ∈ ℂ) ⟶
(((- B)) · A) =
(- (B · A)) by (rule MMI˙mulneg1t)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- B)) · A) =
(- (B · A)) by (rule MMI˙ancoms)
have S3: (A ∈ ℂ ∧ ((- B)) ∈ ℂ) ⟶
(A · ((- B))) =
(((- B)) · A) by (rule MMI˙axmulcom)
have S4: B ∈ ℂ ⟶ ((- B)) ∈ ℂ by (rule MMI˙negclt)
from S3 S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A · ((- B))) =
(((- B)) · A) by (rule MMI˙sylan2)
have S6: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A · B) = (B · A) by (rule MMI˙axmulcom)
from S6 have S7: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A · B)) =
(- (B · A)) by (rule MMI˙negeqd)
from S2 S5 S7 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A · ((- B))) =
(- (A · B)) by (rule MMI˙3eqtr4d)
qed

lemma (in MMIsar0) MMI˙mulneg12t:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) · B) =
(A · ((- B)))
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) · B) =
(- (A · B)) by (rule MMI˙mulneg1t)
have S2: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A · ((- B))) =
(- (A · B)) by (rule MMI˙mulneg2t)
from S1 S2 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) · B) =
(A · ((- B))) by (rule MMI˙eqtr4d)
qed

lemma (in MMIsar0) MMI˙mul2negt:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) · ((- B))) =
(A · B)
proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
((- A)) =
(- if (A ∈ ℂ , A , **0**)) by (rule MMI˙negeq)
from S1 have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
(((- A)) · ((- B))) =
((- if (A ∈ ℂ , A , **0**)) · ((- B))) by (rule MMI˙opreq1d)
have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
(A · B) =
(if (A ∈ ℂ , A , **0**) · B) by (rule MMI˙opreq1)
from S2 S3 have S4: A =
if (A ∈ ℂ , A , **0**) ⟶
((((- A)) · ((- B))) =
(A · B) ⟷
((- if (A ∈ ℂ , A , **0**)) · ((- B))) =
(if (A ∈ ℂ , A , **0**) · B)) by (rule MMI˙eqeq12d)
have S5: B =
if (B ∈ ℂ , B , **0**) ⟶
((- B)) =
(- if (B ∈ ℂ , B , **0**)) by (rule MMI˙negeq)
from S5 have S6: B =
if (B ∈ ℂ , B , **0**) ⟶
((- if (A ∈ ℂ , A , **0**)) · ((- B))) =
((- if (A ∈ ℂ , A , **0**)) · (- if (B ∈ ℂ , B , **0**))) by (rule MMI˙opreq2d)
have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) · B) =
(if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
from S6 S7 have S8: B =
if (B ∈ ℂ , B , **0**) ⟶
(((- if (A ∈ ℂ , A , **0**)) · ((- B))) =
(if (A ∈ ℂ , A , **0**) · B) ⟷
((- if (A ∈ ℂ , A , **0**)) · (- if (B ∈ ℂ , B , **0**))) =
(if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**))) by (rule MMI˙eqeq12d)
have S9: **0** ∈ ℂ by (rule MMI˙0cn)
from S9 have S10: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S11: **0** ∈ ℂ by (rule MMI˙0cn)
from S11 have S12: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
from S10 S12 have S13: ((- if (A ∈ ℂ , A , **0**)) · (- if (B ∈ ℂ , B , **0**))) =
(if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) by (rule MMI˙mul2neg)
from S4 S8 S13 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) · ((- B))) =

(A · B) by (rule MMI˙dedth2h)
qed

lemma (in MMIsar0) MMI˙negdit:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A + B)) =
(((- A)) + ((- B)))
proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
(A + B) =
(if (A ∈ ℂ , A , **0**) + B) by (rule MMI˙opreq1)
from S1 have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
(- (A + B)) =
(- (if (A ∈ ℂ , A , **0**) + B)) by (rule MMI˙negeqd)
have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
((- A)) =
(- if (A ∈ ℂ , A , **0**)) by (rule MMI˙negeq)
from S3 have S4: A =
if (A ∈ ℂ , A , **0**) ⟶
(((- A)) + ((- B))) =
((- if (A ∈ ℂ , A , **0**)) + ((- B))) by (rule MMI˙opreq1d)
from S2 S4 have S5: A =
if (A ∈ ℂ , A , **0**) ⟶
((- (A + B)) =
(((- A)) + ((- B))) ⟷
(- (if (A ∈ ℂ , A , **0**) + B)) =
((- if (A ∈ ℂ , A , **0**)) + ((- B)))) by (rule MMI˙eqeq12d)
have S6: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) + B) =
(if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
from S6 have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(- (if (A ∈ ℂ , A , **0**) + B)) =
(- (if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**))) by (rule MMI˙negeqd)
have S8: B =
if (B ∈ ℂ , B , **0**) ⟶
((- B)) =
(- if (B ∈ ℂ , B , **0**)) by (rule MMI˙negeq)
from S8 have S9: B =
if (B ∈ ℂ , B , **0**) ⟶
((- if (A ∈ ℂ , A , **0**)) + ((- B))) =
((- if (A ∈ ℂ , A , **0**)) + (- if (B ∈ ℂ , B , **0**))) by (rule MMI˙opreq2d)
from S7 S9 have S10: B =
if (B ∈ ℂ , B , **0**) ⟶
((- (if (A ∈ ℂ , A , **0**) + B)) =

((- if (A ∈ ℂ , A , **0**)) + ((- B))) ⟷
(- (if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**))) =
((- if (A ∈ ℂ , A , **0**)) + (- if (B ∈ ℂ , B , **0**)))) by (rule MMI˙eqeq12d)
have S11: **0** ∈ ℂ by (rule MMI˙0cn)
from S11 have S12: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S13: **0** ∈ ℂ by (rule MMI˙0cn)
from S13 have S14: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
from S12 S14 have S15: (- (if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**))) =
((- if (A ∈ ℂ , A , **0**)) + (- if (B ∈ ℂ , B , **0**))) by (rule MMI˙negdi)
from S5 S10 S15 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A + B)) =
(((- A)) + ((- B))) by (rule MMI˙dedth2h)
qed

lemma (in MMIsar0) MMI˙negdi2t:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A + B)) = (((- A)) - B)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A + B)) =
(((- A)) + ((- B))) by (rule MMI˙negdit)
have S2: (((- A)) ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) + ((- B))) =
(((- A)) - B) by (rule MMI˙negsubt)
have S3: A ∈ ℂ ⟶ ((- A)) ∈ ℂ by (rule MMI˙negclt)
from S2 S3 have S4: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) + ((- B))) =
(((- A)) - B) by (rule MMI˙sylan)
from S1 S4 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A + B)) = (((- A)) - B)
by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙negsubdit:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A - B)) = (((- A)) + B)
proof -
have S1: (A ∈ ℂ ∧ ((- B)) ∈ ℂ) ⟶
(- (A + ((- B)))) =
(((- A)) + (- ((- B)))) by (rule MMI˙negdit)
have S2: B ∈ ℂ ⟶ ((- B)) ∈ ℂ by (rule MMI˙negclt)
from S1 S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A + ((- B)))) =
(((- A)) + (- ((- B)))) by (rule MMI˙sylan2)
have S4: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A + ((- B))) = (A - B) by (rule MMI˙negsubt)
from S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ) ⟶

(- (A + ((- B)))) =
(- (A - B)) by (rule MMI˙negeqd)
have S6: B ∈ ℂ ⟶ (- ((- B))) = B by (rule MMI˙negnegt)
from S6 have S7: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (- ((- B))) = B
by (rule MMI˙adantl)
from S7 have S8: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) + (- ((- B)))) =
(((- A)) + B) by (rule MMI˙opreq2d)
from S3 S5 S8 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A - B)) = (((- A)) + B)
by (rule MMI˙3eqtr3d)
qed

lemma (in MMIsar0) MMI˙negsubdi2t:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A - B)) = (B - A)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A - B)) = (((- A)) + B) by (rule MMI˙negsubdit)
have S2: (((- A)) ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) + B) = (B + ((- A))) by (rule MMI˙axaddcom)
have S3: A ∈ ℂ ⟶ ((- A)) ∈ ℂ by (rule MMI˙negclt)
from S2 S3 have S4: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) + B) = (B + ((- A))) by (rule MMI˙sylan)
have S5: (B ∈ ℂ ∧ A ∈ ℂ) ⟶
(B + ((- A))) = (B - A) by (rule MMI˙negsubt)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(B + ((- A))) = (B - A) by (rule MMI˙ancoms)
from S1 S4 S6 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A - B)) = (B - A)
by (rule MMI˙3eqtrd)
qed

lemma (in MMIsar0) MMI˙subsub2t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - (B - C)) = (A + (C - B))
proof -
have S1: (A ∈ ℂ ∧ (B - C) ∈ ℂ) ⟶
(A + (- (B - C))) =
(A - (B - C)) by (rule MMI˙negsubt)
have S2: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B - C) ∈ ℂ by (rule MMI˙subclt)
from S1 S2 have S3: (A ∈ ℂ ∧ (B ∈ ℂ ∧ C ∈ ℂ)) ⟶
(A + (- (B - C))) =
(A - (B - C)) by (rule MMI˙sylan2)
from S3 have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (- (B - C))) =
(A - (B - C)) by (rule MMI˙3impb)
have S5: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
(- (B - C)) = (C - B) by (rule MMI˙negsubdi2t)

from S5 have S6: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (- (B - C))) =
(A + (C - B)) by (rule MMI˙opreq2d)
from S6 have S7: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (- (B - C))) =
(A + (C - B)) by (rule MMI˙3adant1)
from S4 S7 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - (B - C)) = (A + (C - B))
by (rule MMI˙eqtr3d)
qed

lemma (in MMIsar0) MMI˙subsubt:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - (B - C)) = ((A - B) + C)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - (B - C)) = (A + (C - B)) by (rule MMI˙subsub2t)
have S2: (A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ) ⟶
((A + C) - B) = (A + (C - B)) by (rule MMI˙addsubasst)
have S3: (A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ) ⟶
((A + C) - B) = ((A - B) + C) by (rule MMI˙addsubt)
from S2 S3 have S4: (A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ) ⟶
(A + (C - B)) = ((A - B) + C) by (rule MMI˙eqtr3d)
from S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (C - B)) = ((A - B) + C) by (rule MMI˙3com23)
from S1 S5 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - (B - C)) = ((A - B) + C)
by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙subsub3t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - (B - C)) = ((A + C) - B)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - (B - C)) = (A + (C - B)) by (rule MMI˙subsub2t)
have S2: (A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ) ⟶
((A + C) - B) = (A + (C - B)) by (rule MMI˙addsubasst)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + C) - B) = (A + (C - B)) by (rule MMI˙3com23)
from S1 S3 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - (B - C)) = ((A + C) - B)
by (rule MMI˙eqtr4d)
qed

lemma (in MMIsar0) MMI˙subsub4t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - C) = (A - (B + C))
proof -

have S1: (A ∈ ℂ ∧ B ∈ ℂ ∧ (- C) ∈ ℂ) ⟶
(A - (B - (- C))) =
((A - B) + (- C)) by (rule MMI˙subsubt)
have S2: C ∈ ℂ ⟶ (- C) ∈ ℂ by (rule MMI˙negclt)
from S1 S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - (B - (- C))) =
((A - B) + (- C)) by (rule MMI˙syl3an3)
have S4: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B - (- C)) = (B + C) by (rule MMI˙subnegt)
from S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B - (- C)) = (B + C) by (rule MMI˙3adant1)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - (B - (- C))) =
(A - (B + C)) by (rule MMI˙opreq2d)
have S7: ((A - B) ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) + (- C)) =
((A - B) - C) by (rule MMI˙negsubt)
have S8: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A - B) ∈ ℂ by (rule MMI˙subclt)
from S7 S8 have S9: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ C ∈ ℂ) ⟶
((A - B) + (- C)) =
((A - B) - C) by (rule MMI˙sylan)
from S9 have S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) + (- C)) =
((A - B) - C) by (rule MMI˙3impa)
from S3 S6 S10 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - C) = (A - (B + C))
by (rule MMI˙3eqtr3rd)
qed

lemma (in MMIsar0) MMI˙sub23t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - C) = ((A - C) - B)
proof -
have S1: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B + C) = (C + B) by (rule MMI˙axaddcom)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B + C) = (C + B) by (rule MMI˙3adant1)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - (B + C)) = (A - (C + B)) by (rule MMI˙opreq2d)
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - C) = (A - (B + C)) by (rule MMI˙subsub4t)
have S5: (A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ) ⟶
((A - C) - B) = (A - (C + B)) by (rule MMI˙subsub4t)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - C) - B) = (A - (C + B)) by (rule MMI˙3com23)
from S3 S4 S6 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - C) = ((A - C) - B)
by (rule MMI˙3eqtr4d)
qed

lemma (in MMIsar0) MMI˙nnncant:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) = (A - B)
proof -
have S1: (A ∈ ℂ ∧ (B - C) ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) =
(A - ((B - C) + C)) by (rule MMI˙subsub4t)
have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ A ∈ ℂ by (rule MMI˙3simp1)
have S3: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B - C) ∈ ℂ by (rule MMI˙subclt)
from S3 have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B - C) ∈ ℂ by (rule MMI˙3adant1)
have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ C ∈ ℂ by (rule MMI˙3simp3)
from S1 S2 S4 S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) =
(A - ((B - C) + C)) by (rule MMI˙syl3anc)
have S7: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
((B - C) + C) = B by (rule MMI˙npcant)
from S7 have S8: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - ((B - C) + C)) = (A - B) by (rule MMI˙opreq2d)
from S8 have S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - ((B - C) + C)) = (A - B) by (rule MMI˙3adant1)
from S6 S9 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) = (A - B)
by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙nnncan1t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - (A - C)) = (C - B)
proof -
have S1: ((A - B) ∈ ℂ ∧ (A - C) ∈ ℂ) ⟶
((A - B) + (- (A - C))) =
((A - B) - (A - C)) by (rule MMI˙negsubt)
have S2: ((A - B) ∈ ℂ ∧ (- (A - C)) ∈ ℂ) ⟶
((A - B) + (- (A - C))) =
((- (A - C)) + (A - B)) by (rule MMI˙axaddcom)
have S3: (A - C) ∈ ℂ ⟶ (- (A - C)) ∈ ℂ
by (rule MMI˙negclt)
from S2 S3 have S4: ((A - B) ∈ ℂ ∧ (A - C) ∈ ℂ) ⟶
((A - B) + (- (A - C))) =
((- (A - C)) + (A - B)) by (rule MMI˙sylan2)
from S1 S4 have S5: ((A - B) ∈ ℂ ∧ (A - C) ∈ ℂ) ⟶
((A - B) - (A - C)) =
((- (A - C)) + (A - B)) by (rule MMI˙eqtr3d)
have S6: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A - B) ∈ ℂ by (rule MMI˙subclt)
from S6 have S7: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - B) ∈ ℂ by (rule MMI˙3adant3)
have S8: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A - C) ∈ ℂ by (rule MMI˙subclt)

from S8 have S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - C) ∈ ℂ by (rule MMI˙3adant2)
from S5 S7 S9 have S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - (A - C)) =
((- (A - C)) + (A - B)) by (rule MMI˙sylanc)
have S11: (A ∈ ℂ ∧ C ∈ ℂ) ⟶
(- (A - C)) = (C - A) by (rule MMI˙negsubdi2t)
from S11 have S12: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(- (A - C)) = (C - A) by (rule MMI˙3adant2)
from S12 have S13: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((- (A - C)) + (A - B)) =
((C - A) + (A - B)) by (rule MMI˙opreq1d)
have S14: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶
((C - A) + (A - B)) = (C - B) by (rule MMI˙npncant)
from S14 have S15: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((C - A) + (A - B)) = (C - B) by (rule MMI˙3coml)
from S10 S13 S15 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - (A - C)) = (C - B)
by (rule MMI˙3eqtrd)
qed

lemma (in MMIsar0) MMI˙nnncan2t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - C) - (B - C)) = (A - B)
proof -
have S1: (A ∈ ℂ ∧ (B - C) ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) =
((A - C) - (B - C)) by (rule MMI˙sub23t)
have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ A ∈ ℂ by (rule MMI˙3simp1)
have S3: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B - C) ∈ ℂ by (rule MMI˙subclt)
from S3 have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B - C) ∈ ℂ by (rule MMI˙3adant1)
have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ C ∈ ℂ by (rule MMI˙3simp3)
from S1 S2 S4 S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) =
((A - C) - (B - C)) by (rule MMI˙syl3anc)
have S7: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) = (A - B) by (rule MMI˙nnncant)
from S6 S7 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - C) - (B - C)) = (A - B) by (rule MMI˙eqtr3d)
qed

lemma (in MMIsar0) MMI˙nncant:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A - (A - B)) = B
proof -
have S1: **0** ∈ ℂ by (rule MMI˙0cn)

have S2: (A ∈ ℂ ∧ **0** ∈ ℂ ∧ B ∈ ℂ) ⟶
((A - **0**) - (A - B)) = (B - **0**) by (rule MMI˙nnncan1t)
from S1 S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A - **0**) - (A - B)) = (B - **0**) by (rule MMI˙mp3an2)
have S4: A ∈ ℂ ⟶ (A - **0**) = A by (rule MMI˙subid1t)
from S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A - **0**) = A
by (rule MMI˙adantr)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A - **0**) - (A - B)) =
(A - (A - B)) by (rule MMI˙opreq1d)
have S7: B ∈ ℂ ⟶ (B - **0**) = B by (rule MMI˙subid1t)
from S7 have S8: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (B - **0**) = B
by (rule MMI˙adantl)
from S3 S6 S8 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A - (A - B)) = B by (rule MMI˙3eqtr3d)
qed

lemma (in MMIsar0) MMI˙nppcan2t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B + C)) + C) = (A - B)
proof -
have S1: (A ∈ ℂ ∧ (B + C) ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - ((B + C) - C)) =
((A - (B + C)) + C) by (rule MMI˙subsubt)
have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ A ∈ ℂ by (rule MMI˙3simp1)
have S3: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B + C) ∈ ℂ by (rule MMI˙axaddcl)
from S3 have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B + C) ∈ ℂ by (rule MMI˙3adant1)
have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ C ∈ ℂ by (rule MMI˙3simp3)
from S1 S2 S4 S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - ((B + C) - C)) =
((A - (B + C)) + C) by (rule MMI˙syl3anc)
have S7: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
((B + C) - C) = B by (rule MMI˙pncant)
from S7 have S8: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((B + C) - C) = B by (rule MMI˙3adant1)
from S8 have S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - ((B + C) - C)) = (A - B) by (rule MMI˙opreq2d)
from S6 S9 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B + C)) + C) = (A - B) by (rule MMI˙eqtr3d)
qed

lemma (in MMIsar0) MMI˙mulm1t:
shows A ∈ ℂ ⟶ ((- **1**) · A) = ((- A))
proof -
have S1: **1** ∈ ℂ by (rule MMI˙1cn)
have S2: (**1** ∈ ℂ ∧ A ∈ ℂ) ⟶
((- **1**) · A) = (- (**1** · A)) by (rule MMI˙mulneg1t)
from S1 S2 have S3: A ∈ ℂ ⟶

((- **1**) · A) = (- (**1** · A)) by (rule MMI˙mpan)
have S4: A ∈ ℂ ⟶ (**1** · A) = A by (rule MMI˙mulid2t)
from S4 have S5: A ∈ ℂ ⟶ (- (**1** · A)) = ((- A))
by (rule MMI˙negeqd)
from S3 S5 show A ∈ ℂ ⟶ ((- **1**) · A) = ((- A))
by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙mulm1: assumes A1: A ∈ ℂ
shows ((- **1**) · A) = ((- A))
proof -
from A1 have S1: A ∈ ℂ.
have S2: A ∈ ℂ ⟶ ((- **1**) · A) = ((- A)) by (rule MMI˙mulm1t)
from S1 S2 show ((- **1**) · A) = ((- A)) by (rule MMI˙ax˙mp)
qed

lemma (in MMIsar0) MMI˙sub4t:
shows ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) - (C + D)) =
((A - C) + (B - D))
proof -
have S1: (C ∈ ℂ ∧ D ∈ ℂ) ⟶
(- (C + D)) =
((- C) + (- D)) by (rule MMI˙negdit)
from S1 have S2: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(- (C + D)) =
((- C) + (- D)) by (rule MMI˙adantl)
from S2 have S3: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) + (- (C + D))) =
((A + B) + ((- C) + (- D)))
by (rule MMI˙opreq2d)
have S4:
((A ∈ ℂ ∧ B ∈ ℂ) ∧ ((- C) ∈ ℂ ∧ (- D) ∈ ℂ)) ⟶
((A + B) + ((- C) + (- D))) =
((A + (- C)) + (B + (- D))) by (rule MMI˙add4t)
have S5: C ∈ ℂ ⟶ (- C) ∈ ℂ by (rule MMI˙negclt)
have S6: D ∈ ℂ ⟶ (- D) ∈ ℂ by (rule MMI˙negclt)
from S5 S6 have S7: (C ∈ ℂ ∧ D ∈ ℂ) ⟶
((- C) ∈ ℂ ∧ (- D) ∈ ℂ) by (rule MMI˙anim12i)
from S4 S7 have S8: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) + ((- C) + (- D))) =
((A + (- C)) + (B + (- D))) by (rule MMI˙sylan2)
from S3 S8 have S9: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) + (- (C + D))) =
((A + (- C)) + (B + (- D))) by (rule MMI˙eqtrd)
have S10: ((A + B) ∈ ℂ ∧ (C + D) ∈ ℂ) ⟶
((A + B) + (- (C + D))) =
((A + B) - (C + D)) by (rule MMI˙negsubt)
have S11: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + B) ∈ ℂ by (rule MMI˙axaddcl)

have S12: (C ∈ ℂ ∧ D ∈ ℂ) ⟶ (C + D) ∈ ℂ by (rule MMI˙axaddcl)
from S10 S11 S12 have S13:
((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) + (- (C + D))) =
((A + B) - (C + D)) by (rule MMI˙syl2an)
have S14: (A ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (- C)) = (A - C) by (rule MMI˙negsubt)
from S14 have S15: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(A + (- C)) = (A - C) by (rule MMI˙ad2ant2r)
have S16: (B ∈ ℂ ∧ D ∈ ℂ) ⟶
(B + (- D)) = (B - D) by (rule MMI˙negsubt)
from S16 have S17: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(B + (- D)) = (B - D) by (rule MMI˙ad2ant2l)
from S15 S17 have S18: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + (- C)) + (B + (- D))) =
((A - C) + (B - D)) by (rule MMI˙opreq12d)
from S9 S13 S18 show ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) - (C + D)) =
((A - C) + (B - D)) by (rule MMI˙3eqtr3d)
qed

lemma (in MMIsar0) MMI˙sub4: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ and
A4: D ∈ ℂ
shows ((A + B) - (C + D)) =
((A - C) + (B - D))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from S1 S2 have S3: A ∈ ℂ ∧ B ∈ ℂ by (rule MMI˙pm3˙2i)
from A3 have S4: C ∈ ℂ.
from A4 have S5: D ∈ ℂ.
from S4 S5 have S6: C ∈ ℂ ∧ D ∈ ℂ by (rule MMI˙pm3˙2i)
have S7: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) - (C + D)) =
((A - C) + (B - D)) by (rule MMI˙sub4t)
from S3 S6 S7 show ((A + B) - (C + D)) =
((A - C) + (B - D)) by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙mulsubt:
shows ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A - B) · (C - D)) =
(((A · C) + (D · B)) - ((A · D) + (C · B)))
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A + ((- B))) = (A - B) by (rule MMI˙negsubt)
have S2: (C ∈ ℂ ∧ D ∈ ℂ) ⟶

(C + (- D)) = (C - D) by (rule MMI˙negsubt)
from S1 S2 have S3: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + ((- B))) · (C + (- D))) =
((A - B) · (C - D)) by (rule MMI˙opreqan12d)
have S4: ((A ∈ ℂ ∧ ((- B)) ∈ ℂ) ∧ (C ∈ ℂ ∧ (- D) ∈ ℂ)) ⟶
((A + ((- B))) · (C + (- D))) =
(((A · C) + ((- D) · ((- B)))) + ((A · (- D)) + (C · ((- B))))) by (rule MMI˙muladdt)
have S5: D ∈ ℂ ⟶ (- D) ∈ ℂ by (rule MMI˙negclt)
from S4 S5 have S6: ((A ∈ ℂ ∧ ((- B)) ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + ((- B))) · (C + (- D))) =
(((A · C) + ((- D) · ((- B)))) +
((A · (- D)) + (C · ((- B))))) by (rule MMI˙sylanr2)
have S7: B ∈ ℂ ⟶ ((- B)) ∈ ℂ by (rule MMI˙negclt)
from S6 S7 have S8: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + ((- B))) · (C + (- D))) =
(((A · C) + ((- D) · ((- B))))
+ ((A · (- D)) + (C · ((- B)))))
by (rule MMI˙sylanl2)
have S9: (D ∈ ℂ ∧ B ∈ ℂ) ⟶
((- D) · ((- B))) = (D · B) by (rule MMI˙mul2negt)
from S9 have S10: (B ∈ ℂ ∧ D ∈ ℂ) ⟶
((- D) · ((- B))) = (D · B) by (rule MMI˙ancoms)
from S10 have S11: (B ∈ ℂ ∧ D ∈ ℂ) ⟶
((A · C) + ((- D) · ((- B)))) =
((A · C) + (D · B)) by (rule MMI˙opreq2d)
from S11 have S12: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · C) + ((- D) · ((- B)))) =
((A · C) + (D · B)) by (rule MMI˙ad2ant2l)
have S13: (A ∈ ℂ ∧ D ∈ ℂ) ⟶
(A · (- D)) = (- (A · D)) by (rule MMI˙mulneg2t)
have S14: (C ∈ ℂ ∧ B ∈ ℂ) ⟶
(C · ((- B))) = (- (C · B)) by (rule MMI˙mulneg2t)
from S13 S14 have S15: ((A ∈ ℂ ∧ D ∈ ℂ) ∧ (C ∈ ℂ ∧ B ∈ ℂ)) ⟶
((A · (- D)) + (C · ((- B)))) =
((- (A · D)) + (- (C · B))) by (rule MMI˙opreqan12d)
have S16: ((A · D) ∈ ℂ ∧ (C · B) ∈ ℂ) ⟶
(- ((A · D) + (C · B))) =
((- (A · D)) + (- (C · B))) by (rule MMI˙negdit)
have S17: (A ∈ ℂ ∧ D ∈ ℂ) ⟶ (A · D) ∈ ℂ by (rule MMI˙axmulcl)
have S18: (C ∈ ℂ ∧ B ∈ ℂ) ⟶ (C · B) ∈ ℂ by (rule MMI˙axmulcl)
from S16 S17 S18 have S19:
((A ∈ ℂ ∧ D ∈ ℂ) ∧ (C ∈ ℂ ∧ B ∈ ℂ)) ⟶
(- ((A · D) + (C · B))) =
((- (A · D)) + (- (C · B))) by (rule MMI˙syl2an)
from S15 S19 have S20: ((A ∈ ℂ ∧ D ∈ ℂ) ∧ (C ∈ ℂ ∧ B ∈ ℂ)) ⟶
((A · (- D)) + (C · ((- B)))) =
(- ((A · D) + (C · B))) by (rule MMI˙eqtr4d)
from S20 have S21: ((A ∈ ℂ ∧ D ∈ ℂ) ∧ (B ∈ ℂ ∧ C ∈ ℂ)) ⟶

((A · (- D)) + (C · ((- B)))) =
(- ((A · D) + (C · B))) by (rule MMI˙ancom2s)
from S21 have S22: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · (- D)) + (C · ((- B)))) =
(- ((A · D) + (C · B))) by (rule MMI˙an42s)
from S12 S22 have S23: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A · C) + ((- D) · ((- B)))) +
((A · (- D)) + (C · ((- B))))) =
(((A · C) + (D · B)) + (- ((A · D) +
(C · B)))) by (rule MMI˙opreq12d)
have S24: (((A · C) + (D · B)) ∈ ℂ ∧ ((A · D) +
(C · B)) ∈ ℂ) ⟶
(((A · C) + (D · B)) + (- ((A · D) + (C · B)))) =
(((A · C) + (D · B)) - ((A · D) + (C · B)))
by (rule MMI˙negsubt)
have S25: ((A · C) ∈ ℂ ∧ (D · B) ∈ ℂ) ⟶
((A · C) + (D · B)) ∈ ℂ by (rule MMI˙axaddcl)
have S26: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) ∈ ℂ by (rule MMI˙axmulcl)
have S27: (D ∈ ℂ ∧ B ∈ ℂ) ⟶ (D · B) ∈ ℂ by (rule MMI˙axmulcl)
from S27 have S28: (B ∈ ℂ ∧ D ∈ ℂ) ⟶ (D · B) ∈ ℂ
by (rule MMI˙ancoms)
from S25 S26 S28 have S29:
((A ∈ ℂ ∧ C ∈ ℂ) ∧ (B ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · C) + (D · B)) ∈ ℂ by (rule MMI˙syl2an)
from S29 have S30: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · C) + (D · B)) ∈ ℂ by (rule MMI˙an4s)
have S31: ((A · D) ∈ ℂ ∧ (C · B) ∈ ℂ) ⟶
((A · D) + (C · B)) ∈ ℂ by (rule MMI˙axaddcl)
from S17 have S32: (A ∈ ℂ ∧ D ∈ ℂ) ⟶ (A · D) ∈ ℂ .
from S18 have S33: (C ∈ ℂ ∧ B ∈ ℂ) ⟶ (C · B) ∈ ℂ .
from S33 have S34: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (C · B) ∈ ℂ
by (rule MMI˙ancoms)
from S31 S32 S34 have S35:
((A ∈ ℂ ∧ D ∈ ℂ) ∧ (B ∈ ℂ ∧ C ∈ ℂ)) ⟶
((A · D) + (C · B)) ∈ ℂ by (rule MMI˙syl2an)
from S35 have S36: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · D) + (C · B)) ∈ ℂ by (rule MMI˙an42s)
from S24 S30 S36 have S37:
((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A · C) + (D · B)) + (- ((A · D) + (C · B)))) =
(((A · C) + (D · B)) - ((A · D) + (C · B)))
by (rule MMI˙sylanc)
from S8 S23 S37 have S38: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + ((- B))) · (C + (- D))) =
(((A · C) + (D · B)) - ((A · D) + (C · B)))
by (rule MMI˙3eqtrd)
from S3 S38 show ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A - B) · (C - D)) =
(((A · C) + (D · B)) - ((A · D) + (C · B)))

by (rule MMI˙eqtr3d)
qed

lemma (in MMIsar0) MMI˙pnpcant:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - (A + C)) = (B - C)
proof -
have S1: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (A ∈ ℂ ∧ C ∈ ℂ)) ⟶
((A + B) - (A + C)) =
((A - A) + (B - C)) by (rule MMI˙sub4t)
from S1 have S2: (A ∈ ℂ ∧ (B ∈ ℂ ∧ C ∈ ℂ)) ⟶
((A + B) - (A + C)) =
((A - A) + (B - C)) by (rule MMI˙anandis)
have S3: A ∈ ℂ ⟶ (A - A) = **0** by (rule MMI˙subidt)
from S3 have S4: A ∈ ℂ ⟶
((A - A) + (B - C)) =
(**0** + (B - C)) by (rule MMI˙opreq1d)
have S5: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B - C) ∈ ℂ by (rule MMI˙subclt)
have S6: (B - C) ∈ ℂ ⟶
(**0** + (B - C)) = (B - C) by (rule MMI˙addid2t)
from S5 S6 have S7: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
(**0** + (B - C)) = (B - C) by (rule MMI˙syl)
from S4 S7 have S8: (A ∈ ℂ ∧ (B ∈ ℂ ∧ C ∈ ℂ)) ⟶
((A - A) + (B - C)) = (B - C) by (rule MMI˙sylan9eq)
from S2 S8 have S9: (A ∈ ℂ ∧ (B ∈ ℂ ∧ C ∈ ℂ)) ⟶
((A + B) - (A + C)) = (B - C) by (rule MMI˙eqtrd)
from S9 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - (A + C)) = (B - C) by (rule MMI˙3impb)
qed

lemma (in MMIsar0) MMI˙pnpcan2t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + C) - (B + C)) = (A - B)
proof -
have S1: (A ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + C) = (C + A) by (rule MMI˙axaddcom)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + C) = (C + A) by (rule MMI˙3adant2)
have S3: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B + C) = (C + B) by (rule MMI˙axaddcom)
from S3 have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B + C) = (C + B) by (rule MMI˙3adant1)
from S2 S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + C) - (B + C)) =
((C + A) - (C + B)) by (rule MMI˙opreq12d)
have S6: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶
((C + A) - (C + B)) = (A - B) by (rule MMI˙pnpcant)
from S6 have S7: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((C + A) - (C + B)) = (A - B) by (rule MMI˙3coml)

from S5 S7 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + C) - (B + C)) = (A - B) by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙pnncant:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - (A - C)) = (B + C)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ ∧ (- C) ∈ ℂ) ⟶
((A + B) - (A + (- C))) =
(B - (- C)) by (rule MMI˙pnpcant)
have S2: C ∈ ℂ ⟶ (- C) ∈ ℂ by (rule MMI˙negclt)
from S1 S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - (A + (- C))) =
(B - (- C)) by (rule MMI˙syl3an3)
have S4: (A ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (- C)) = (A - C) by (rule MMI˙negsubt)
from S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (- C)) = (A - C) by (rule MMI˙3adant2)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - (A + (- C))) =
((A + B) - (A - C)) by (rule MMI˙opreq2d)
have S7: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B - (- C)) = (B + C) by (rule MMI˙subnegt)
from S7 have S8: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B - (- C)) = (B + C) by (rule MMI˙3adant1)
from S3 S6 S8 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - (A - C)) = (B + C) by (rule MMI˙3eqtr3d)
qed

lemma (in MMIsar0) MMI˙ppncant:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) + (C - B)) = (A + C)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A + B) = (B + A) by (rule MMI˙axaddcom)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + B) = (B + A) by (rule MMI˙3adant3)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - (B - C)) =
((B + A) - (B - C)) by (rule MMI˙opreq1d)
have S4: ((A + B) ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - (B - C)) =
((A + B) + (C - B)) by (rule MMI˙subsub2t)
have S5: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + B) ∈ ℂ by (rule MMI˙axaddcl)
from S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + B) ∈ ℂ by (rule MMI˙3adant3)
have S7: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ B ∈ ℂ by (rule MMI˙3simp2)

have S8: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ C ∈ ℂ by (rule MMI˙3simp3)
from S4 S6 S7 S8 have S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - (B - C)) =
((A + B) + (C - B)) by (rule MMI˙syl3anc)
have S10: (B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ) ⟶
((B + A) - (B - C)) = (A + C) by (rule MMI˙pnncant)
from S10 have S11: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((B + A) - (B - C)) = (A + C) by (rule MMI˙3com12)
from S3 S9 S11 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) + (C - B)) = (A + C) by (rule MMI˙3eqtr3d)
qed

lemma (in MMIsar0) MMI˙pnncan: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows ((A + B) - (A - C)) = (B + C)
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - (A - C)) = (B + C) by (rule MMI˙pnncant)
from S1 S2 S3 S4 show ((A + B) - (A - C)) = (B + C) by (rule MMI˙mp3an)
qed

lemma (in MMIsar0) MMI˙mulcan: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ and
A4: A ≠ **0**
shows (A · B) = (A · C) ⟷ B = C
proof -
from A1 have S1: A ∈ ℂ.
from A4 have S2: A ≠ **0**.
from S1 S2 have S3: ∃ x ∈ ℂ . (A · x) = **1** by (rule MMI˙recex)
from A1 have S4: A ∈ ℂ.
from A2 have S5: B ∈ ℂ.
{ fix x
have S6: (x ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶
((x · A) · B) = (x · (A · B)) by (rule MMI˙axmulass)
from S5 S6 have S7: (x ∈ ℂ ∧ A ∈ ℂ) ⟶
((x · A) · B) = (x · (A · B)) by (rule MMI˙mp3an3)
from A3 have S8: C ∈ ℂ.
have S9: (x ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ) ⟶
((x · A) · C) = (x · (A · C)) by (rule MMI˙axmulass)
from S8 S9 have S10: (x ∈ ℂ ∧ A ∈ ℂ) ⟶
((x · A) · C) = (x · (A · C)) by (rule MMI˙mp3an3)
from S7 S10 have S11: (x ∈ ℂ ∧ A ∈ ℂ) ⟶
(((x · A) · B) =
((x · A) · C) ⟷

$(x \cdot (A \cdot B)) =$
$(x \cdot (A \cdot C)))$ by (rule MMI˙eqeq12d)
from S4 S11 have S12: $x \in \mathbb{C} \longrightarrow$
$(((x \cdot A) \cdot B) =$
$((x \cdot A) \cdot C) \longleftrightarrow$
$(x \cdot (A \cdot B)) =$
$(x \cdot (A \cdot C)))$ by (rule MMI˙mpan2)
have S13:
$(A \cdot B) = (A \cdot C) \longrightarrow$
$(x \cdot (A \cdot B)) = (x \cdot (A \cdot C))$ by (rule MMI˙opreq2)
from S12 S13 have S14: $x \in \mathbb{C} \longrightarrow$
$((A \cdot B) = (A \cdot C) \longrightarrow ((x \cdot A) \cdot B) =$
$((x \cdot A) \cdot C))$ by (rule MMI˙syl5bir)
from S14 have S15:
$(x \in \mathbb{C} \wedge (A \cdot x) = \mathbf{1}) \longrightarrow ((A \cdot B) =$
$(A \cdot C) \longrightarrow ((x \cdot A) \cdot B) =$
$((x \cdot A) \cdot C))$ by (rule MMI˙adantr)
from A1 have S16: $A \in \mathbb{C}$.
have S17: $(A \in \mathbb{C} \wedge x \in \mathbb{C}) \longrightarrow$
$(A \cdot x) = (x \cdot A)$ by (rule MMI˙axmulcom)
from S16 S17 have S18: $x \in \mathbb{C} \longrightarrow (A \cdot x) = (x \cdot A)$
by (rule MMI˙mpan)
from S18 have S19: $x \in \mathbb{C} \longrightarrow$
$((A \cdot x) = \mathbf{1} \longleftrightarrow (x \cdot A) = \mathbf{1})$ by (rule MMI˙eqeq1d)
have S20: $(x \cdot A) =$
$\mathbf{1} \longrightarrow ((x \cdot A) \cdot B) = (\mathbf{1} \cdot B)$ by (rule MMI˙opreq1)
from A2 have S21: $B \in \mathbb{C}$.
from S21 have S22: $(\mathbf{1} \cdot B) = B$ by (rule MMI˙mulid2)
from S20 S22 have S23: $(x \cdot A) = \mathbf{1} \longrightarrow ((x \cdot A) \cdot B) = B$
by (rule MMI˙syl6eq)
have S24: $(x \cdot A) =$
$\mathbf{1} \longrightarrow ((x \cdot A) \cdot C) = (\mathbf{1} \cdot C)$ by (rule MMI˙opreq1)
from A3 have S25: $C \in \mathbb{C}$.
from S25 have S26: $(\mathbf{1} \cdot C) = C$ by (rule MMI˙mulid2)
from S24 S26 have S27: $(x \cdot A) = \mathbf{1} \longrightarrow ((x \cdot A) \cdot C) = C$
by (rule MMI˙syl6eq)
from S23 S27 have S28: $(x \cdot A) = \mathbf{1} \longrightarrow$
$(((x \cdot A) \cdot B) =$
$((x \cdot A) \cdot C) \longleftrightarrow B = C)$ by (rule MMI˙eqeq12d)
from S19 S28 have S29: $x \in \mathbb{C} \longrightarrow$
$((A \cdot x) = \mathbf{1} \longrightarrow$
$(((x \cdot A) \cdot B) =$
$((x \cdot A) \cdot C) \longleftrightarrow B = C))$ by (rule MMI˙syl6bi)
from S29 have S30:
$(x \in \mathbb{C} \wedge (A \cdot x) = \mathbf{1}) \longrightarrow$
$(((x \cdot A) \cdot B) =$
$((x \cdot A) \cdot C) \longleftrightarrow B = C)$ by (rule MMI˙imp)
from S15 S30 have S31:
$(x \in \mathbb{C} \wedge (A \cdot x) = \mathbf{1}) \longrightarrow$

((A · B) = (A · C) ⟶ B = C) by (rule MMI˙sylibd)
from S31 have x ∈ ℂ ⟶
((A · x) = **1** ⟶ ((A · B) = (A · C) ⟶ B = C))
by (rule MMI˙ex)
} then have S32: ∀ x. x ∈ ℂ ⟶
((A · x) = **1** ⟶ ((A · B) = (A · C) ⟶ B = C))
by auto
from S32 have S33: (∃ x ∈ ℂ . (A · x) = **1**) ⟶
((A · B) = (A · C) ⟶ B = C) by (rule MMI˙r19˙23aiv)
from S3 S33 have S34: (A · B) = (A · C) ⟶ B = C
by (rule MMI˙ax˙mp)
have S35: B = C ⟶ (A · B) = (A · C) by (rule MMI˙opreq2)
from S34 S35 show (A · B) = (A · C) ⟷ B = C by (rule MMI˙impbi)
qed

lemma (in MMIsar0) MMI˙mulcant2: assumes A1: A ≠ **0**
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · B) = (A · C) ⟷ B = C)
proof -
have S1: A =
if (A ∈ ℂ , A , **1**) ⟶
(A · B) =
(if (A ∈ ℂ , A , **1**) · B) by (rule MMI˙opreq1)
have S2: A =
if (A ∈ ℂ , A , **1**) ⟶
(A · C) =
(if (A ∈ ℂ , A , **1**) · C) by (rule MMI˙opreq1)
from S1 S2 have S3: A =
if (A ∈ ℂ , A , **1**) ⟶
((A · B) =
(A · C) ⟷
(if (A ∈ ℂ , A , **1**) · B) =
(if (A ∈ ℂ , A , **1**) · C)) by (rule MMI˙eqeq12d)
from S3 have S4: A =
if (A ∈ ℂ , A , **1**) ⟶
(((A · B) = (A · C) ⟷ B = C) ⟷
((if (A ∈ ℂ , A , **1**) · B) =
(if (A ∈ ℂ , A , **1**) · C) ⟷
B = C)) by (rule MMI˙bibi1d)
have S5: B =
if (B ∈ ℂ , B , **1**) ⟶
(if (A ∈ ℂ , A , **1**) · B) =
(if (A ∈ ℂ , A , **1**) · if (B ∈ ℂ , B , **1**)) by (rule MMI˙opreq2)
from S5 have S6: B =
if (B ∈ ℂ , B , **1**) ⟶
((if (A ∈ ℂ , A , **1**) · B) =
(if (A ∈ ℂ , A , **1**) · C) ⟷
(if (A ∈ ℂ , A , **1**) · if (B ∈ ℂ , B , **1**)) =
(if (A ∈ ℂ , A , **1**) · C)) by (rule MMI˙eqeq1d)

have S7: B =
if (B ∈ ℂ , B , **1**) ⟶
(B = C ⟷ if (B ∈ ℂ , B , **1**) = C) by (rule MMI˙eqeq1)
from S6 S7 have S8: B =
if (B ∈ ℂ , B , **1**) ⟶
(((if (A ∈ ℂ , A , **1**) · B) = (if (A ∈ ℂ , A , **1**) · C) ⟷ B = C) ⟷
((if (A ∈ ℂ , A , **1**) · if (B ∈ ℂ , B , **1**)) =
(if (A ∈ ℂ , A , **1**) · C) ⟷
if (B ∈ ℂ , B , **1**) = C)) by (rule MMI˙bibi12d)
have S9: C =
if (C ∈ ℂ , C , **1**) ⟶
(if (A ∈ ℂ , A , **1**) · C) =
(if (A ∈ ℂ , A , **1**) · if (C ∈ ℂ , C , **1**)) by (rule MMI˙opreq2)
from S9 have S10: C =
if (C ∈ ℂ , C , **1**) ⟶
((if (A ∈ ℂ , A , **1**) · if (B ∈ ℂ , B , **1**)) =
(if (A ∈ ℂ , A , **1**) · C) ⟷
(if (A ∈ ℂ , A , **1**) · if (B ∈ ℂ , B , **1**)) =
(if (A ∈ ℂ , A , **1**) · if (C ∈ ℂ , C , **1**))) by (rule MMI˙eqeq2d)
have S11: C =
if (C ∈ ℂ , C , **1**) ⟶
(if (B ∈ ℂ , B , **1**) =
C ⟷
if (B ∈ ℂ , B , **1**) =
if (C ∈ ℂ , C , **1**)) by (rule MMI˙eqeq2)
from S10 S11 have S12: C =
if (C ∈ ℂ , C , **1**) ⟶
(((if (A ∈ ℂ , A , **1**) · if (B ∈ ℂ , B , **1**)) = (if (A ∈ ℂ , A , **1**) · C) ⟷
if (B ∈ ℂ , B , **1**) = C) ⟷
((if (A ∈ ℂ , A , **1**) · if (B ∈ ℂ , B , **1**)) =
(if (A ∈ ℂ , A , **1**) · if (C ∈ ℂ , C , **1**)) ⟷
if (B ∈ ℂ , B , **1**) =
if (C ∈ ℂ , C , **1**))) by (rule MMI˙bibi12d)
have S13: **1** ∈ ℂ by (rule MMI˙1cn)
from S13 have S14: if (A ∈ ℂ , A , **1**) ∈ ℂ by (rule MMI˙elimel)
have S15: **1** ∈ ℂ by (rule MMI˙1cn)
from S15 have S16: if (B ∈ ℂ , B , **1**) ∈ ℂ by (rule MMI˙elimel)
have S17: **1** ∈ ℂ by (rule MMI˙1cn)
from S17 have S18: if (C ∈ ℂ , C , **1**) ∈ ℂ by (rule MMI˙elimel)
have S19: A =
if (A ∈ ℂ , A , **1**) ⟶
(A ≠ **0** ⟷ if (A ∈ ℂ , A , **1**) ≠ **0**) by (rule MMI˙neeq1)
have S20: **1** =
if (A ∈ ℂ , A , **1**) ⟶
(**1** ≠ **0** ⟷ if (A ∈ ℂ , A , **1**) ≠ **0**) by (rule MMI˙neeq1)
from A1 have S21: A ≠ **0**.
have S22: **1** ≠ **0** by (rule MMI˙ax1ne0)
from S19 S20 S21 S22 have S23: if (A ∈ ℂ , A , **1**) ≠ **0** by (rule MMI˙keephyp)
from S14 S16 S18 S23 have S24: (if (A ∈ ℂ , A , **1**) · if (B ∈ ℂ , B , **1**)) =

(if (A ∈ ℂ , A , **1**) · if (C ∈ ℂ , C , **1**)) ⟷
if (B ∈ ℂ , B , **1**) =
if (C ∈ ℂ , C , **1**) by (rule MMI˙mulcan)
from S4 S8 S12 S24 show (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · B) = (A · C) ⟷ B = C) by (rule MMI˙dedth3h)
qed

lemma (in MMIsar0) MMI˙mulcant:
shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ A ≠ **0**) ⟶
((A · B) = (A · C) ⟷ B = C)
proof -
have S1: A =
if (A ≠ **0** , A , **1**) ⟶
(A ∈ ℂ ⟷ if (A ≠ **0** , A , **1**) ∈ ℂ) by (rule MMI˙eleq1)
have S2: A =
if (A ≠ **0** , A , **1**) ⟶
(B ∈ ℂ ⟷ B ∈ ℂ) by (rule MMI˙pm4˙2i)
have S3: A =
if (A ≠ **0** , A , **1**) ⟶
(C ∈ ℂ ⟷ C ∈ ℂ) by (rule MMI˙pm4˙2i)
from S1 S2 S3 have S4: A =
if (A ≠ **0** , A , **1**) ⟶
((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟷
(if (A ≠ **0** , A , **1**) ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ)) by (rule MMI˙3anbi123d)
have S5: A =
if (A ≠ **0** , A , **1**) ⟶
(A · B) =
(if (A ≠ **0** , A , **1**) · B) by (rule MMI˙opreq1)
have S6: A =
if (A ≠ **0** , A , **1**) ⟶
(A · C) =
(if (A ≠ **0** , A , **1**) · C) by (rule MMI˙opreq1)
from S5 S6 have S7: A =
if (A ≠ **0** , A , **1**) ⟶
((A · B) =
(A · C) ⟷
(if (A ≠ **0** , A , **1**) · B) =
(if (A ≠ **0** , A , **1**) · C)) by (rule MMI˙eqeq12d)
from S7 have S8: A =
if (A ≠ **0** , A , **1**) ⟶
(((A · B) = (A · C) ⟷ B = C) ⟷
((if (A ≠ **0** , A , **1**) · B) =
(if (A ≠ **0** , A , **1**) · C) ⟷
B = C)) by (rule MMI˙bibi1d)
from S4 S8 have S9: A =
if (A ≠ **0** , A , **1**) ⟶
(((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A · B) = (A · C) ⟷ B = C)) ⟷
((if (A ≠ **0** , A , **1**) ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((if (A ≠ **0** , A , **1**) · B) =

(if (A $\neq$ **0** , A , **1**) $\cdot$ C) $\longleftrightarrow$
B = C))) by (rule MMI˙imbi12d)
have S10: if (A $\neq$ **0** , A , **1**) $\neq$ **0** by (rule MMI˙elimne0)
from S10 have S11: (if (A $\neq$ **0** , A , **1**) $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((if (A $\neq$ **0** , A , **1**) $\cdot$ B) =
(if (A $\neq$ **0** , A , **1**) $\cdot$ C) $\longleftrightarrow$ B = C) by (rule MMI˙mulcant2)
from S9 S11 have S12: A $\neq$ **0** $\longrightarrow$
((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A $\cdot$ B) = (A $\cdot$ C) $\longleftrightarrow$ B = C)) by (rule MMI˙dedth)
from S12 show ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ A $\neq$ **0**) $\longrightarrow$
((A $\cdot$ B) = (A $\cdot$ C) $\longleftrightarrow$ B = C) by (rule MMI˙impcom)
qed

lemma (in MMIsar0) MMI˙mulcan2t:
shows ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((A $\cdot$ C) = (B $\cdot$ C) $\longleftrightarrow$ A = B)
proof -
have S1: (A $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(A $\cdot$ C) = (C $\cdot$ A) by (rule MMI˙axmulcom)
from S1 have S2: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(A $\cdot$ C) = (C $\cdot$ A) by (rule MMI˙3adant2)
have S3: (B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(B $\cdot$ C) = (C $\cdot$ B) by (rule MMI˙axmulcom)
from S3 have S4: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(B $\cdot$ C) = (C $\cdot$ B) by (rule MMI˙3adant1)
from S2 S4 have S5: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A $\cdot$ C) =
(B $\cdot$ C) $\longleftrightarrow$ (C $\cdot$ A) = (C $\cdot$ B)) by (rule MMI˙eqeq12d)
from S5 have S6: ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((A $\cdot$ C) =
(B $\cdot$ C) $\longleftrightarrow$ (C $\cdot$ A) = (C $\cdot$ B)) by (rule MMI˙adantr)
have S7: ((C $\in$ $\mathbb{C}$ $\wedge$ A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((C $\cdot$ A) = (C $\cdot$ B) $\longleftrightarrow$ A = B) by (rule MMI˙mulcant)
from S7 have S8: (C $\in$ $\mathbb{C}$ $\wedge$ A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
(C $\neq$ **0** $\longrightarrow$
((C $\cdot$ A) = (C $\cdot$ B) $\longleftrightarrow$ A = B)) by (rule MMI˙ex)
from S8 have S9: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(C $\neq$ **0** $\longrightarrow$
((C $\cdot$ A) = (C $\cdot$ B) $\longleftrightarrow$ A = B)) by (rule MMI˙3coml)
from S9 have S10: ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((C $\cdot$ A) = (C $\cdot$ B) $\longleftrightarrow$ A = B) by (rule MMI˙imp)
from S6 S10 show ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((A $\cdot$ C) = (B $\cdot$ C) $\longleftrightarrow$ A = B) by (rule MMI˙bitrd)
qed

lemma (in MMIsar0) MMI˙mul0or: assumes A1: A $\in$ $\mathbb{C}$ and
A2: B $\in$ $\mathbb{C}$
shows (A $\cdot$ B) = **0** $\longleftrightarrow$ (A = **0** $\vee$ B = **0**)
proof -

have S1: $A \neq \mathbf{0} \longleftrightarrow \neg (A = \mathbf{0})$ by (rule MMI˙df˙ne)
from A1 have S2: $A \in \mathbb{C}$.
from A2 have S3: $B \in \mathbb{C}$.
have S4: $\mathbf{0} \in \mathbb{C}$ by (rule MMI˙0cn)
from S2 S3 S4 have S5: $A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge \mathbf{0} \in \mathbb{C}$ by (rule MMI˙3pm3˙2i)
have S6: $((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge \mathbf{0} \in \mathbb{C}) \wedge A \neq \mathbf{0}) \longrightarrow$
$((A \cdot B) = (A \cdot \mathbf{0}) \longleftrightarrow B = \mathbf{0})$ by (rule MMI˙mulcant)
from S5 S6 have S7: $A \neq \mathbf{0} \longrightarrow$
$((A \cdot B) = (A \cdot \mathbf{0}) \longleftrightarrow B = \mathbf{0})$ by (rule MMI˙mpan)
from A1 have S8: $A \in \mathbb{C}$.
from S8 have S9: $(A \cdot \mathbf{0}) = \mathbf{0}$ by (rule MMI˙mul01)
from S9 have S10: $(A \cdot B) = (A \cdot \mathbf{0}) \longleftrightarrow (A \cdot B) = \mathbf{0}$ by (rule MMI˙eqeq2i)
from S7 S10 have S11: $A \neq \mathbf{0} \longrightarrow ((A \cdot B) = \mathbf{0} \longleftrightarrow B = \mathbf{0})$ by (rule MMI˙syl5bbr)
from S11 have S12: $A \neq \mathbf{0} \longrightarrow ((A \cdot B) = \mathbf{0} \longrightarrow B = \mathbf{0})$ by (rule MMI˙biimpd)
from S1 S12 have S13: $\neg (A =$
$\mathbf{0}) \longrightarrow ((A \cdot B) = \mathbf{0} \longrightarrow B = \mathbf{0})$ by (rule MMI˙sylbir)
from S13 have S14: $(A \cdot B) =$
$\mathbf{0} \longrightarrow (\neg (A = \mathbf{0}) \longrightarrow B = \mathbf{0})$ by (rule MMI˙com12)
from S14 have S15: $(A \cdot B) = \mathbf{0} \longrightarrow (A = \mathbf{0} \vee B = \mathbf{0})$ by (rule MMI˙orrd)
have S16: $A = \mathbf{0} \longrightarrow (A \cdot B) = (\mathbf{0} \cdot B)$ by (rule MMI˙opreq1)
from A2 have S17: $B \in \mathbb{C}$.
from S17 have S18: $(\mathbf{0} \cdot B) = \mathbf{0}$ by (rule MMI˙mul02)
from S16 S18 have S19: $A = \mathbf{0} \longrightarrow (A \cdot B) = \mathbf{0}$ by (rule MMI˙syl6eq)
have S20: $B = \mathbf{0} \longrightarrow (A \cdot B) = (A \cdot \mathbf{0})$ by (rule MMI˙opreq2)
from S9 have S21: $(A \cdot \mathbf{0}) = \mathbf{0}$.
from S20 S21 have S22: $B = \mathbf{0} \longrightarrow (A \cdot B) = \mathbf{0}$ by (rule MMI˙syl6eq)
from S19 S22 have S23: $(A = \mathbf{0} \vee B = \mathbf{0}) \longrightarrow (A \cdot B) = \mathbf{0}$ by (rule MMI˙jaoi)
from S15 S23 show $(A \cdot B) = \mathbf{0} \longleftrightarrow (A = \mathbf{0} \vee B = \mathbf{0})$ by (rule MMI˙impbi)
qed

lemma (in MMIsar0) MMI˙msq0: assumes A1: $A \in \mathbb{C}$
shows $(A \cdot A) = \mathbf{0} \longleftrightarrow A = \mathbf{0}$
proof -
from A1 have S1: $A \in \mathbb{C}$.
from A1 have S2: $A \in \mathbb{C}$.
from S1 S2 have S3: $(A \cdot A) = \mathbf{0} \longleftrightarrow (A = \mathbf{0} \vee A = \mathbf{0})$ by (rule MMI˙mul0or)
have S4: $(A = \mathbf{0} \vee A = \mathbf{0}) \longleftrightarrow A = \mathbf{0}$ by (rule MMI˙oridm)
from S3 S4 show $(A \cdot A) = \mathbf{0} \longleftrightarrow A = \mathbf{0}$ by (rule MMI˙bitr)
qed

lemma (in MMIsar0) MMI˙mul0ort:
shows $(A \in \mathbb{C} \wedge B \in \mathbb{C}) \longrightarrow$
$((A \cdot B) = \mathbf{0} \longleftrightarrow (A = \mathbf{0} \vee B = \mathbf{0}))$
proof -
have S1: $A =$
$\text{if} (A \in \mathbb{C} , A , \mathbf{0}) \longrightarrow$
$(A \cdot B) =$
$(\text{if} (A \in \mathbb{C} , A , \mathbf{0}) \cdot B)$ by (rule MMI˙opreq1)

from S1 have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
((A · B) =
0 ⟷ (if (A ∈ ℂ , A , **0**) · B) = **0**) by (rule MMI˙eqeq1d)
have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
(A = **0** ⟷ if (A ∈ ℂ , A , **0**) = **0**) by (rule MMI˙eqeq1)
from S3 have S4: A =
if (A ∈ ℂ , A , **0**) ⟶
((A = **0** ∨ B = **0**) ⟷
(if (A ∈ ℂ , A , **0**) = **0** ∨ B = **0**)) by (rule MMI˙orbi1d)
from S2 S4 have S5: A =
if (A ∈ ℂ , A , **0**) ⟶
(((A · B) = **0** ⟷ (A = **0** ∨ B = **0**)) ⟷
((if (A ∈ ℂ , A , **0**) · B) =
0 ⟷
(if (A ∈ ℂ , A , **0**) =
0 ∨ B = **0**))) by (rule MMI˙bibi12d)
have S6: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) · B) =
(if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
from S6 have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) · B) =
0 ⟷
(if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) =
0) by (rule MMI˙eqeq1d)
have S8: B =
if (B ∈ ℂ , B , **0**) ⟶
(B = **0** ⟷ if (B ∈ ℂ , B , **0**) = **0**) by (rule MMI˙eqeq1)
from S8 have S9: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) = **0** ∨ B = **0**) ⟷
(if (A ∈ ℂ , A , **0**) =
0 ∨ if (B ∈ ℂ , B , **0**) = **0**)) by (rule MMI˙orbi2d)
from S7 S9 have S10: B =
if (B ∈ ℂ , B , **0**) ⟶
(((if (A ∈ ℂ , A , **0**) · B) = **0** ⟷ (if (A ∈ ℂ , A , **0**) = **0** ∨ B = **0**)) ⟷
((if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) =
0 ⟷
(if (A ∈ ℂ , A , **0**) =
0 ∨ if (B ∈ ℂ , B , **0**) = **0**))) by (rule MMI˙bibi12d)
have S11: **0** ∈ ℂ by (rule MMI˙0cn)
from S11 have S12: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S13: **0** ∈ ℂ by (rule MMI˙0cn)
from S13 have S14: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
from S12 S14 have S15: (if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) =
0 ⟷

(if (A ∈ ℂ , A , **0**) =
0 ∨ if (B ∈ ℂ , B , **0**) = **0**) by (rule MMI˙mul0or)
from S5 S10 S15 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A · B) = **0** ⟷ (A = **0** ∨ B = **0**)) by (rule MMI˙dedth2h)
qed

lemma (in MMIsar0) MMI˙muln0bt:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A ≠ **0** ∧ B ≠ **0**) ⟷ (A · B) ≠ **0**)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A · B) = **0** ⟷ (A = **0** ∨ B = **0**)) by (rule MMI˙mul0ort)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(¬ ((A · B) = **0**) ⟷
¬ ((A = **0** ∨ B = **0**))) by (rule MMI˙negbid)
have S3: ¬ ((A = **0** ∨ B = **0**)) ⟷
(¬ (A = **0**) ∧ ¬ (B = **0**)) by (rule MMI˙ioran)
from S2 S3 have S4: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((¬ (A = **0**) ∧ ¬ (B = **0**)) ⟷
¬ ((A · B) = **0**)) by (rule MMI˙syl6rbb)
have S5: A ≠ **0** ⟷ ¬ (A = **0**) by (rule MMI˙df˙ne)
have S6: B ≠ **0** ⟷ ¬ (B = **0**) by (rule MMI˙df˙ne)
from S5 S6 have S7: (A ≠ **0** ∧ B ≠ **0**) ⟷
(¬ (A = **0**) ∧ ¬ (B = **0**)) by (rule MMI˙anbi12i)
have S8: (A · B) ≠ **0** ⟷ ¬ ((A · B) = **0**) by (rule MMI˙df˙ne)
from S4 S7 S8 show (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A ≠ **0** ∧ B ≠ **0**) ⟷ (A · B) ≠ **0**) by (rule MMI˙3bitr4g)
qed

lemma (in MMIsar0) MMI˙muln0: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: A ≠ **0** and
A4: B ≠ **0**
shows (A · B) ≠ **0**
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: A ≠ **0**.
from A4 have S4: B ≠ **0**.
from S3 S4 have S5: A ≠ **0** ∧ B ≠ **0** by (rule MMI˙pm3˙2i)
have S6: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A ≠ **0** ∧ B ≠ **0**) ⟷ (A · B) ≠ **0**) by (rule MMI˙muln0bt)
from S5 S6 have S7: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A · B) ≠ **0** by (rule MMI˙mpbii)
from S1 S2 S7 show (A · B) ≠ **0** by (rule MMI˙mp2an)
qed

lemma (in MMIsar0) MMI˙receu: assumes A1: A ∈ ℂ and

A2: B ∈ ℂ and
A3: A ≠ **0**
shows ∃! x . x ∈ ℂ ∧ (A · x) = B
proof -
{ fix x y
have S1: x = y ⟶ (A · x) = (A · y) by (rule MMI˙opreq2)
from S1 have S2: x = y ⟶ ((A · x) = B ⟷ (A · y) = B)
by (rule MMI˙eqeq1d)
} then have S2: ∀ x y. x = y ⟶ ((A · x) = B ⟷ (A · y) = B)
by simp
from S2 have S3:
(∃! x . x ∈ ℂ ∧ (A · x) = B) ⟷
((∃ x ∈ ℂ . (A · x) = B) ∧
(∀ x ∈ ℂ . ∀ y ∈ ℂ . (((A · x) = B ∧ (A · y) = B) ⟶ x = y)))
by (rule MMI˙reu4)
from A1 have S4: A ∈ ℂ.
from A3 have S5: A ≠ **0**.
from S4 S5 have S6: ∃ y ∈ ℂ . (A · y) = **1** by (rule MMI˙recex)
from A2 have S7: B ∈ ℂ.
{ fix y
have S8: (y ∈ ℂ ∧ B ∈ ℂ) ⟶ (y · B) ∈ ℂ by (rule MMI˙axmulcl)
from S7 S8 have S9: y ∈ ℂ ⟶ (y · B) ∈ ℂ by (rule MMI˙mpan2)
have S10: (y · B) ∈ ℂ ⟷
(∃ x ∈ ℂ . x = (y · B)) by (rule MMI˙risset)
from S9 S10 have S11: y ∈ ℂ ⟶ (∃ x ∈ ℂ . x = (y · B))
by (rule MMI˙sylib)
{ fix x
have S12: x = (y · B) ⟶
(A · x) = (A · (y · B)) by (rule MMI˙opreq2)
from A1 have S13: A ∈ ℂ.
from A2 have S14: B ∈ ℂ.
have S15: (A ∈ ℂ ∧ y ∈ ℂ ∧ B ∈ ℂ) ⟶
((A · y) · B) = (A · (y · B)) by (rule MMI˙axmulass)
from S13 S14 S15 have S16: y ∈ ℂ ⟶
((A · y) · B) = (A · (y · B)) by (rule MMI˙mp3an13)
from S16 have S17: y ∈ ℂ ⟶
(A · (y · B)) = ((A · y) · B) by (rule MMI˙eqcomd)
from S12 S17 have S18: (y ∈ ℂ ∧ x =
(y · B)) ⟶
(A · x) = ((A · y) · B) by (rule MMI˙sylan9eqr)
have S19: (A · y) =
1 ⟶ ((A · y) · B) = (**1** · B) by (rule MMI˙opreq1)
from A2 have S20: B ∈ ℂ.
from S20 have S21: (**1** · B) = B by (rule MMI˙mulid2)
from S19 S21 have S22: (A · y) = **1** ⟶ ((A · y) · B) = B
by (rule MMI˙syl6eq)
from S18 S22 have S23:
((A · y) = **1** ∧ (y ∈ ℂ ∧ x =
(y · B))) ⟶ (A · x) = B by (rule MMI˙sylan9eqr)

from S23 have S24:
$(A \cdot y) = \mathbf{1} \longrightarrow (y \in \mathbb{C} \longrightarrow$
$(x = (y \cdot B) \longrightarrow (A \cdot x) = B))$ by (rule MMI˙exp32)
from S24 have S25: $(y \in \mathbb{C} \wedge (A \cdot y) =$
$\mathbf{1}) \longrightarrow$
$(x = (y \cdot B) \longrightarrow (A \cdot x) = B)$ by (rule MMI˙impcom)
from S25 have
$(y \in \mathbb{C} \wedge (A \cdot y) = \mathbf{1}) \longrightarrow (x \in \mathbb{C} \longrightarrow$
$(x = (y \cdot B) \longrightarrow (A \cdot x) = B))$ by (rule MMI˙a1d)
} then have S26:
$\forall\ x . (y \in \mathbb{C} \wedge (A \cdot y) = \mathbf{1}) \longrightarrow (x \in \mathbb{C} \longrightarrow$
$(x = (y \cdot B) \longrightarrow (A \cdot x) = B))$ by simp
from S26 have S27:
$(y \in \mathbb{C} \wedge (A \cdot y) = \mathbf{1}) \longrightarrow$
$(\forall\ x \in \mathbb{C} . (x = (y \cdot B) \longrightarrow (A \cdot x) = B))$ by (rule MMI˙r19˙21aiv)
from S27 have S28: $y \in \mathbb{C} \longrightarrow$
$((A \cdot y) = \mathbf{1} \longrightarrow$
$(\forall\ x \in \mathbb{C} . (x = (y \cdot B) \longrightarrow (A \cdot x) = B)))$ by (rule MMI˙ex)
have S29: $(\forall\ x \in \mathbb{C} . (x = (y \cdot B) \longrightarrow (A \cdot x) = B)) \longrightarrow$
$((\exists\ x \in \mathbb{C} . x = (y \cdot B)) \longrightarrow$
$(\exists\ x \in \mathbb{C} . (A \cdot x) = B))$ by (rule MMI˙r19˙22)
from S28 S29 have S30:
$y \in \mathbb{C} \longrightarrow ((A \cdot y) = \mathbf{1} \longrightarrow$
$((\exists\ x \in \mathbb{C} . x = (y \cdot B)) \longrightarrow$
$(\exists\ x \in \mathbb{C} . (A \cdot x) = B)))$ by (rule MMI˙syl6)
from S11 S30 have
$y \in \mathbb{C} \longrightarrow ((A \cdot y) = \mathbf{1} \longrightarrow (\exists\ x \in \mathbb{C} . (A \cdot x) = B))$
by (rule MMI˙mpid)
} then have S31:
$\forall\ y . y \in \mathbb{C} \longrightarrow ((A \cdot y) = \mathbf{1} \longrightarrow (\exists\ x \in \mathbb{C} . (A \cdot x) = B))$
by simp
from S31 have S32: $(\exists\ y \in \mathbb{C} . (A \cdot y) =$
$\mathbf{1}) \longrightarrow (\exists\ x \in \mathbb{C} . (A \cdot x) = B)$ by (rule MMI˙r19˙23aiv)
from S6 S32 have S33: $\exists\ x \in \mathbb{C} . (A \cdot x) = B$ by (rule MMI˙ax˙mp)
from A1 have S34: $A \in \mathbb{C}$.
from A3 have S35: $A \neq \mathbf{0}$.
{ fix x y
from S35 have S36: $(A \in \mathbb{C} \wedge x \in \mathbb{C} \wedge y \in \mathbb{C}) \longrightarrow$
$((A \cdot x) = (A \cdot y) \longleftrightarrow x = y)$ by (rule MMI˙mulcant2)
have S37:
$((A \cdot x) = B \wedge (A \cdot y) =$
$B) \longrightarrow (A \cdot x) = (A \cdot y)$ by (rule MMI˙eqtr3t)
from S36 S37 have S38: $(A \in \mathbb{C} \wedge x \in \mathbb{C} \wedge y \in \mathbb{C}) \longrightarrow$
$(((A \cdot x) = B \wedge (A \cdot y) = B) \longrightarrow$
$x = y)$ by (rule MMI˙syl5bi)
from S34 S38 have $(x \in \mathbb{C} \wedge y \in \mathbb{C}) \longrightarrow$
$(((A \cdot x) = B \wedge (A \cdot y) = B) \longrightarrow$
$x = y)$ by (rule MMI˙mp3an1)
} then have S39: $\forall\ x\ y . (x \in \mathbb{C} \wedge y \in \mathbb{C}) \longrightarrow$

(((A · x) = B ∧ (A · y) = B) ⟶
x = y) by auto
from S39 have S40:
∀ x ∈ ℂ . ∀ y ∈ ℂ . (((A · x) = B ∧ (A · y) = B) ⟶
x = y) by (rule MMI˙rgen2)
from S3 S33 S40 show ∃! x . x ∈ ℂ ∧ (A · x) = B by (rule MMI˙mpbir2an)
qed

lemma (in MMIsar0) MMI˙divval: assumes A ∈ ℂ B ∈ ℂ B ≠ **0**
shows A / B = ⋃ − x ∈ ℂ . B · x = A ″
using cdiv˙def by simp

lemma (in MMIsar0) MMI˙divmul: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ and
A4: B ≠ **0**
shows (A / B) = C ⟷ (B · C) = A
proof -
from A3 have S1: C ∈ ℂ.
{ fix x
have S2: x =
C ⟶ ((A / B) = x ⟷ (A / B) = C) by (rule MMI˙eqeq2)
have S3: x = C ⟶ (B · x) = (B · C) by (rule MMI˙opreq2)
from S3 have S4: x =
C ⟶ ((B · x) = A ⟷ (B · C) = A) by (rule MMI˙eqeq1d)
from S2 S4 have
x = C ⟶
(((A / B) = x ⟷ (B · x) = A) ⟷
((A / B) = C ⟷ (B · C) = A)) by (rule MMI˙bibi12d)
} then have S5: ∀x. x = C ⟶
(((A / B) = x ⟷ (B · x) = A) ⟷
((A / B) = C ⟷ (B · C) = A))
by simp
from A2 have S6: B ∈ ℂ.
from A1 have S7: A ∈ ℂ.
from A4 have S8: B ≠ **0**.
from S6 S7 S8 have S9: ∃! x . x ∈ ℂ ∧ (B · x) = A by (rule MMI˙receu)
{ fix x
have S10: (x ∈ ℂ ∧ (∃! x . x ∈ ℂ ∧ (B · x) = A)) ⟶
((B · x) =
A ⟷ ⋃ − x ∈ ℂ . (B · x) = A ″ = x) by (rule MMI˙reuuni1)
from S9 S10 have
x ∈ ℂ ⟶ ((B · x) = A ⟷ ⋃ − x ∈ ℂ . (B · x) = A ″ = x)
by (rule MMI˙mpan2)

} then have S11:
$\forall$ x. x $\in \mathbb{C} \longrightarrow$ ((B $\cdot$ x) = A $\longleftrightarrow \bigcup$ − x $\in \mathbb{C}$. (B $\cdot$ x) = A ″ = x)
by blast
from A1 have S12: A $\in \mathbb{C}$.
from A2 have S13: B $\in \mathbb{C}$.
from A4 have S14: B $\neq$ **0**.
from S12 S13 S14 have S15: (A / B) =
$\bigcup$ − x $\in \mathbb{C}$. (B $\cdot$ x) = A ″ by (rule MMI˙divval)
from S15 have S16: $\forall$ x. (A / B) =
x $\longleftrightarrow \bigcup$ − x $\in \mathbb{C}$. (B $\cdot$ x) = A ″ = x by simp
from S11 S16 have S17: $\forall$ x. x $\in \mathbb{C} \longrightarrow$
((A / B) = x $\longleftrightarrow$ (B $\cdot$ x) = A) by (rule MMI˙syl6rbbr)
from S5 S17 have S18: C $\in \mathbb{C} \longrightarrow$
((A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A) by (rule MMI˙vtoclga)
from S1 S18 show (A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A by (rule MMI˙ax˙mp)
qed

lemma (in MMIsar0) MMI˙divmulz: assumes A1: A $\in \mathbb{C}$ and
A2: B $\in \mathbb{C}$ and
A3: C $\in \mathbb{C}$
shows B $\neq$ **0** $\longrightarrow$
((A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A)
proof -
have S1: B =
if (B $\neq$ **0** , B , **1**) $\longrightarrow$
(A / B) =
(A / if (B $\neq$ **0** , B , **1**)) by (rule MMI˙opreq2)
from S1 have S2: B =
if (B $\neq$ **0** , B , **1**) $\longrightarrow$
((A / B) =
C $\longleftrightarrow$ (A / if (B $\neq$ **0** , B , **1**)) = C) by (rule MMI˙eqeq1d)
have S3: B =
if (B $\neq$ **0** , B , **1**) $\longrightarrow$
(B $\cdot$ C) =
(if (B $\neq$ **0** , B , **1**) $\cdot$ C) by (rule MMI˙opreq1)
from S3 have S4: B =
if (B $\neq$ **0** , B , **1**) $\longrightarrow$
((B $\cdot$ C) =
A $\longleftrightarrow$ (if (B $\neq$ **0** , B , **1**) $\cdot$ C) = A) by (rule MMI˙eqeq1d)
from S2 S4 have S5: B =
if (B $\neq$ **0** , B , **1**) $\longrightarrow$
(((A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A) $\longleftrightarrow$
((A / if (B $\neq$ **0** , B , **1**)) =
C $\longleftrightarrow$
(if (B $\neq$ **0** , B , **1**) $\cdot$ C) = A)) by (rule MMI˙bibi12d)
from A1 have S6: A $\in \mathbb{C}$.
from A2 have S7: B $\in \mathbb{C}$.
have S8: **1** $\in \mathbb{C}$ by (rule MMI˙1cn)
from S7 S8 have S9: if (B $\neq$ **0** , B , **1**) $\in \mathbb{C}$ by (rule MMI˙keepel)

from A3 have S10: $C \in \mathbb{C}$.
have S11: if ($B \neq \mathbf{0}$, B , $\mathbf{1}$) $\neq \mathbf{0}$ by (rule MMI˙elimne0)
from S6 S9 S10 S11 have S12: (A / if ($B \neq \mathbf{0}$, B , $\mathbf{1}$)) =
C $\longleftrightarrow$ (if ($B \neq \mathbf{0}$, B , $\mathbf{1}$) $\cdot$ C) = A by (rule MMI˙divmul)
from S5 S12 show $B \neq \mathbf{0} \longrightarrow$
((A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A) by (rule MMI˙dedth)
qed

lemma (in MMIsar0) MMI˙divmult:
shows (($A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}$) $\wedge B \neq \mathbf{0}$) $\longrightarrow$
((A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A)
proof -
have S1: A =
if ($A \in \mathbb{C}$, A , $\mathbf{0}$) $\longrightarrow$
(A / B) =
(if ($A \in \mathbb{C}$, A , $\mathbf{0}$) / B) by (rule MMI˙opreq1)
from S1 have S2: A =
if ($A \in \mathbb{C}$, A , $\mathbf{0}$) $\longrightarrow$
((A / B) =
C $\longleftrightarrow$ (if ($A \in \mathbb{C}$, A , $\mathbf{0}$) / B) = C) by (rule MMI˙eqeq1d)
have S3: A =
if ($A \in \mathbb{C}$, A , $\mathbf{0}$) $\longrightarrow$
((B $\cdot$ C) =
A $\longleftrightarrow$ (B $\cdot$ C) = if ($A \in \mathbb{C}$, A , $\mathbf{0}$)) by (rule MMI˙eqeq2)
from S2 S3 have S4: A =
if ($A \in \mathbb{C}$, A , $\mathbf{0}$) $\longrightarrow$
(((A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A) $\longleftrightarrow$
((if ($A \in \mathbb{C}$, A , $\mathbf{0}$) / B) =
C $\longleftrightarrow$
(B $\cdot$ C) = if ($A \in \mathbb{C}$, A , $\mathbf{0}$))) by (rule MMI˙bibi12d)
from S4 have S5: A =
if ($A \in \mathbb{C}$, A , $\mathbf{0}$) $\longrightarrow$
(($B \neq \mathbf{0} \longrightarrow$ ((A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A)) $\longleftrightarrow$
($B \neq \mathbf{0} \longrightarrow$
((if ($A \in \mathbb{C}$, A , $\mathbf{0}$) / B) =
C $\longleftrightarrow$
(B $\cdot$ C) = if ($A \in \mathbb{C}$, A , $\mathbf{0}$)))) by (rule MMI˙imbi2d)
have S6: B =
if ($B \in \mathbb{C}$, B , $\mathbf{0}$) $\longrightarrow$
($B \neq \mathbf{0} \longleftrightarrow$ if ($B \in \mathbb{C}$, B , $\mathbf{0}$) $\neq \mathbf{0}$) by (rule MMI˙neeq1)
have S7: B =
if ($B \in \mathbb{C}$, B , $\mathbf{0}$) $\longrightarrow$
(if ($A \in \mathbb{C}$, A , $\mathbf{0}$) / B) =
(if ($A \in \mathbb{C}$, A , $\mathbf{0}$) / if ($B \in \mathbb{C}$, B , $\mathbf{0}$)) by (rule MMI˙opreq2)
from S7 have S8: B =
if ($B \in \mathbb{C}$, B , $\mathbf{0}$) $\longrightarrow$
((if ($A \in \mathbb{C}$, A , $\mathbf{0}$) / B) =
C $\longleftrightarrow$
(if ($A \in \mathbb{C}$, A , $\mathbf{0}$) / if ($B \in \mathbb{C}$, B , $\mathbf{0}$)) =

C) by (rule MMI˙eqeq1d)
have S9: B =
if (B ∈ ℂ , B , **0**) ⟶
(B · C) =
(if (B ∈ ℂ , B , **0**) · C) by (rule MMI˙opreq1)
from S9 have S10: B =
if (B ∈ ℂ , B , **0**) ⟶
((B · C) =
if (A ∈ ℂ , A , **0**) ⟷
(if (B ∈ ℂ , B , **0**) · C) =
if (A ∈ ℂ , A , **0**)) by (rule MMI˙eqeq1d)
from S8 S10 have S11: B =
if (B ∈ ℂ , B , **0**) ⟶
(((if (A ∈ ℂ , A , **0**) / B) = C ⟷ (B · C) = if (A ∈ ℂ , A , **0**)) ⟷
((if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
C ⟷
(if (B ∈ ℂ , B , **0**) · C) =
if (A ∈ ℂ , A , **0**))) by (rule MMI˙bibi12d)
from S6 S11 have S12: B =
if (B ∈ ℂ , B , **0**) ⟶
((B ≠ **0** ⟶ ((if (A ∈ ℂ , A , **0**) / B) = C ⟷ (B · C) = if (A ∈ ℂ , A ,
0))) ⟷
(if (B ∈ ℂ , B , **0**) ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
C ⟷
(if (B ∈ ℂ , B , **0**) · C) =
if (A ∈ ℂ , A , **0**)))) by (rule MMI˙imbi12d)
have S13: C =
if (C ∈ ℂ , C , **0**) ⟶
((if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
C ⟷
(if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
if (C ∈ ℂ , C , **0**)) by (rule MMI˙eqeq2)
have S14: C =
if (C ∈ ℂ , C , **0**) ⟶
(if (B ∈ ℂ , B , **0**) · C) =
(if (B ∈ ℂ , B , **0**) · if (C ∈ ℂ , C , **0**)) by (rule MMI˙opreq2)
from S14 have S15: C =
if (C ∈ ℂ , C , **0**) ⟶
((if (B ∈ ℂ , B , **0**) · C) =
if (A ∈ ℂ , A , **0**) ⟷
(if (B ∈ ℂ , B , **0**) · if (C ∈ ℂ , C , **0**)) =
if (A ∈ ℂ , A , **0**)) by (rule MMI˙eqeq1d)
from S13 S15 have S16: C =
if (C ∈ ℂ , C , **0**) ⟶
(((if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) = C ⟷ (if (B ∈ ℂ , B , **0**) · C
) = if (A ∈ ℂ , A , **0**)) ⟷
((if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
if (C ∈ ℂ , C , **0**) ⟷

(if (B ∈ ℂ , B , **0**) · if (C ∈ ℂ , C , **0**)) =
if (A ∈ ℂ , A , **0**))) by (rule MMI˙bibi12d)
from S16 have S17: C =
if (C ∈ ℂ , C , **0**) ⟶
((if (B ∈ ℂ , B , **0**) ≠ **0** ⟶ ((if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
C ⟷ (if (B ∈ ℂ , B , **0**) · C) = if (A ∈ ℂ , A , **0**))) ⟷
(if (B ∈ ℂ , B , **0**) ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
if (C ∈ ℂ , C , **0**) ⟷
(if (B ∈ ℂ , B , **0**) · if (C ∈ ℂ , C , **0**)) =
if (A ∈ ℂ , A , **0**)))) by (rule MMI˙imbi2d)
have S18: **0** ∈ ℂ by (rule MMI˙0cn)
from S18 have S19: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S20: **0** ∈ ℂ by (rule MMI˙0cn)
from S20 have S21: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
have S22: **0** ∈ ℂ by (rule MMI˙0cn)
from S22 have S23: if (C ∈ ℂ , C , **0**) ∈ ℂ by (rule MMI˙elimel)
from S19 S21 S23 have S24: if (B ∈ ℂ , B , **0**) ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
if (C ∈ ℂ , C , **0**) ⟷
(if (B ∈ ℂ , B , **0**) · if (C ∈ ℂ , C , **0**)) =
if (A ∈ ℂ , A , **0**)) by (rule MMI˙divmulz)
from S5 S12 S17 S24 have S25: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B ≠ **0** ⟶
((A / B) = C ⟷ (B · C) = A)) by (rule MMI˙dedth3h)
from S25 show ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A / B) = C ⟷ (B · C) = A) by (rule MMI˙imp)
qed

lemma (in MMIsar0) MMI˙divmul2t:
shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A / B) = C ⟷ A = (B · C))
proof -
have S1: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A / B) = C ⟷ (B · C) = A) by (rule MMI˙divmult)
have S2: (B · C) = A ⟷ A = (B · C) by (rule MMI˙eqcom)
from S1 S2 show ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A / B) = C ⟷ A = (B · C)) by (rule MMI˙syl6bb)
qed

lemma (in MMIsar0) MMI˙divmul3t:
shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A / B) = C ⟷ A = (C · B))
proof -
have S1: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A / B) = C ⟷ A = (B · C)) by (rule MMI˙divmul2t)
have S2: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B · C) = (C · B) by (rule MMI˙axmulcom)
from S2 have S3: (B ∈ ℂ ∧ C ∈ ℂ) ⟶

(A = (B · C) ⟷ A = (C · B)) by (rule MMI˙eqeq2d)
from S3 have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A = (B · C) ⟷ A = (C · B)) by (rule MMI˙3adant1)
from S4 have S5: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
(A = (B · C) ⟷ A = (C · B)) by (rule MMI˙adantr)
from S1 S5 show ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A / B) = C ⟷ A = (C · B)) by (rule MMI˙bitrd)
qed

lemma (in MMIsar0) MMI˙divcl: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: B ≠ **0**
shows (A / B) ∈ ℂ
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: B ≠ **0**.
from S1 S2 S3 have S4: (A / B) =
⋃ − x ∈ ℂ . (B · x) = A ″ by (rule MMI˙divval)
from A2 have S5: B ∈ ℂ.
from A1 have S6: A ∈ ℂ.
from A3 have S7: B ≠ **0**.
from S5 S6 S7 have S8: ∃! x . x ∈ ℂ ∧ (B · x) = A by (rule MMI˙receu)
have S9: (∃! x . x ∈ ℂ ∧ (B · x) =
A) ⟶ ⋃ − x ∈ ℂ . (B · x) = A ″ ∈ ℂ by (rule MMI˙reucl)
from S8 S9 have S10: ⋃ − x ∈ ℂ . (B · x) = A ″ ∈ ℂ by (rule MMI˙ax˙mp)
from S4 S10 show (A / B) ∈ ℂ by (rule MMI˙eqeltr)
qed

lemma (in MMIsar0) MMI˙divclz: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows B ≠ **0** ⟶ (A / B) ∈ ℂ
proof -
have S1: B =
if (B ≠ **0** , B , **1**) ⟶
(A / B) =
(A / if (B ≠ **0** , B , **1**)) by (rule MMI˙opreq2)
from S1 have S2: B =
if (B ≠ **0** , B , **1**) ⟶
((A / B) ∈ ℂ ⟷
(A / if (B ≠ **0** , B , **1**)) ∈ ℂ) by (rule MMI˙eleq1d)
from A1 have S3: A ∈ ℂ.
from A2 have S4: B ∈ ℂ.
have S5: **1** ∈ ℂ by (rule MMI˙1cn)
from S4 S5 have S6: if (B ≠ **0** , B , **1**) ∈ ℂ by (rule MMI˙keepel)
have S7: if (B ≠ **0** , B , **1**) ≠ **0** by (rule MMI˙elimne0)
from S3 S6 S7 have S8: (A / if (B ≠ **0** , B , **1**)) ∈ ℂ by (rule MMI˙divcl)
from S2 S8 show B ≠ **0** ⟶ (A / B) ∈ ℂ by (rule MMI˙dedth)
qed

lemma (in MMIsar0) MMI˙divclt:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A / B) ∈ ℂ
proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
(A / B) =
(if (A ∈ ℂ , A , **0**) / B) by (rule MMI˙opreq1)
from S1 have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
((A / B) ∈ ℂ ⟷
(if (A ∈ ℂ , A , **0**) / B) ∈ ℂ) by (rule MMI˙eleq1d)
from S2 have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
((B ≠ **0** ⟶ (A / B) ∈ ℂ) ⟷
(B ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) / B) ∈ ℂ)) by (rule MMI˙imbi2d)
have S4: B =
if (B ∈ ℂ , B , **0**) ⟶
(B ≠ **0** ⟷ if (B ∈ ℂ , B , **0**) ≠ **0**) by (rule MMI˙neeq1)
have S5: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) / B) =
(if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
from S5 have S6: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) / B) ∈ ℂ ⟷
(if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) ∈ ℂ) by (rule MMI˙eleq1d)
from S4 S6 have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
((B ≠ **0** ⟶ (if (A ∈ ℂ , A , **0**) / B) ∈ ℂ) ⟷
(if (B ∈ ℂ , B , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) ∈ ℂ)) by (rule MMI˙imbi12d)
have S8: **0** ∈ ℂ by (rule MMI˙0cn)
from S8 have S9: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S10: **0** ∈ ℂ by (rule MMI˙0cn)
from S10 have S11: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
from S9 S11 have S12: if (B ∈ ℂ , B , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) ∈ ℂ by (rule MMI˙divclz)
from S3 S7 S12 have S13: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(B ≠ **0** ⟶ (A / B) ∈ ℂ) by (rule MMI˙dedth2h)
from S13 show (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A / B) ∈ ℂ by (rule MMI˙3impia)
qed

lemma (in MMIsar0) MMI˙reccl: assumes A1: A ∈ ℂ and
A2: A ≠ **0**
shows (**1** / A) ∈ ℂ
proof -
have S1: **1** ∈ ℂ by (rule MMI˙1cn)
from A1 have S2: A ∈ ℂ.
from A2 have S3: A ≠ **0**.
from S1 S2 S3 show (**1** / A) ∈ ℂ by (rule MMI˙divcl)
qed

lemma (in MMIsar0) MMI˙recclz: assumes A1: A ∈ ℂ
shows A ≠ **0** ⟶ (**1** / A) ∈ ℂ
proof -
have S1: **1** ∈ ℂ by (rule MMI˙1cn)
from A1 have S2: A ∈ ℂ.
from S1 S2 show A ≠ **0** ⟶ (**1** / A) ∈ ℂ by (rule MMI˙divclz)
qed

lemma (in MMIsar0) MMI˙recclt:
shows (A ∈ ℂ ∧ A ≠ **0**) ⟶ (**1** / A) ∈ ℂ
proof -
have S1: **1** ∈ ℂ by (rule MMI˙1cn)
have S2: (**1** ∈ ℂ ∧ A ∈ ℂ ∧ A ≠ **0**) ⟶
(**1** / A) ∈ ℂ by (rule MMI˙divclt)
from S1 S2 show (A ∈ ℂ ∧ A ≠ **0**) ⟶ (**1** / A) ∈ ℂ by (rule MMI˙mp3an1)
qed

lemma (in MMIsar0) MMI˙divcan2: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: A ≠ **0**
shows (A · (B / A)) = B
proof -
have S1: (B / A) = (B / A) by (rule MMI˙eqid)
from A2 have S2: B ∈ ℂ.
from A1 have S3: A ∈ ℂ.
from A2 have S4: B ∈ ℂ.
from A1 have S5: A ∈ ℂ.
from A3 have S6: A ≠ **0**.
from S4 S5 S6 have S7: (B / A) ∈ ℂ by (rule MMI˙divcl)
from A3 have S8: A ≠ **0**.
from S2 S3 S7 S8 have S9: (B / A) =
(B / A) ⟷ (A · (B / A)) = B by (rule MMI˙divmul)
from S1 S9 show (A · (B / A)) = B by (rule MMI˙mpbi)
qed

lemma (in MMIsar0) MMI˙divcan1: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: A ≠ **0**
shows ((B / A) · A) = B

proof -
from A2 have S1: B ∈ ℂ.
from A1 have S2: A ∈ ℂ.
from A3 have S3: A ≠ **0**.
from S1 S2 S3 have S4: (B / A) ∈ ℂ by (rule MMI˙divcl)
from A1 have S5: A ∈ ℂ.
from S4 S5 have S6: ((B / A) · A) = (A · (B / A)) by (rule MMI˙mulcom)
from A1 have S7: A ∈ ℂ.
from A2 have S8: B ∈ ℂ.
from A3 have S9: A ≠ **0**.
from S7 S8 S9 have S10: (A · (B / A)) = B by (rule MMI˙divcan2)
from S6 S10 show ((B / A) · A) = B by (rule MMI˙eqtr)
qed

lemma (in MMIsar0) MMI˙divcan1z: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows A ≠ **0** ⟶ ((B / A) · A) = B
proof -
have S1: A =
if (A ≠ **0** , A , **1**) ⟶
(B / A) =
(B / if (A ≠ **0** , A , **1**)) by (rule MMI˙opreq2)
have S2: A =
if (A ≠ **0** , A , **1**) ⟶
A = if (A ≠ **0** , A , **1**) by (rule MMI˙id)
from S1 S2 have S3: A =
if (A ≠ **0** , A , **1**) ⟶
((B / A) · A) =
((B / if (A ≠ **0** , A , **1**)) · if (A ≠ **0** , A , **1**)) by (rule MMI˙opreq12d)
from S3 have S4: A =
if (A ≠ **0** , A , **1**) ⟶
(((B / A) · A) =
B ⟷
((B / if (A ≠ **0** , A , **1**)) · if (A ≠ **0** , A , **1**)) =
B) by (rule MMI˙eqeq1d)
from A1 have S5: A ∈ ℂ.
have S6: **1** ∈ ℂ by (rule MMI˙1cn)
from S5 S6 have S7: if (A ≠ **0** , A , **1**) ∈ ℂ by (rule MMI˙keepel)
from A2 have S8: B ∈ ℂ.
have S9: if (A ≠ **0** , A , **1**) ≠ **0** by (rule MMI˙elimne0)
from S7 S8 S9 have S10: ((B / if (A ≠ **0** , A , **1**)) · if (A ≠ **0** , A , **1**)) =
B by (rule MMI˙divcan1)
from S4 S10 show A ≠ **0** ⟶ ((B / A) · A) = B by (rule MMI˙dedth)
qed

lemma (in MMIsar0) MMI˙divcan2z: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows A ≠ **0** ⟶ (A · (B / A)) = B
proof -

have S1: A =
if (A ≠ **0** , A , **1**) ⟶
A = if (A ≠ **0** , A , **1**) by (rule MMI˙id)
have S2: A =
if (A ≠ **0** , A , **1**) ⟶
(B / A) =
(B / if (A ≠ **0** , A , **1**)) by (rule MMI˙opreq2)
from S1 S2 have S3: A =
if (A ≠ **0** , A , **1**) ⟶
(A · (B / A)) =
(if (A ≠ **0** , A , **1**) · (B / if (A ≠ **0** , A , **1**))) by (rule MMI˙opreq12d)
from S3 have S4: A =
if (A ≠ **0** , A , **1**) ⟶
((A · (B / A)) =
B ⟷
(if (A ≠ **0** , A , **1**) · (B / if (A ≠ **0** , A , **1**))) =
B) by (rule MMI˙eqeq1d)
from A1 have S5: A ∈ ℂ.
have S6: **1** ∈ ℂ by (rule MMI˙1cn)
from S5 S6 have S7: if (A ≠ **0** , A , **1**) ∈ ℂ by (rule MMI˙keepel)
from A2 have S8: B ∈ ℂ.
have S9: if (A ≠ **0** , A , **1**) ≠ **0** by (rule MMI˙elimne0)
from S7 S8 S9 have S10: (if (A ≠ **0** , A , **1**) · (B / if (A ≠ **0** , A , **1**))) =
B by (rule MMI˙divcan2)
from S4 S10 show A ≠ **0** ⟶ (A · (B / A)) = B by (rule MMI˙dedth)
qed

lemma (in MMIsar0) MMI˙divcan1t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶
((B / A) · A) = B
proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
(A ≠ **0** ⟷ if (A ∈ ℂ , A , **0**) ≠ **0**) by (rule MMI˙neeq1)
have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
(B / A) =
(B / if (A ∈ ℂ , A , **0**)) by (rule MMI˙opreq2)
have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
A = if (A ∈ ℂ , A , **0**) by (rule MMI˙id)
from S2 S3 have S4: A =
if (A ∈ ℂ , A , **0**) ⟶
((B / A) · A) =
((B / if (A ∈ ℂ , A , **0**)) · if (A ∈ ℂ , A , **0**)) by (rule MMI˙opreq12d)
from S4 have S5: A =
if (A ∈ ℂ , A , **0**) ⟶
(((B / A) · A) =
B ⟷

((B / if (A ∈ ℂ , A , **0**)) · if (A ∈ ℂ , A , **0**)) =
B) by (rule MMI˙eqeq1d)
from S1 S5 have S6: A =
if (A ∈ ℂ , A , **0**) ⟶
((A ≠ **0** ⟶ ((B / A) · A) = B) ⟷
(if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
((B / if (A ∈ ℂ , A , **0**)) · if (A ∈ ℂ , A , **0**)) =
B)) by (rule MMI˙imbi12d)
have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(B / if (A ∈ ℂ , A , **0**)) =
(if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**)) by (rule MMI˙opreq1)
from S7 have S8: B =
if (B ∈ ℂ , B , **0**) ⟶
((B / if (A ∈ ℂ , A , **0**)) · if (A ∈ ℂ , A , **0**)) =
((if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**)) · if (A ∈ ℂ , A , **0**)) by (rule MMI˙opreq1d)
have S9: B =
if (B ∈ ℂ , B , **0**) ⟶
B = if (B ∈ ℂ , B , **0**) by (rule MMI˙id)
from S8 S9 have S10: B =
if (B ∈ ℂ , B , **0**) ⟶
(((B / if (A ∈ ℂ , A , **0**)) · if (A ∈ ℂ , A , **0**)) =
B ⟷
((if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**)) · if (A ∈ ℂ , A , **0**)) =
if (B ∈ ℂ , B , **0**)) by (rule MMI˙eqeq12d)
from S10 have S11: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) ≠ **0** ⟶ ((B / if (A ∈ ℂ , A , **0**)) · if (A ∈ ℂ , A , **0**)) = B) ⟷
(if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
((if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**)) · if (A ∈ ℂ , A , **0**)) =
if (B ∈ ℂ , B , **0**))) by (rule MMI˙imbi2d)
have S12: **0** ∈ ℂ by (rule MMI˙0cn)
from S12 have S13: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S14: **0** ∈ ℂ by (rule MMI˙0cn)
from S14 have S15: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
from S13 S15 have S16: if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
((if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**)) · if (A ∈ ℂ , A , **0**)) =
if (B ∈ ℂ , B , **0**) by (rule MMI˙divcan1z)
from S6 S11 S16 have S17: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A ≠ **0** ⟶ ((B / A) · A) = B) by (rule MMI˙dedth2h)
from S17 show (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶
((B / A) · A) = B by (rule MMI˙3impia)
qed

lemma (in MMIsar0) MMI˙divcan2t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶
(A · (B / A)) = B

proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
(A ≠ **0** ⟷ if (A ∈ ℂ , A , **0**) ≠ **0**) by (rule MMI˙neeq1)
have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
A = if (A ∈ ℂ , A , **0**) by (rule MMI˙id)
have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
(B / A) =
(B / if (A ∈ ℂ , A , **0**)) by (rule MMI˙opreq2)
from S2 S3 have S4: A =
if (A ∈ ℂ , A , **0**) ⟶
(A · (B / A)) =
(if (A ∈ ℂ , A , **0**) · (B / if (A ∈ ℂ , A , **0**))) by (rule MMI˙opreq12d)
from S4 have S5: A =
if (A ∈ ℂ , A , **0**) ⟶
((A · (B / A)) =
B ⟷
(if (A ∈ ℂ , A , **0**) · (B / if (A ∈ ℂ , A , **0**))) =
B) by (rule MMI˙eqeq1d)
from S1 S5 have S6: A =
if (A ∈ ℂ , A , **0**) ⟶
((A ≠ **0** ⟶ (A · (B / A)) = B) ⟷
(if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) · (B / if (A ∈ ℂ , A , **0**))) =
B)) by (rule MMI˙imbi12d)
have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(B / if (A ∈ ℂ , A , **0**)) =
(if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**)) by (rule MMI˙opreq1)
from S7 have S8: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) · (B / if (A ∈ ℂ , A , **0**))) =
(if (A ∈ ℂ , A , **0**) · (if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**))) by (rule MMI˙opreq2d)
have S9: B =
if (B ∈ ℂ , B , **0**) ⟶
B = if (B ∈ ℂ , B , **0**) by (rule MMI˙id)
from S8 S9 have S10: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) · (B / if (A ∈ ℂ , A , **0**))) =
B ⟷
(if (A ∈ ℂ , A , **0**) · (if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**))) =
if (B ∈ ℂ , B , **0**)) by (rule MMI˙eqeq12d)
from S10 have S11: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) ≠ **0** ⟶ (if (A ∈ ℂ , A , **0**) · (B / if (A ∈ ℂ , A , **0**))) = B) ⟷

(if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) · (if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**))) =
if (B ∈ ℂ , B , **0**))) by (rule MMI˙imbi2d)
have S12: **0** ∈ ℂ by (rule MMI˙0cn)
from S12 have S13: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S14: **0** ∈ ℂ by (rule MMI˙0cn)
from S14 have S15: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
from S13 S15 have S16: if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) · (if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**))) =
if (B ∈ ℂ , B , **0**) by (rule MMI˙divcan2z)
from S6 S11 S16 have S17: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A ≠ **0** ⟶ (A · (B / A)) = B) by (rule MMI˙dedth2h)
from S17 show (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶
(A · (B / A)) = B by (rule MMI˙3impia)
qed

lemma (in MMIsar0) MMI˙divne0bt:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A ≠ **0** ⟷ (A / B) ≠ **0**)
proof -
have S1: B ∈ ℂ ⟶ (B · **0**) = **0** by (rule MMI˙mul01t)
from S1 have S2: B ∈ ℂ ⟶ ((B · **0**) = A ⟷ **0** = A) by (rule MMI˙eqeq1d)
have S3: A = **0** ⟷ **0** = A by (rule MMI˙eqcom)
from S2 S3 have S4: B ∈ ℂ ⟶ (A = **0** ⟷ (B · **0**) = A) by (rule MMI˙syl6rbbrA)
from S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A = **0** ⟷ (B · **0**) = A) by (rule MMI˙3ad2ant2)
have S6: **0** ∈ ℂ by (rule MMI˙0cn)
have S7: ((A ∈ ℂ ∧ B ∈ ℂ ∧ **0** ∈ ℂ) ∧ B ≠ **0**) ⟶
((A / B) = **0** ⟷ (B · **0**) = A) by (rule MMI˙divmult)
from S6 S7 have S8: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ B ≠ **0**) ⟶
((A / B) = **0** ⟷ (B · **0**) = A) by (rule MMI˙mp3anl3)
from S8 have S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
((A / B) = **0** ⟷ (B · **0**) = A) by (rule MMI˙3impa)
from S5 S9 have S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A = **0** ⟷ (A / B) = **0**) by (rule MMI˙bitr4d)
from S10 show (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A ≠ **0** ⟷ (A / B) ≠ **0**) by (rule MMI˙eqneqd)
qed

lemma (in MMIsar0) MMI˙divne0: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: A ≠ **0** and
A4: B ≠ **0**
shows (A / B) ≠ **0**
proof -
from A1 have S1: A ∈ ℂ.

from A2 have S2: B ∈ ℂ.
from A4 have S3: B ≠ **0**.
from A3 have S4: A ≠ **0**.
have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A ≠ **0** ⟷ (A / B) ≠ **0**) by (rule MMI˙divne0bt)
from S4 S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A / B) ≠ **0** by (rule MMI˙mpbii)
from S1 S2 S3 S6 show (A / B) ≠ **0** by (rule MMI˙mp3an)
qed

lemma (in MMIsar0) MMI˙recne0z: assumes A1: A ∈ ℂ
shows A ≠ **0** ⟶ (**1** / A) ≠ **0**
proof -
have S1: A =
if (A ≠ **0** , A , **1**) ⟶
(**1** / A) =
(**1** / if (A ≠ **0** , A , **1**)) by (rule MMI˙opreq2)
from S1 have S2: A =
if (A ≠ **0** , A , **1**) ⟶
((**1** / A) ≠ **0** ⟷
(**1** / if (A ≠ **0** , A , **1**)) ≠ **0**) by (rule MMI˙neeq1d)
have S3: **1** ∈ ℂ by (rule MMI˙1cn)
from A1 have S4: A ∈ ℂ.
have S5: **1** ∈ ℂ by (rule MMI˙1cn)
from S4 S5 have S6: if (A ≠ **0** , A , **1**) ∈ ℂ by (rule MMI˙keepel)
have S7: **1** ≠ **0** by (rule MMI˙ax1ne0)
have S8: if (A ≠ **0** , A , **1**) ≠ **0** by (rule MMI˙elimne0)
from S3 S6 S7 S8 have S9: (**1** / if (A ≠ **0** , A , **1**)) ≠ **0** by (rule MMI˙divne0)
from S2 S9 show A ≠ **0** ⟶ (**1** / A) ≠ **0** by (rule MMI˙dedth)
qed

lemma (in MMIsar0) MMI˙recne0t:
shows (A ∈ ℂ ∧ A ≠ **0**) ⟶ (**1** / A) ≠ **0**
proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
(A ≠ **0** ⟷ if (A ∈ ℂ , A , **0**) ≠ **0**) by (rule MMI˙neeq1)
have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
(**1** / A) =
(**1** / if (A ∈ ℂ , A , **0**)) by (rule MMI˙opreq2)
from S2 have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
((**1** / A) ≠ **0** ⟷
(**1** / if (A ∈ ℂ , A , **0**)) ≠ **0**) by (rule MMI˙neeq1d)
from S1 S3 have S4: A =
if (A ∈ ℂ , A , **0**) ⟶
((A ≠ **0** ⟶ (**1** / A) ≠ **0**) ⟷
(if (A ∈ ℂ , A , **0**) ≠ **0** ⟶

(**1** / if (A ∈ ℂ , A , **0**)) ≠ **0**)) by (rule MMI˙imbi12d)
have S5: **0** ∈ ℂ by (rule MMI˙0cn)
from S5 have S6: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
from S6 have S7: if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
(**1** / if (A ∈ ℂ , A , **0**)) ≠ **0** by (rule MMI˙recne0z)
from S4 S7 have S8: A ∈ ℂ ⟶ (A ≠ **0** ⟶ (**1** / A) ≠ **0**) by (rule MMI˙dedth)
from S8 show (A ∈ ℂ ∧ A ≠ **0**) ⟶ (**1** / A) ≠ **0** by (rule MMI˙imp)
qed

lemma (in MMIsar0) MMI˙recid: assumes A1: A ∈ ℂ and
A2: A ≠ **0**
shows (A · (**1** / A)) = **1**
proof -
from A1 have S1: A ∈ ℂ.
have S2: **1** ∈ ℂ by (rule MMI˙1cn)
from A2 have S3: A ≠ **0**.
from S1 S2 S3 show (A · (**1** / A)) = **1** by (rule MMI˙divcan2)
qed

lemma (in MMIsar0) MMI˙recidz: assumes A1: A ∈ ℂ
shows A ≠ **0** ⟶ (A · (**1** / A)) = **1**
proof -
from A1 have S1: A ∈ ℂ.
have S2: **1** ∈ ℂ by (rule MMI˙1cn)
from S1 S2 show A ≠ **0** ⟶ (A · (**1** / A)) = **1** by (rule MMI˙divcan2z)
qed

lemma (in MMIsar0) MMI˙recidt:
shows (A ∈ ℂ ∧ A ≠ **0**) ⟶
(A · (**1** / A)) = **1**
proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
(A ≠ **0** ⟷ if (A ∈ ℂ , A , **0**) ≠ **0**) by (rule MMI˙neeq1)
have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
A = if (A ∈ ℂ , A , **0**) by (rule MMI˙id)
have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
(**1** / A) =
(**1** / if (A ∈ ℂ , A , **0**)) by (rule MMI˙opreq2)
from S2 S3 have S4: A =
if (A ∈ ℂ , A , **0**) ⟶
(A · (**1** / A)) =
(if (A ∈ ℂ , A , **0**) · (**1** / if (A ∈ ℂ , A , **0**))) by (rule MMI˙opreq12d)
from S4 have S5: A =
if (A ∈ ℂ , A , **0**) ⟶
((A · (**1** / A)) =
1 ⟷

(if (A ∈ ℂ , A , **0**) · (**1** / if (A ∈ ℂ , A , **0**))) =
1) by (rule MMI˙eqeq1d)
from S1 S5 have S6: A =
if (A ∈ ℂ , A , **0**) ⟶
((A ≠ **0** ⟶ (A · (**1** / A)) = **1**) ⟷
(if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) · (**1** / if (A ∈ ℂ , A , **0**))) =
1)) by (rule MMI˙imbi12d)
have S7: **0** ∈ ℂ by (rule MMI˙0cn)
from S7 have S8: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
from S8 have S9: if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) · (**1** / if (A ∈ ℂ , A , **0**))) =
1 by (rule MMI˙recidz)
from S6 S9 have S10: A ∈ ℂ ⟶
(A ≠ **0** ⟶ (A · (**1** / A)) = **1**) by (rule MMI˙dedth)
from S10 show (A ∈ ℂ ∧ A ≠ **0**) ⟶
(A · (**1** / A)) = **1** by (rule MMI˙imp)
qed

lemma (in MMIsar0) MMI˙recid2t:
shows (A ∈ ℂ ∧ A ≠ **0**) ⟶
((**1** / A) · A) = **1**
proof -
have S1: ((**1** / A) ∈ ℂ ∧ A ∈ ℂ) ⟶
((**1** / A) · A) = (A · (**1** / A)) by (rule MMI˙axmulcom)
have S2: (A ∈ ℂ ∧ A ≠ **0**) ⟶ (**1** / A) ∈ ℂ by (rule MMI˙recclt)
have S3: (A ∈ ℂ ∧ A ≠ **0**) ⟶ A ∈ ℂ by (rule MMI˙pm3˙26)
from S1 S2 S3 have S4: (A ∈ ℂ ∧ A ≠ **0**) ⟶
((**1** / A) · A) = (A · (**1** / A)) by (rule MMI˙sylanc)
have S5: (A ∈ ℂ ∧ A ≠ **0**) ⟶
(A · (**1** / A)) = **1** by (rule MMI˙recidt)
from S4 S5 show (A ∈ ℂ ∧ A ≠ **0**) ⟶
((**1** / A) · A) = **1** by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙divrec: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: B ≠ **0**
shows (A / B) = (A · (**1** / B))
proof -
from A2 have S1: B ∈ ℂ.
from A1 have S2: A ∈ ℂ.
from A2 have S3: B ∈ ℂ.
from A3 have S4: B ≠ **0**.
from S3 S4 have S5: (**1** / B) ∈ ℂ by (rule MMI˙reccl)
from S2 S5 have S6: (A · (**1** / B)) ∈ ℂ by (rule MMI˙mulcl)
from S1 S6 have S7: (B · (A · (**1** / B))) =
((A · (**1** / B)) · B) by (rule MMI˙mulcom)
from A1 have S8: A ∈ ℂ.

from S5 have S9: (**1** / B) ∈ ℂ .
from A2 have S10: B ∈ ℂ.
from S8 S9 S10 have S11: ((A · (**1** / B)) · B) =
(A · ((**1** / B) · B)) by (rule MMI˙mulass)
from A2 have S12: B ∈ ℂ.
have S13: **1** ∈ ℂ by (rule MMI˙1cn)
from A3 have S14: B ≠ **0**.
from S12 S13 S14 have S15: ((**1** / B) · B) = **1** by (rule MMI˙divcan1)
from S15 have S16: (A · ((**1** / B) · B)) = (A · **1**) by (rule MMI˙opreq2i)
from A1 have S17: A ∈ ℂ.
from S17 have S18: (A · **1**) = A by (rule MMI˙mulid1)
from S16 S18 have S19: (A · ((**1** / B) · B)) = A by (rule MMI˙eqtr)
from S7 S11 S19 have S20: (B · (A · (**1** / B))) = A by (rule MMI˙3eqtr)
from A1 have S21: A ∈ ℂ.
from A2 have S22: B ∈ ℂ.
from S6 have S23: (A · (**1** / B)) ∈ ℂ .
from A3 have S24: B ≠ **0**.
from S21 S22 S23 S24 have S25: (A / B) =
(A · (**1** / B)) ⟷
(B · (A · (**1** / B))) = A by (rule MMI˙divmul)
from S20 S25 show (A / B) = (A · (**1** / B)) by (rule MMI˙mpbir)
qed

lemma (in MMIsar0) MMI˙divrecz: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows B ≠ **0** ⟶ (A / B) = (A · (**1** / B))
proof -
have S1: B =
if (B ≠ **0** , B , **1**) ⟶
(A / B) =
(A / if (B ≠ **0** , B , **1**)) by (rule MMI˙opreq2)
have S2: B =
if (B ≠ **0** , B , **1**) ⟶
(**1** / B) =
(**1** / if (B ≠ **0** , B , **1**)) by (rule MMI˙opreq2)
from S2 have S3: B =
if (B ≠ **0** , B , **1**) ⟶
(A · (**1** / B)) =
(A · (**1** / if (B ≠ **0** , B , **1**))) by (rule MMI˙opreq2d)
from S1 S3 have S4: B =
if (B ≠ **0** , B , **1**) ⟶
((A / B) =
(A · (**1** / B)) ⟷
(A / if (B ≠ **0** , B , **1**)) =
(A · (**1** / if (B ≠ **0** , B , **1**)))) by (rule MMI˙eqeq12d)
from A1 have S5: A ∈ ℂ.
from A2 have S6: B ∈ ℂ.
have S7: **1** ∈ ℂ by (rule MMI˙1cn)
from S6 S7 have S8: if (B ≠ **0** , B , **1**) ∈ ℂ by (rule MMI˙keepel)

have S9: if (B ≠ **0** , B , **1**) ≠ **0** by (rule MMI˙elimne0)
from S5 S8 S9 have S10: (A / if (B ≠ **0** , B , **1**)) =
(A · (**1** / if (B ≠ **0** , B , **1**))) by (rule MMI˙divrec)
from S4 S10 show B ≠ **0** ⟶ (A / B) = (A · (**1** / B))
by (rule MMI˙dedth)
qed

lemma (in MMIsar0) MMI˙divrect:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A / B) = (A · (**1** / B))
proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
(A / B) =
(if (A ∈ ℂ , A , **0**) / B) by (rule MMI˙opreq1)
have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
(A · (**1** / B)) =
(if (A ∈ ℂ , A , **0**) · (**1** / B)) by (rule MMI˙opreq1)
from S1 S2 have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
((A / B) =
(A · (**1** / B)) ⟷
(if (A ∈ ℂ , A , **0**) / B) =
(if (A ∈ ℂ , A , **0**) · (**1** / B))) by (rule MMI˙eqeq12d)
from S3 have S4: A =
if (A ∈ ℂ , A , **0**) ⟶
((B ≠ **0** ⟶ (A / B) = (A · (**1** / B))) ⟷
(B ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) / B) =
(if (A ∈ ℂ , A , **0**) · (**1** / B)))) by (rule MMI˙imbi2d)
have S5: B =
if (B ∈ ℂ , B , **0**) ⟶
(B ≠ **0** ⟷ if (B ∈ ℂ , B , **0**) ≠ **0**) by (rule MMI˙neeq1)
have S6: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) / B) =
(if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(**1** / B) =
(**1** / if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
from S7 have S8: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) · (**1** / B)) =
(if (A ∈ ℂ , A , **0**) · (**1** / if (B ∈ ℂ , B , **0**))) by (rule MMI˙opreq2d)
from S6 S8 have S9: B =

if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) / B) =
(if (A ∈ ℂ , A , **0**) · (**1** / B)) ⟷
(if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
(if (A ∈ ℂ , A , **0**) · (**1** / if (B ∈ ℂ , B , **0**)))) by (rule MMI˙eqeq12d)
from S5 S9 have S10: B =
if (B ∈ ℂ , B , **0**) ⟶
((B ≠ **0** ⟶ (if (A ∈ ℂ , A , **0**) / B) = (if (A ∈ ℂ , A , **0**) · (**1** / B)))
⟷
(if (B ∈ ℂ , B , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
(if (A ∈ ℂ , A , **0**) · (**1** / if (B ∈ ℂ , B , **0**))))) by (rule MMI˙imbi12d)
have S11: **0** ∈ ℂ by (rule MMI˙0cn)
from S11 have S12: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S13: **0** ∈ ℂ by (rule MMI˙0cn)
from S13 have S14: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
from S12 S14 have S15: if (B ∈ ℂ , B , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
(if (A ∈ ℂ , A , **0**) · (**1** / if (B ∈ ℂ , B , **0**))) by (rule MMI˙divrecz)
from S4 S10 S15 have S16: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(B ≠ **0** ⟶
(A / B) = (A · (**1** / B))) by (rule MMI˙dedth2h)
from S16 show (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A / B) = (A · (**1** / B)) by (rule MMI˙3impia)
qed

lemma (in MMIsar0) MMI˙divrec2t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A / B) = ((**1** / B) · A)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A / B) = (A · (**1** / B)) by (rule MMI˙divrect)
have S2: (A ∈ ℂ ∧ (**1** / B) ∈ ℂ) ⟶
(A · (**1** / B)) = ((**1** / B) · A) by (rule MMI˙axmulcom)
have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶ A ∈ ℂ by (rule MMI˙3simp1)
have S4: (B ∈ ℂ ∧ B ≠ **0**) ⟶ (**1** / B) ∈ ℂ by (rule MMI˙recclt)
from S4 have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(**1** / B) ∈ ℂ by (rule MMI˙3adant1)
from S2 S3 S5 have S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A · (**1** / B)) = ((**1** / B) · A) by (rule MMI˙sylanc)
from S1 S6 show (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A / B) = ((**1** / B) · A) by (rule MMI˙eqtrd)
qed

lemma (in MMIsar0) MMI˙divasst:
shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A · B) / C) = (A · (B / C))
proof -
have S1: A ∈ ℂ ⟶ A ∈ ℂ by (rule MMI˙id)

have S2: $B \in \mathbb{C} \longrightarrow B \in \mathbb{C}$ by (rule MMI˙id)
have S3: $(C \in \mathbb{C} \wedge C \neq \mathbf{0}) \longrightarrow (\mathbf{1} / C) \in \mathbb{C}$ by (rule MMI˙recclt)
from S1 S2 S3 have S4: $(A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge (C \in \mathbb{C} \wedge C \neq \mathbf{0})) \longrightarrow$
$(A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge (\mathbf{1} / C) \in \mathbb{C})$ by (rule MMI˙3anim123i)
from S4 have S5: $A \in \mathbb{C} \longrightarrow$
$(B \in \mathbb{C} \longrightarrow$
$((C \in \mathbb{C} \wedge C \neq \mathbf{0}) \longrightarrow$
$(A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge (\mathbf{1} / C) \in \mathbb{C})))$ by (rule MMI˙3exp)
from S5 have S6: $A \in \mathbb{C} \longrightarrow$
$(B \in \mathbb{C} \longrightarrow$
$(C \in \mathbb{C} \longrightarrow$
$(C \neq \mathbf{0} \longrightarrow$
$(A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge (\mathbf{1} / C) \in \mathbb{C}))))$ by (rule MMI˙exp4a)
from S6 have S7: $((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge C \neq \mathbf{0}) \longrightarrow$
$(A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge (\mathbf{1} / C) \in \mathbb{C})$ by (rule MMI˙3imp1)
have S8: $(A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge (\mathbf{1} / C) \in \mathbb{C}) \longrightarrow$
$((A \cdot B) \cdot (\mathbf{1} / C)) =$
$(A \cdot (B \cdot (\mathbf{1} / C)))$ by (rule MMI˙axmulass)
from S7 S8 have S9: $((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge C \neq \mathbf{0}) \longrightarrow$
$((A \cdot B) \cdot (\mathbf{1} / C)) =$
$(A \cdot (B \cdot (\mathbf{1} / C)))$ by (rule MMI˙syl)
have S10: $((A \cdot B) \in \mathbb{C} \wedge C \in \mathbb{C} \wedge C \neq \mathbf{0}) \longrightarrow$
$((A \cdot B) / C) =$
$((A \cdot B) \cdot (\mathbf{1} / C))$ by (rule MMI˙divrect)
from S10 have S11: $(((A \cdot B) \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge C \neq \mathbf{0}) \longrightarrow$
$((A \cdot B) / C) =$
$((A \cdot B) \cdot (\mathbf{1} / C))$ by (rule MMI˙3expa)
have S12: $(A \in \mathbb{C} \wedge B \in \mathbb{C}) \longrightarrow (A \cdot B) \in \mathbb{C}$ by (rule MMI˙axmulcl)
from S12 have S13: $((A \in \mathbb{C} \wedge B \in \mathbb{C}) \wedge C \in \mathbb{C}) \longrightarrow$
$((A \cdot B) \in \mathbb{C} \wedge C \in \mathbb{C})$ by (rule MMI˙anim1i)
from S13 have S14: $(A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \longrightarrow$
$((A \cdot B) \in \mathbb{C} \wedge C \in \mathbb{C})$ by (rule MMI˙3impa)
from S11 S14 have S15: $((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge C \neq \mathbf{0}) \longrightarrow$
$((A \cdot B) / C) =$
$((A \cdot B) \cdot (\mathbf{1} / C))$ by (rule MMI˙sylan)
have S16: $(B \in \mathbb{C} \wedge C \in \mathbb{C} \wedge C \neq \mathbf{0}) \longrightarrow$
$(B / C) = (B \cdot (\mathbf{1} / C))$ by (rule MMI˙divrect)
from S16 have S17: $((B \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge C \neq \mathbf{0}) \longrightarrow$
$(B / C) = (B \cdot (\mathbf{1} / C))$ by (rule MMI˙3expa)
from S17 have S18: $((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge C \neq \mathbf{0}) \longrightarrow$
$(B / C) = (B \cdot (\mathbf{1} / C))$ by (rule MMI˙3adantl1)
from S18 have S19: $((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge C \neq \mathbf{0}) \longrightarrow$
$(A \cdot (B / C)) =$
$(A \cdot (B \cdot (\mathbf{1} / C)))$ by (rule MMI˙opreq2d)
from S9 S15 S19 show $((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge C \neq \mathbf{0}) \longrightarrow$
$((A \cdot B) / C) = (A \cdot (B / C))$ by (rule MMI˙3eqtr4d)
qed

lemma (in MMIsar0) MMI˙div23t:

shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A · B) / C) = ((A / C) · B)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A · B) = (B · A) by (rule MMI˙axmulcom)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A · B) = (B · A) by (rule MMI˙3adant3)
from S2 have S3: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(A · B) = (B · A) by (rule MMI˙adantr)
from S3 have S4: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A · B) / C) = ((B · A) / C) by (rule MMI˙opreq1d)
have S5: ((B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((B · A) / C) = (B · (A / C)) by (rule MMI˙divasst)
from S5 have S6: (B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ) ⟶
(C ≠ **0** ⟶
((B · A) / C) =
(B · (A / C))) by (rule MMI˙ex)
from S6 have S7: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(C ≠ **0** ⟶
((B · A) / C) =
(B · (A / C))) by (rule MMI˙3com12)
from S7 have S8: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((B · A) / C) = (B · (A / C)) by (rule MMI˙imp)
have S9: (B ∈ ℂ ∧ (A / C) ∈ ℂ) ⟶
(B · (A / C)) = ((A / C) · B) by (rule MMI˙axmulcom)
have S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ B ∈ ℂ by (rule MMI˙3simp2)
from S10 have S11: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
B ∈ ℂ by (rule MMI˙adantr)
have S12: (A ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
(A / C) ∈ ℂ by (rule MMI˙divclt)
from S12 have S13: ((A ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(A / C) ∈ ℂ by (rule MMI˙3expa)
from S13 have S14: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(A / C) ∈ ℂ by (rule MMI˙3adantl2)
from S9 S11 S14 have S15: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(B · (A / C)) = ((A / C) · B) by (rule MMI˙sylanc)
from S4 S8 S15 show ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A · B) / C) = ((A / C) · B) by (rule MMI˙3eqtrd)
qed

lemma (in MMIsar0) MMI˙div13t:
shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A / B) · C) = ((C / B) · A)
proof -
have S1: (A ∈ ℂ ∧ C ∈ ℂ) ⟶
(A · C) = (C · A) by (rule MMI˙axmulcom)
from S1 have S2: (A ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · C) / B) = ((C · A) / B) by (rule MMI˙opreq1d)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶

((A · C) / B) = ((C · A) / B) by (rule MMI˙3adant2)
from S3 have S4: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A · C) / B) = ((C · A) / B) by (rule MMI˙adantr)
have S5: ((A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ) ∧ B ≠ **0**) ⟶
((A · C) / B) = ((A / B) · C) by (rule MMI˙div23t)
from S5 have S6: (A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ) ⟶
(B ≠ **0** ⟶
((A · C) / B) =
((A / B) · C)) by (rule MMI˙ex)
from S6 have S7: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B ≠ **0** ⟶
((A · C) / B) =
((A / B) · C)) by (rule MMI˙3com23)
from S7 have S8: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A · C) / B) = ((A / B) · C) by (rule MMI˙imp)
have S9: ((C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ∧ B ≠ **0**) ⟶
((C · A) / B) = ((C / B) · A) by (rule MMI˙div23t)
from S9 have S10: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶
(B ≠ **0** ⟶
((C · A) / B) =
((C / B) · A)) by (rule MMI˙ex)
from S10 have S11: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B ≠ **0** ⟶
((C · A) / B) =
((C / B) · A)) by (rule MMI˙3coml)
from S11 have S12: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((C · A) / B) = ((C / B) · A) by (rule MMI˙imp)
from S4 S8 S12 show ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A / B) · C) = ((C / B) · A) by (rule MMI˙3eqtr3d)
qed

lemma (in MMIsar0) MMI˙div12t:
shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(A · (B / C)) = (B · (A / C))
proof -
have S1: (A ∈ ℂ ∧ (B / C) ∈ ℂ) ⟶
(A · (B / C)) = ((B / C) · A) by (rule MMI˙axmulcom)
have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ A ∈ ℂ by (rule MMI˙3simp1)
from S2 have S3: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
A ∈ ℂ by (rule MMI˙adantr)
have S4: (B ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
(B / C) ∈ ℂ by (rule MMI˙divclt)
from S4 have S5: ((B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(B / C) ∈ ℂ by (rule MMI˙3expa)
from S5 have S6: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(B / C) ∈ ℂ by (rule MMI˙3adantl1)
from S1 S3 S6 have S7: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(A · (B / C)) = ((B / C) · A) by (rule MMI˙sylanc)
have S8: ((B ∈ ℂ ∧ C ∈ ℂ ∧ A ∈ ℂ) ∧ C ≠ **0**) ⟶

((B / C) · A) = ((A / C) · B) by (rule MMI˙div13t)
from S8 have S9: (B ∈ ℂ ∧ C ∈ ℂ ∧ A ∈ ℂ) ⟶
(C ≠ **0** ⟶
((B / C) · A) =
((A / C) · B)) by (rule MMI˙ex)
from S9 have S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(C ≠ **0** ⟶
((B / C) · A) =
((A / C) · B)) by (rule MMI˙3comr)
from S10 have S11: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((B / C) · A) = ((A / C) · B) by (rule MMI˙imp)
have S12: ((A / C) ∈ ℂ ∧ B ∈ ℂ) ⟶
((A / C) · B) = (B · (A / C)) by (rule MMI˙axmulcom)
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: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ B ∈ ℂ by (rule MMI˙3simp2)
from S16 have S17: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
B ∈ ℂ by (rule MMI˙adantr)
from S12 S15 S17 have S18: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A / C) · B) = (B · (A / C)) by (rule MMI˙sylanc)
from S7 S11 S18 show ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(A · (B / C)) = (B · (A / C)) by (rule MMI˙3eqtrd)
qed

lemma (in MMIsar0) MMI˙divassz: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows C ≠ **0** ⟶
((A · B) / C) = (A · (B / C))
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A3 have S3: C ∈ ℂ.
from S1 S2 S3 have S4: A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ by (rule MMI˙3pm3˙2i)
have S5: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A · B) / C) = (A · (B / C)) by (rule MMI˙divasst)
from S4 S5 show C ≠ **0** ⟶
((A · B) / C) = (A · (B / C)) by (rule MMI˙mpan)
qed

lemma (in MMIsar0) MMI˙divass: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ and
A4: C ≠ **0**
shows ((A · B) / C) = (A · (B / C))

proof -
from A4 have S1: C $\neq$ **0**.
from A1 have S2: A $\in$ $\mathbb{C}$.
from A2 have S3: B $\in$ $\mathbb{C}$.
from A3 have S4: C $\in$ $\mathbb{C}$.
from S2 S3 S4 have S5: C $\neq$ **0** $\longrightarrow$
((A · B) / C) = (A · (B / C)) by (rule MMI˙divassz)
from S1 S5 show ((A · B) / C) = (A · (B / C)) by (rule MMI˙ax˙mp)
qed

lemma (in MMIsar0) MMI˙divdir: assumes A1: A $\in$ $\mathbb{C}$ and
A2: B $\in$ $\mathbb{C}$ and
A3: C $\in$ $\mathbb{C}$ and
A4: C $\neq$ **0**
shows ((A + B) / C) =
((A / C) + (B / C))
proof -
from A1 have S1: A $\in$ $\mathbb{C}$.
from A2 have S2: B $\in$ $\mathbb{C}$.
from A3 have S3: C $\in$ $\mathbb{C}$.
from A4 have S4: C $\neq$ **0**.
from S3 S4 have S5: (**1** / C) $\in$ $\mathbb{C}$ by (rule MMI˙reccl)
from S1 S2 S5 have S6: ((A + B) · (**1** / C)) =
((A · (**1** / C)) + (B · (**1** / C))) by (rule MMI˙adddir)
from A1 have S7: A $\in$ $\mathbb{C}$.
from A2 have S8: B $\in$ $\mathbb{C}$.
from S7 S8 have S9: (A + B) $\in$ $\mathbb{C}$ by (rule MMI˙addcl)
from A3 have S10: C $\in$ $\mathbb{C}$.
from A4 have S11: C $\neq$ **0**.
from S9 S10 S11 have S12: ((A + B) / C) =
((A + B) · (**1** / C)) by (rule MMI˙divrec)
from A1 have S13: A $\in$ $\mathbb{C}$.
from A3 have S14: C $\in$ $\mathbb{C}$.
from A4 have S15: C $\neq$ **0**.
from S13 S14 S15 have S16: (A / C) = (A · (**1** / C)) by (rule MMI˙divrec)
from A2 have S17: B $\in$ $\mathbb{C}$.
from A3 have S18: C $\in$ $\mathbb{C}$.
from A4 have S19: C $\neq$ **0**.
from S17 S18 S19 have S20: (B / C) = (B · (**1** / C)) by (rule MMI˙divrec)
from S16 S20 have S21: ((A / C) + (B / C)) =
((A · (**1** / C)) + (B · (**1** / C))) by (rule MMI˙opreq12i)
from S6 S12 S21 show ((A + B) / C) =
((A / C) + (B / C)) by (rule MMI˙3eqtr4)
qed

lemma (in MMIsar0) MMI˙div23: assumes A1: A $\in$ $\mathbb{C}$ and
A2: B $\in$ $\mathbb{C}$ and
A3: C $\in$ $\mathbb{C}$ and
A4: C $\neq$ **0**

shows ((A · B) / C) = ((A / C) · B)
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from S1 S2 have S3: (A · B) = (B · A) by (rule MMI˙mulcom)
from S3 have S4: ((A · B) / C) = ((B · A) / C)
by (rule MMI˙opreq1i)
from A2 have S5: B ∈ ℂ.
from A1 have S6: A ∈ ℂ.
from A3 have S7: C ∈ ℂ.
from A4 have S8: C ≠ **0**.
from S5 S6 S7 S8 have
S9: ((B · A) / C) = (B · (A / C)) by (rule MMI˙divass)
from A2 have S10: B ∈ ℂ.
from A1 have S11: A ∈ ℂ.
from A3 have S12: C ∈ ℂ.
from A4 have S13: C ≠ **0**.
from S11 S12 S13 have S14: (A / C) ∈ ℂ by (rule MMI˙divcl)
from S10 S14 have S15: (B · (A / C)) = ((A / C) · B)
by (rule MMI˙mulcom)
from S4 S9 S15 show ((A · B) / C) = ((A / C) · B)
by (rule MMI˙3eqtr)
qed

lemma (in MMIsar0) MMI˙divdirz: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ
shows C ≠ **0** ⟶
((A + B) / C) =
((A / C) + (B / C))
proof -
have S1: C =
if (C ≠ **0** , C , **1**) ⟶
((A + B) / C) =
((A + B) / if (C ≠ **0** , C , **1**)) by (rule MMI˙opreq2)
have S2: C =
if (C ≠ **0** , C , **1**) ⟶
(A / C) =
(A / if (C ≠ **0** , C , **1**)) by (rule MMI˙opreq2)
have S3: C =
if (C ≠ **0** , C , **1**) ⟶
(B / C) =
(B / if (C ≠ **0** , C , **1**)) by (rule MMI˙opreq2)
from S2 S3 have S4: C =
if (C ≠ **0** , C , **1**) ⟶
((A / C) + (B / C)) =

((A / if (C ≠ **0** , C , **1**)) + (B / if (C ≠ **0** , C , **1**))) by (rule MMI˙opreq12d)
from S1 S4 have S5: C =
if (C ≠ **0** , C , **1**) ⟶
(((A + B) / C) =
((A / C) + (B / C)) ⟷
((A + B) / if (C ≠ **0** , C , **1**)) =
((A / if (C ≠ **0** , C , **1**)) + (B / if (C ≠ **0** , C , **1**)))) by (rule MMI˙eqeq12d)
from A1 have S6: A ∈ ℂ.
from A2 have S7: B ∈ ℂ.
from A3 have S8: C ∈ ℂ.
have S9: **1** ∈ ℂ by (rule MMI˙1cn)
from S8 S9 have S10: if (C ≠ **0** , C , **1**) ∈ ℂ by (rule MMI˙keepel)
have S11: if (C ≠ **0** , C , **1**) ≠ **0** by (rule MMI˙elimne0)
from S6 S7 S10 S11 have S12: ((A + B) / if (C ≠ **0** , C , **1**)) =
((A / if (C ≠ **0** , C , **1**)) + (B / if (C ≠ **0** , C , **1**))) by (rule MMI˙divdir)
from S5 S12 show C ≠ **0** ⟶
((A + B) / C) =
((A / C) + (B / C)) by (rule MMI˙dedth)
qed

lemma (in MMIsar0) MMI˙divdirt:
shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A + B) / C) =
((A / C) + (B / C))
proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
(A + B) =
(if (A ∈ ℂ , A , **0**) + B) by (rule MMI˙opreq1)
from S1 have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
((A + B) / C) =
((if (A ∈ ℂ , A , **0**) + B) / C) by (rule MMI˙opreq1d)
have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
(A / C) =
(if (A ∈ ℂ , A , **0**) / C) by (rule MMI˙opreq1)
from S3 have S4: A =
if (A ∈ ℂ , A , **0**) ⟶
((A / C) + (B / C)) =
((if (A ∈ ℂ , A , **0**) / C) + (B / C)) by (rule MMI˙opreq1d)
from S2 S4 have S5: A =
if (A ∈ ℂ , A , **0**) ⟶
(((A + B) / C) =
((A / C) + (B / C)) ⟷
((if (A ∈ ℂ , A , **0**) + B) / C) =
((if (A ∈ ℂ , A , **0**) / C) + (B / C))) by (rule MMI˙eqeq12d)
from S5 have S6: A =
if (A ∈ ℂ , A , **0**) ⟶

((C ≠ **0** ⟶ ((A + B) / C) = ((A / C) + (B / C))) ⟷
(C ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) + B) / C) =
((if (A ∈ ℂ , A , **0**) / C) + (B / C)))) by (rule MMI˙imbi2d)
have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) + B) =
(if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
from S7 have S8: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) + B) / C) =
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / C) by (rule MMI˙opreq1d)
have S9: B =
if (B ∈ ℂ , B , **0**) ⟶
(B / C) =
(if (B ∈ ℂ , B , **0**) / C) by (rule MMI˙opreq1)
from S9 have S10: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) / C) + (B / C)) =
((if (A ∈ ℂ , A , **0**) / C) + (if (B ∈ ℂ , B , **0**) / C)) by (rule MMI˙opreq2d)
from S8 S10 have S11: B =
if (B ∈ ℂ , B , **0**) ⟶
(((if (A ∈ ℂ , A , **0**) + B) / C) =
((if (A ∈ ℂ , A , **0**) / C) + (B / C)) ⟷
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / C) =
((if (A ∈ ℂ , A , **0**) / C) + (if (B ∈ ℂ , B , **0**) / C))) by (rule MMI˙eqeq12d)
from S11 have S12: B =
if (B ∈ ℂ , B , **0**) ⟶
((C ≠ **0** ⟶ ((if (A ∈ ℂ , A , **0**) + B) / C) = ((if (A ∈ ℂ , A , **0**) / C)
+ (B / C))) ⟷
(C ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / C) =
((if (A ∈ ℂ , A , **0**) / C) + (if (B ∈ ℂ , B , **0**) / C)))) by (rule MMI˙imbi2d)
have S13: C =
if (C ∈ ℂ , C , **0**) ⟶
(C ≠ **0** ⟷ if (C ∈ ℂ , C , **0**) ≠ **0**) by (rule MMI˙neeq1)
have S14: C =
if (C ∈ ℂ , C , **0**) ⟶
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / C) =
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / if (C ∈ ℂ , C , **0**)) by (rule
MMI˙opreq2)
have S15: C =
if (C ∈ ℂ , C , **0**) ⟶
(if (A ∈ ℂ , A , **0**) / C) =
(if (A ∈ ℂ , A , **0**) / if (C ∈ ℂ , C , **0**)) by (rule MMI˙opreq2)
have S16: C =
if (C ∈ ℂ , C , **0**) ⟶
(if (B ∈ ℂ , B , **0**) / C) =
(if (B ∈ ℂ , B , **0**) / if (C ∈ ℂ , C , **0**)) by (rule MMI˙opreq2)

from S15 S16 have S17: C =
if (C ∈ ℂ , C , **0**) ⟶
((if (A ∈ ℂ , A , **0**) / C) + (if (B ∈ ℂ , B , **0**) / C)) =
((if (A ∈ ℂ , A , **0**) / if (C ∈ ℂ , C , **0**)) + (if (B ∈ ℂ , B , **0**) / if (C ∈ ℂ , C , **0**))) by (rule MMI˙opreq12d)
from S14 S17 have S18: C =
if (C ∈ ℂ , C , **0**) ⟶
(((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / C) =
((if (A ∈ ℂ , A , **0**) / C) + (if (B ∈ ℂ , B , **0**) / C)) ⟷
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / if (C ∈ ℂ , C , **0**)) =
((if (A ∈ ℂ , A , **0**) / if (C ∈ ℂ , C , **0**)) + (if (B ∈ ℂ , B , **0**) / if (C ∈ ℂ , C , **0**)))) by (rule MMI˙eqeq12d)
from S13 S18 have S19: C =
if (C ∈ ℂ , C , **0**) ⟶
((C ≠ **0** ⟶ ((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / C) = ((if (A ∈ ℂ , A , **0**) / C) + (if (B ∈ ℂ , B , **0**) / C))) ⟷
(if (C ∈ ℂ , C , **0**) ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / if (C ∈ ℂ , C , **0**)) =
((if (A ∈ ℂ , A , **0**) / if (C ∈ ℂ , C , **0**)) + (if (B ∈ ℂ , B , **0**) / if (C ∈ ℂ , C , **0**))))) by (rule MMI˙imbi12d)
have S20: **0** ∈ ℂ by (rule MMI˙0cn)
from S20 have S21: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S22: **0** ∈ ℂ by (rule MMI˙0cn)
from S22 have S23: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
have S24: **0** ∈ ℂ by (rule MMI˙0cn)
from S24 have S25: if (C ∈ ℂ , C , **0**) ∈ ℂ by (rule MMI˙elimel)
from S21 S23 S25 have S26: if (C ∈ ℂ , C , **0**) ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / if (C ∈ ℂ , C , **0**)) =
((if (A ∈ ℂ , A , **0**) / if (C ∈ ℂ , C , **0**)) + (if (B ∈ ℂ , B , **0**) / if (C ∈ ℂ , C , **0**))) by (rule MMI˙divdirz)
from S6 S12 S19 S26 have S27: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(C ≠ **0** ⟶
((A + B) / C) =
((A / C) + (B / C))) by (rule MMI˙dedth3h)
from S27 show ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A + B) / C) =
((A / C) + (B / C)) by (rule MMI˙imp)
qed

lemma (in MMIsar0) MMI˙divcan3: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: A ≠ **0**
shows ((A · B) / A) = B
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from A1 have S3: A ∈ ℂ.
from A3 have S4: A ≠ **0**.
from S1 S2 S3 S4 have S5: ((A · B) / A) = (A · (B / A)) by (rule MMI˙divass)

from A1 have S6: A ∈ ℂ.
from A2 have S7: B ∈ ℂ.
from A3 have S8: A ≠ **0**.
from S6 S7 S8 have S9: (A · (B / A)) = B by (rule MMI˙divcan2)
from S5 S9 show ((A · B) / A) = B by (rule MMI˙eqtr)
qed

lemma (in MMIsar0) MMI˙divcan4: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: A ≠ **0**
shows ((B · A) / A) = B
proof -
from A2 have S1: B ∈ ℂ.
from A1 have S2: A ∈ ℂ.
from S1 S2 have S3: (B · A) = (A · B) by (rule MMI˙mulcom)
from S3 have S4: ((B · A) / A) = ((A · B) / A) by (rule MMI˙opreq1i)
from A1 have S5: A ∈ ℂ.
from A2 have S6: B ∈ ℂ.
from A3 have S7: A ≠ **0**.
from S5 S6 S7 have S8: ((A · B) / A) = B by (rule MMI˙divcan3)
from S4 S8 show ((B · A) / A) = B by (rule MMI˙eqtr)
qed

lemma (in MMIsar0) MMI˙divcan3z: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows A ≠ **0** ⟶ ((A · B) / A) = B
proof -
have S1: A =
if (A ≠ **0** , A , **1**) ⟶
(A · B) =
(if (A ≠ **0** , A , **1**) · B) by (rule MMI˙opreq1)
have S2: A =
if (A ≠ **0** , A , **1**) ⟶
A = if (A ≠ **0** , A , **1**) by (rule MMI˙id)
from S1 S2 have S3: A =
if (A ≠ **0** , A , **1**) ⟶
((A · B) / A) =
((if (A ≠ **0** , A , **1**) · B) / if (A ≠ **0** , A , **1**)) by (rule MMI˙opreq12d)
from S3 have S4: A =
if (A ≠ **0** , A , **1**) ⟶
(((A · B) / A) =
B ⟷
((if (A ≠ **0** , A , **1**) · B) / if (A ≠ **0** , A , **1**)) =
B) by (rule MMI˙eqeq1d)
from A1 have S5: A ∈ ℂ.
have S6: **1** ∈ ℂ by (rule MMI˙1cn)
from S5 S6 have S7: if (A ≠ **0** , A , **1**) ∈ ℂ by (rule MMI˙keepel)
from A2 have S8: B ∈ ℂ.
have S9: if (A ≠ **0** , A , **1**) ≠ **0** by (rule MMI˙elimne0)

from S7 S8 S9 have S10: ((if (A ≠ **0** , A , **1**) · B) / if (A ≠ **0** , A , **1**)) = B by (rule MMI˙divcan3)
from S4 S10 show A ≠ **0** ⟶ ((A · B) / A) = B by (rule MMI˙dedth)
qed

lemma (in MMIsar0) MMI˙divcan4z: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ
shows A ≠ **0** ⟶ ((B · A) / A) = B
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: B ∈ ℂ.
from S1 S2 have S3: A ≠ **0** ⟶ ((A · B) / A) = B by (rule MMI˙divcan3z)
from A2 have S4: B ∈ ℂ.
from A1 have S5: A ∈ ℂ.
from S4 S5 have S6: (B · A) = (A · B) by (rule MMI˙mulcom)
from S6 have S7: ((B · A) / A) = ((A · B) / A) by (rule MMI˙opreq1i)
from S3 S7 show A ≠ **0** ⟶ ((B · A) / A) = B by (rule MMI˙syl5eq)
qed

lemma (in MMIsar0) MMI˙divcan3t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶
((A · B) / A) = B
proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
(A ≠ **0** ⟷ if (A ∈ ℂ , A , **0**) ≠ **0**) by (rule MMI˙neeq1)
have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
(A · B) =
(if (A ∈ ℂ , A , **0**) · B) by (rule MMI˙opreq1)
have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
A = if (A ∈ ℂ , A , **0**) by (rule MMI˙id)
from S2 S3 have S4: A =
if (A ∈ ℂ , A , **0**) ⟶
((A · B) / A) =
((if (A ∈ ℂ , A , **0**) · B) / if (A ∈ ℂ , A , **0**)) by (rule MMI˙opreq12d)
from S4 have S5: A =
if (A ∈ ℂ , A , **0**) ⟶
(((A · B) / A) =
B ⟷
((if (A ∈ ℂ , A , **0**) · B) / if (A ∈ ℂ , A , **0**)) =
B) by (rule MMI˙eqeq1d)
from S1 S5 have S6: A =
if (A ∈ ℂ , A , **0**) ⟶
((A ≠ **0** ⟶ ((A · B) / A) = B) ⟷
(if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) · B) / if (A ∈ ℂ , A , **0**)) =
B)) by (rule MMI˙imbi12d)

have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) · B) =
(if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) by (rule MMI˙opreq2)
from S7 have S8: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) · B) / if (A ∈ ℂ , A , **0**)) =
((if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) / if (A ∈ ℂ , A , **0**)) by (rule MMI˙opreq1d)
have S9: B =
if (B ∈ ℂ , B , **0**) ⟶
B = if (B ∈ ℂ , B , **0**) by (rule MMI˙id)
from S8 S9 have S10: B =
if (B ∈ ℂ , B , **0**) ⟶
(((if (A ∈ ℂ , A , **0**) · B) / if (A ∈ ℂ , A , **0**)) =
B ⟷
((if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) / if (A ∈ ℂ , A , **0**)) =
if (B ∈ ℂ , B , **0**)) by (rule MMI˙eqeq12d)
from S10 have S11: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) ≠ **0** ⟶ ((if (A ∈ ℂ , A , **0**) · B) / if (A ∈ ℂ , A , **0**)) = B) ⟷
(if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) / if (A ∈ ℂ , A , **0**)) =
if (B ∈ ℂ , B , **0**))) by (rule MMI˙imbi2d)
have S12: **0** ∈ ℂ by (rule MMI˙0cn)
from S12 have S13: if (A ∈ ℂ , A , **0**) ∈ ℂ by (rule MMI˙elimel)
have S14: **0** ∈ ℂ by (rule MMI˙0cn)
from S14 have S15: if (B ∈ ℂ , B , **0**) ∈ ℂ by (rule MMI˙elimel)
from S13 S15 have S16: if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) / if (A ∈ ℂ , A , **0**)) =
if (B ∈ ℂ , B , **0**) by (rule MMI˙divcan3z)
from S6 S11 S16 have S17: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A ≠ **0** ⟶ ((A · B) / A) = B) by (rule MMI˙dedth2h)
from S17 show (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶
((A · B) / A) = B by (rule MMI˙3impia)
qed

lemma (in MMIsar0) MMI˙divcan4t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶
((B · A) / A) = B
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A · B) = (B · A) by (rule MMI˙axmulcom)
from S1 have S2: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A · B) / A) = ((B · A) / A) by (rule MMI˙opreq1d)
from S2 have S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶
((A · B) / A) = ((B · A) / A) by (rule MMI˙3adant3)
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶

((A · B) / A) = B by (rule MMI˙divcan3t)
from S3 S4 show (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶
((B · A) / A) = B by (rule MMI˙eqtr3d)
qed

lemma (in MMIsar0) MMI˙div11: assumes A1: A ∈ ℂ and
A2: B ∈ ℂ and
A3: C ∈ ℂ and
A4: C ≠ **0**
shows (A / C) = (B / C) ⟷ A = B
proof -
from A3 have S1: C ∈ ℂ.
from A1 have S2: A ∈ ℂ.
from A3 have S3: C ∈ ℂ.
from A4 have S4: C ≠ **0**.
from S2 S3 S4 have S5: (A / C) ∈ ℂ by (rule MMI˙divcl)
from A2 have S6: B ∈ ℂ.
from A3 have S7: C ∈ ℂ.
from A4 have S8: C ≠ **0**.
from S6 S7 S8 have S9: (B / C) ∈ ℂ by (rule MMI˙divcl)
from A4 have S10: C ≠ **0**.
from S1 S5 S9 S10 have S11: (C · (A / C)) =
(C · (B / C)) ⟷
(A / C) = (B / C) by (rule MMI˙mulcan)
from A3 have S12: C ∈ ℂ.
from A1 have S13: A ∈ ℂ.
from A4 have S14: C ≠ **0**.
from S12 S13 S14 have S15: (C · (A / C)) = A by (rule MMI˙divcan2)
from A3 have S16: C ∈ ℂ.
from A2 have S17: B ∈ ℂ.
from A4 have S18: C ≠ **0**.
from S16 S17 S18 have S19: (C · (B / C)) = B by (rule MMI˙divcan2)
from S15 S19 have S20: (C · (A / C)) =
(C · (B / C)) ⟷ A = B by (rule MMI˙eqeq12i)
from S11 S20 show (A / C) = (B / C) ⟷ A = B by (rule MMI˙bitr3)
qed

lemma (in MMIsar0) MMI˙div11t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ (C ∈ ℂ ∧ C ≠ **0**)) ⟶
((A / C) = (B / C) ⟷ A = B)
proof -
have S1: A =
if (A ∈ ℂ , A , **1**) ⟶
(A / C) =
(if (A ∈ ℂ , A , **1**) / C) by (rule MMI˙opreq1)
from S1 have S2: A =
if (A ∈ ℂ , A , **1**) ⟶
((A / C) =
(B / C) ⟷

(if (A ∈ ℂ , A , **1**) / C) =
(B / C)) by (rule MMI˙eqeq1d)
have S3: A =
if (A ∈ ℂ , A , **1**) ⟶
(A = B ⟷ if (A ∈ ℂ , A , **1**) = B) by (rule MMI˙eqeq1)
from S2 S3 have S4: A =
if (A ∈ ℂ , A , **1**) ⟶
(((A / C) = (B / C) ⟷ A = B) ⟷
((if (A ∈ ℂ , A , **1**) / C) =
(B / C) ⟷
if (A ∈ ℂ , A , **1**) = B)) by (rule MMI˙bibi12d)
have S5: B =
if (B ∈ ℂ , B , **1**) ⟶
(B / C) =
(if (B ∈ ℂ , B , **1**) / C) by (rule MMI˙opreq1)
from S5 have S6: B =
if (B ∈ ℂ , B , **1**) ⟶
((if (A ∈ ℂ , A , **1**) / C) =
(B / C) ⟷
(if (A ∈ ℂ , A , **1**) / C) =
(if (B ∈ ℂ , B , **1**) / C)) by (rule MMI˙eqeq2d)
have S7: B =
if (B ∈ ℂ , B , **1**) ⟶
(if (A ∈ ℂ , A , **1**) =
B ⟷
if (A ∈ ℂ , A , **1**) =
if (B ∈ ℂ , B , **1**)) by (rule MMI˙eqeq2)
from S6 S7 have S8: B =
if (B ∈ ℂ , B , **1**) ⟶
(((if (A ∈ ℂ , A , **1**) / C) = (B / C) ⟷ if (A ∈ ℂ , A , **1**) = B) ⟷
((if (A ∈ ℂ , A , **1**) / C) =
(if (B ∈ ℂ , B , **1**) / C) ⟷
if (A ∈ ℂ , A , **1**) =
if (B ∈ ℂ , B , **1**))) by (rule MMI˙bibi12d)
have S9: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(if (A ∈ ℂ , A , **1**) / C) =
(if (A ∈ ℂ , A , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**)) by (rule MMI˙opreq2)
have S10: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(if (B ∈ ℂ , B , **1**) / C) =
(if (B ∈ ℂ , B , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**)) by (rule MMI˙opreq2)
from S9 S10 have S11: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
((if (A ∈ ℂ , A , **1**) / C) =
(if (B ∈ ℂ , B , **1**) / C) ⟷
(if (A ∈ ℂ , A , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**)) =
(if (B ∈ ℂ , B , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**))) by (rule MMI˙eqeq12d)
from S11 have S12: C =

if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(((if (A ∈ ℂ , A , **1**) / C) = (if (B ∈ ℂ , B , **1**) / C) ⟷ if (A ∈ ℂ , A , **1**) = if (B ∈ ℂ , B , **1**)) ⟷
((if (A ∈ ℂ , A , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**)) =
(if (B ∈ ℂ , B , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**)) ⟷
if (A ∈ ℂ , A , **1**) =
if (B ∈ ℂ , B , **1**))) by (rule MMI˙bibi1d)
have S13: **1** ∈ ℂ by (rule MMI˙1cn)
from S13 have S14: if (A ∈ ℂ , A , **1**) ∈ ℂ by (rule MMI˙elimel)
have S15: **1** ∈ ℂ by (rule MMI˙1cn)
from S15 have S16: if (B ∈ ℂ , B , **1**) ∈ ℂ by (rule MMI˙elimel)
have S17: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(C ∈ ℂ ⟷
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ∈ ℂ) by (rule MMI˙eleq1)
have S18: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(C ≠ **0** ⟷
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ≠ **0**) by (rule MMI˙neeq1)
from S17 S18 have S19: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
((C ∈ ℂ ∧ C ≠ **0**) ⟷
(if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ∈ ℂ ∧ if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ≠ **0**)) by (rule MMI˙anbi12d)
have S20: **1** =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(**1** ∈ ℂ ⟷
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ∈ ℂ) by (rule MMI˙eleq1)
have S21: **1** =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(**1** ≠ **0** ⟷
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ≠ **0**) by (rule MMI˙neeq1)
from S20 S21 have S22: **1** =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
((**1** ∈ ℂ ∧ **1** ≠ **0**) ⟷
(if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ∈ ℂ ∧ if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ≠ **0**)) by (rule MMI˙anbi12d)
have S23: **1** ∈ ℂ by (rule MMI˙1cn)
have S24: **1** ≠ **0** by (rule MMI˙ax1ne0)
from S23 S24 have S25: **1** ∈ ℂ ∧ **1** ≠ **0** by (rule MMI˙pm3˙2i)
from S19 S22 S25 have S26: if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ∈ ℂ ∧ if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ≠ **0** by (rule MMI˙elimhyp)
from S26 have S27: if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ∈ ℂ by (rule MMI˙pm3˙26i)
from S26 have S28: if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ∈ ℂ ∧ if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ≠ **0** .
from S28 have S29: if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ≠ **0** by (rule MMI˙pm3˙27i)
from S14 S16 S27 S29 have S30: (if (A ∈ ℂ , A , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**)) =
(if (B ∈ ℂ , B , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**)) ⟷

if (A ∈ ℂ , A , **1**) =
if (B ∈ ℂ , B , **1**) by (rule MMI˙div11)
from S4 S8 S12 S30 show (A ∈ ℂ ∧ B ∈ ℂ ∧ (C ∈ ℂ ∧ C ≠ **0**)) ⟶
((A / C) = (B / C) ⟷ A = B) by (rule MMI˙dedth3h)
qed

end

80 Metamath examples

theory MMI˙examples imports MMI˙Complex˙ZF

begin

This theory contains 10 theorems translated from Metamath (with proofs). It is included in the proof document as an illustration of how a translated Metamath proof looks like. The "known_theorems.txt" file included in the IsarMathLib distribution provides a list of all translated facts.

lemma (in MMIsar0) MMI˙dividt:
shows (A ∈ ℂ ∧ A ≠ **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 ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A - B) / C) =
((A / C) - (B / C)) by (rule MMI˙3eqtr3d)
qed

end

81 Metamath interface

theory Metamath˙Interface imports Complex˙ZF MMI˙prelude

begin

This theory contains some lemmas that make it possible to use the theorems translated from Metamath in a the complex0 context.

81.1 MMisar0 and complex0 contexts.

In the section we show a lemma that the assumptions in complex0 context imply the assumptions of the MMIsar0 context. The Metamath˙sampler theory provides examples how this lemma can be used.

The next lemma states that we can use the theorems proven in the MMIsar0 context in the complex0 context. Unfortunately we have to use low level Isabelle methods "rule" and "unfold" in the proof, simp and blast fail on the order axioms.

lemma (in complex0) MMIsar˙valid:
shows MMIsar0(ℝ,ℂ,**1**,**0**,i,CplxAdd(R,A),CplxMul(R,A,M),
StrictVersion(CplxROrder(R,A,r)))
proof -
let real = ℝ
let complex = ℂ
let zero = **0**
let one = **1**
let iunit = i
let caddset = CplxAdd(R,A)

```
let cmulset = CplxMul(R,A,M)
let lessrrel = StrictVersion(CplxROrder(R,A,r))
have (∀a b. a ∈ real ∧ b ∈ real ⟶
  ⟨a, b⟩ ∈ lessrrel ⟷ ¬ (a = b ∨ ⟨b, a⟩ ∈ lessrrel))
proof -
  have I:
    ∀a b. a ∈ ℝ ∧ b ∈ ℝ ⟶ (a <ℝ b ⟷ ¬(a=b ∨ b <ℝ a))
    using pre˙axlttri by blast
  { fix a b assume a ∈ real ∧ b ∈ real
    with I have (a <ℝ b ⟷ ¬(a=b ∨ b <ℝ a))
by blast
    hence
⟨a, b⟩ ∈ lessrrel ⟷ ¬ (a = b ∨ ⟨b, a⟩ ∈ lessrrel)
by simp
  } thus (∀a b. a ∈ real ∧ b ∈ real ⟶
(⟨a, b⟩ ∈ lessrrel ⟷ ¬ (a = b ∨ ⟨b, a⟩ ∈ lessrrel)))
    by blast
qed
moreover
have (∀a b c.
  a ∈ real ∧ b ∈ real ∧ c ∈ real ⟶
  ⟨a, b⟩ ∈ lessrrel ∧ ⟨b, c⟩ ∈ lessrrel ⟶ ⟨a, c⟩ ∈ lessrrel)
proof -
  have II: ∀a b c.  a ∈ ℝ ∧ b ∈ ℝ ∧ c ∈ ℝ ⟶
    ((a <ℝ b ∧ b <ℝ c) ⟶ a <ℝ c)
    using pre˙axlttrn by blast
  { fix a b c assume a ∈ real ∧ b ∈ real ∧ c ∈ real
    with II have (a <ℝ b ∧ b <ℝ c) ⟶ a <ℝ c
by blast
    hence
⟨a, b⟩ ∈ lessrrel ∧ ⟨b, c⟩ ∈ lessrrel ⟶ ⟨a, c⟩ ∈ lessrrel
by simp
  } thus  (∀a b c.
a ∈ real ∧ b ∈ real ∧ c ∈ real ⟶
⟨a, b⟩ ∈ lessrrel ∧ ⟨b, c⟩ ∈ lessrrel ⟶ ⟨a, c⟩ ∈ lessrrel)
    by blast
qed
moreover have (∀A B C.
  A ∈ real ∧ B ∈ real ∧ C ∈ real ⟶
  ⟨A, B⟩ ∈ lessrrel ⟶
  ⟨caddset  ⟨C, A⟩, caddset  ⟨C, B⟩⟩ ∈ lessrrel)
  using pre˙axltadd by simp
moreover have (∀A B. A ∈ real ∧ B ∈ real ⟶
  ⟨zero, A⟩ ∈ lessrrel ∧ ⟨zero, B⟩ ∈ lessrrel ⟶
  ⟨zero, cmulset  ⟨A, B⟩⟩ ∈ lessrrel)
  using pre˙axmulgt0 by simp
moreover have
  (∀S. S ⊆ real ∧ S ≠ 0 ∧ (∃x∈real. ∀y∈S. ⟨y, x⟩ ∈ lessrrel) ⟶
  (∃x∈real.
```

(∀ y∈S. ⟨x, y⟩ ∉ lessrrel) ∧
(∀ y∈real. ⟨y, x⟩ ∈ lessrrel ⟶ (∃ z∈S. ⟨y, z⟩ ∈ lessrrel))))
using pre˙axsup by simp
moreover have ℝ ⊆ ℂ using axresscn by simp
moreover have **1** ≠ **0** using ax1ne0 by simp
moreover have ℂ isASet by simp
moreover have CplxAdd(R,A) : ℂ × ℂ → ℂ
using axaddopr by simp
moreover have CplxMul(R,A,M) : ℂ × ℂ → ℂ
using axmulopr by simp
moreover have
∀ a b. a ∈ ℂ ∧ b ∈ ℂ ⟶ a· b = b · a
using axmulcom by simp
hence (∀ a b. a ∈ ℂ ∧ b ∈ ℂ ⟶
cmulset ⟨a, b⟩ = cmulset ⟨b, a⟩
) by simp
moreover have ∀ a b. a ∈ ℂ ∧ b ∈ ℂ ⟶ a + b ∈ ℂ
using axaddcl by simp
hence (∀ a b. a ∈ ℂ ∧ b ∈ ℂ ⟶
caddset ⟨a, b⟩ ∈ ℂ
) by simp
moreover have ∀ a b. a ∈ ℂ ∧ b ∈ ℂ ⟶ a · b ∈ ℂ
using axmulcl by simp
hence (∀ a b. a ∈ ℂ ∧ b ∈ ℂ ⟶
cmulset ⟨a, b⟩ ∈ ℂ) by simp
moreover have
∀ a b C. a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
a · (b + C) = a · b + a · C
using axdistr by simp
hence ∀ a b C.
a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
cmulset ⟨a, caddset ⟨b, C⟩⟩ =
caddset
⟨cmulset ⟨a, b⟩, cmulset ⟨a, C⟩⟩
by simp
moreover have ∀ a b. a ∈ ℂ ∧ b ∈ ℂ ⟶
a + b = b + a
using axaddcom by simp
hence ∀ a b.
a ∈ ℂ ∧ b ∈ ℂ ⟶
caddset ⟨a, b⟩ = caddset ⟨b, a⟩ by simp
moreover have ∀ a b C. a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
a + b + C = a + (b + C)
using axaddass by simp
hence ∀ a b C.
a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
caddset ⟨caddset ⟨a, b⟩, C⟩ =
caddset ⟨a, caddset ⟨b, C⟩⟩ by simp
moreover have

$\forall$ a b c. a $\in \mathbb{C} \wedge$ b $\in \mathbb{C} \wedge$ c $\in \mathbb{C} \longrightarrow$ a·b·c = a·(b·c)
using axmulass by simp
hence $\forall$ a b C.
a $\in \mathbb{C} \wedge$ b $\in \mathbb{C} \wedge$ C $\in \mathbb{C} \longrightarrow$
cmulset $\langle$cmulset $\langle$a, b$\rangle$, C$\rangle$ =
cmulset $\langle$a, cmulset $\langle$b, C$\rangle\rangle$ by simp
moreover have $\mathbf{1} \in \mathbb{R}$ using ax1re by simp
moreover have i·i + $\mathbf{1} = \mathbf{0}$
using axi2m1 by simp
hence caddset $\langle$cmulset $\langle$i, i$\rangle$, $\mathbf{1}\rangle = \mathbf{0}$ by simp
moreover have $\forall$ a. a $\in \mathbb{C} \longrightarrow$ a + $\mathbf{0}$ = a
using ax0id by simp
hence $\forall$ a. a $\in \mathbb{C} \longrightarrow$ caddset $\langle$a, $\mathbf{0}\rangle$ = a by simp
moreover have i $\in \mathbb{C}$ using axicn by simp
moreover have $\forall$ a. a $\in \mathbb{C} \longrightarrow (\exists$ x$\in\mathbb{C}$. a + x = $\mathbf{0})$
using axnegex by simp
hence $\forall$ a. a $\in \mathbb{C} \longrightarrow$
($\exists$ x$\in\mathbb{C}$. caddset $\langle$a, x$\rangle = \mathbf{0}$) by simp
moreover have $\forall$ a. a $\in \mathbb{C} \wedge$ a $\neq \mathbf{0} \longrightarrow (\exists$ x$\in\mathbb{C}$. a · x = $\mathbf{1})$
using axrecex by simp
hence $\forall$ a. a $\in \mathbb{C} \wedge$ a $\neq \mathbf{0} \longrightarrow$
($\exists$ x$\in\mathbb{C}$. cmulset $\langle$a, x$\rangle = \mathbf{1}$) by simp
moreover have $\forall$ a. a $\in \mathbb{C} \longrightarrow$ a·$\mathbf{1}$ = a
using ax1id by simp
hence $\forall$ a. a $\in \mathbb{C} \longrightarrow$
cmulset $\langle$a, $\mathbf{1}\rangle$ = a by simp
moreover have $\forall$ a b. a $\in \mathbb{R} \wedge$ b $\in \mathbb{R} \longrightarrow$ a + b $\in \mathbb{R}$
using axaddrcl by simp
hence $\forall$ a b. a $\in \mathbb{R} \wedge$ b $\in \mathbb{R} \longrightarrow$
caddset $\langle$a, b$\rangle \in \mathbb{R}$ by simp
moreover have $\forall$ a b. a $\in \mathbb{R} \wedge$ b $\in \mathbb{R} \longrightarrow$ a · b $\in \mathbb{R}$
using axmulrcl by simp
hence $\forall$ a b. a $\in \mathbb{R} \wedge$ b $\in \mathbb{R} \longrightarrow$
cmulset $\langle$a, b$\rangle \in \mathbb{R}$ by simp
moreover have $\forall$ a. a $\in \mathbb{R} \longrightarrow (\exists$ x$\in\mathbb{R}$. a + x = $\mathbf{0})$
using axrnegex by simp
hence $\forall$ a. a $\in \mathbb{R} \longrightarrow$
($\exists$ x$\in\mathbb{R}$. caddset $\langle$a, x$\rangle = \mathbf{0}$) by simp
moreover have $\forall$ a. a $\in \mathbb{R} \wedge$ a$\neq\mathbf{0} \longrightarrow (\exists$ x$\in\mathbb{R}$. a · x = $\mathbf{1})$
using axrrecex by simp
hence $\forall$ a. a $\in \mathbb{R} \wedge$ a $\neq \mathbf{0} \longrightarrow$
($\exists$ x$\in\mathbb{R}$. cmulset $\langle$a, x$\rangle = \mathbf{1}$) by simp

ultimately have
(
(
(
($\forall$ a b.
a $\in \mathbb{R} \wedge$ b $\in \mathbb{R} \longrightarrow$

$\langle a, b\rangle \in \text{lessrrel} \longleftrightarrow$
$\neg\ (a = b \vee \langle b, a\rangle \in \text{lessrrel})$
$) \wedge$

$(\ \forall a\ b\ C.$
$a \in \mathbb{R} \wedge b \in \mathbb{R} \wedge C \in \mathbb{R} \longrightarrow$
$\langle a, b\rangle \in \text{lessrrel} \wedge$
$\langle b, C\rangle \in \text{lessrrel} \longrightarrow$
$\langle a, C\rangle \in \text{lessrrel}$
$) \wedge$

$(\forall a\ b\ C.$
$a \in \mathbb{R} \wedge b \in \mathbb{R} \wedge C \in \mathbb{R} \longrightarrow$
$\langle a, b\rangle \in \text{lessrrel} \longrightarrow$
$\langle \text{caddset}\ \ \langle C, a\rangle, \text{caddset}\ \ \langle C, b\rangle\rangle \in$
lessrrel
$)$
$) \wedge$

$($
$(\ \forall a\ b.$
$a \in \mathbb{R} \wedge b \in \mathbb{R} \longrightarrow$
$\langle \mathbf{0}, a\rangle \in \text{lessrrel} \wedge$
$\langle \mathbf{0}, b\rangle \in \text{lessrrel} \longrightarrow$
$\langle \mathbf{0}, \text{cmulset}\ \ \langle a, b\rangle\rangle \in$
lessrrel
$) \wedge$

$(\ \forall S.\ S \subseteq \mathbb{R} \wedge S \neq 0 \wedge$
$(\ \exists x{\in}\mathbb{R}.\ \forall y{\in}S.\ \langle y, x\rangle \in \text{lessrrel}$
$) \longrightarrow$
$(\ \exists x{\in}\mathbb{R}.$
$(\ \forall y{\in}S.\ \langle x, y\rangle \notin \text{lessrrel}$
$) \wedge$
$(\ \forall y{\in}\mathbb{R}.\ \langle y, x\rangle \in \text{lessrrel} \longrightarrow$
$(\ \exists z{\in}S.\ \langle y, z\rangle \in \text{lessrrel}$
$)$
$)$
$)$
$)$
$) \wedge$

$\mathbb{R} \subseteq \mathbb{C} \wedge$
$\mathbf{1} \neq \mathbf{0}$
$) \wedge$

$(\ \mathbb{C}\ \text{isASet} \wedge \text{caddset} \in \mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C} \wedge$
$\text{cmulset} \in \mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C}$
$) \wedge$

$$
\begin{array}{l}
\quad (\\
\quad\quad (\forall\, a\ b.\\
\quad\quad\quad a \in \mathbb{C} \wedge b \in \mathbb{C} \longrightarrow\\
\quad\quad\quad \text{cmulset}\ \langle a, b\rangle = \text{cmulset}\ \langle b, a\rangle\\
\quad\quad) \wedge\\
\\
\quad\quad (\forall\, a\ b.\ a \in \mathbb{C} \wedge b \in \mathbb{C} \longrightarrow\\
\quad\quad\quad \text{caddset}\ \langle a, b\rangle \in \mathbb{C}\\
\quad\quad)\\
\\
\quad) \wedge\\
\\
\quad (\forall\, a\ b.\ a \in \mathbb{C} \wedge b \in \mathbb{C} \longrightarrow\\
\quad\quad \text{cmulset}\ \langle a, b\rangle \in \mathbb{C}\\
\quad) \wedge\\
\\
\quad (\forall\, a\ b\ C.\\
\quad\quad\quad a \in \mathbb{C} \wedge b \in \mathbb{C} \wedge C \in \mathbb{C} \longrightarrow\\
\quad\quad\quad \text{cmulset}\ \langle a, \text{caddset}\ \langle b, C\rangle\rangle =\\
\quad\quad\quad \text{caddset}\\
\quad\quad\quad \langle \text{cmulset}\ \langle a, b\rangle, \text{cmulset}\ \langle a, C\rangle\rangle\\
\quad)\\
) \wedge
\end{array}
$$

$$
\begin{array}{l}
(\\
\quad (\\
\quad\quad (\forall\, a\ b.\\
\quad\quad\quad a \in \mathbb{C} \wedge b \in \mathbb{C} \longrightarrow\\
\quad\quad\quad \text{caddset}\ \langle a, b\rangle = \text{caddset}\ \langle b, a\rangle\\
\quad\quad) \wedge\\
\\
\quad\quad (\forall\, a\ b\ C.\\
\quad\quad\quad a \in \mathbb{C} \wedge b \in \mathbb{C} \wedge C \in \mathbb{C} \longrightarrow\\
\quad\quad\quad \text{caddset}\ \langle \text{caddset}\ \langle a, b\rangle, C\rangle =\\
\quad\quad\quad \text{caddset}\ \langle a, \text{caddset}\ \langle b, C\rangle\rangle\\
\quad\quad) \wedge\\
\\
\quad\quad (\forall\, a\ b\ C.\\
\quad\quad\quad a \in \mathbb{C} \wedge b \in \mathbb{C} \wedge C \in \mathbb{C} \longrightarrow\\
\quad\quad\quad \text{cmulset}\ \langle \text{cmulset}\ \langle a, b\rangle, C\rangle =\\
\quad\quad\quad \text{cmulset}\ \langle a, \text{cmulset}\ \langle b, C\rangle\rangle\\
\quad\quad)\\
\quad) \wedge\\
\\
\\
\quad (\mathbf{1} \in \mathbb{R} \wedge\\
\quad\ \text{caddset}\ \langle \text{cmulset}\ \langle i, i\rangle, \mathbf{1}\rangle = \mathbf{0}
\end{array}
$$

$) \wedge$

$(\forall a.\ a \in \mathbb{C} \longrightarrow \text{caddset}\ \langle a, \mathbf{0}\rangle = a$
$) \wedge$

$i \in \mathbb{C}$
$) \wedge$

$($
$(\forall a.\ a \in \mathbb{C} \longrightarrow$
$(\exists x{\in}\mathbb{C}.\ \text{caddset}\ \langle a, x\rangle = \mathbf{0}$
$)$
$) \wedge$

$(\ \forall a.\ a \in \mathbb{C} \wedge a \neq \mathbf{0} \longrightarrow$
$(\ \exists x{\in}\mathbb{C}.\ \text{cmulset}\ \langle a, x\rangle = \mathbf{1}$
$)$
$) \wedge$

$(\ \forall a.\ a \in \mathbb{C} \longrightarrow$
$\text{cmulset}\ \langle a, \mathbf{1}\rangle = a$
$)$
$) \wedge$

$($
$(\ \forall a\ b.\ a \in \mathbb{R} \wedge b \in \mathbb{R} \longrightarrow$
$\text{caddset}\ \langle a, b\rangle \in \mathbb{R}$
$) \wedge$

$(\ \forall a\ b.\ a \in \mathbb{R} \wedge b \in \mathbb{R} \longrightarrow$
$\text{cmulset}\ \langle a, b\rangle \in \mathbb{R}$
$)$
$) \wedge$

$(\ \forall a.\ a \in \mathbb{R} \longrightarrow$
$(\ \exists x{\in}\mathbb{R}.\ \text{caddset}\ \langle a, x\rangle = \mathbf{0}$
$)$
$) \wedge$

$(\ \forall a.\ a \in \mathbb{R} \wedge a \neq \mathbf{0} \longrightarrow$
$(\ \exists x{\in}\mathbb{R}.\ \text{cmulset}\ \langle a, x\rangle = \mathbf{1}$
$)$
$)$
by blast
then show MMIsar0($\mathbb{R}$,$\mathbb{C}$,$\mathbf{1}$,$\mathbf{0}$,i,CplxAdd(R,A),CplxMul(R,A,M),
StrictVersion(CplxROrder(R,A,r))) unfolding MMIsar0˙def by blast
qed

end

82 Metamath sampler

theory Metamath˙Sampler imports Metamath˙Interface MMI˙Complex˙ZF˙2

begin

The theorems translated from Metamath reside in the MMI˙Complex˙ZF, MMI˙Complex˙ZF˙1 and MMI˙Complex˙ZF˙2 theories. The proofs of these theorems are very verbose and for this reason the theories are not shown in the proof document or the FormaMath.org site. This theory file contains some examples of theorems translated from Metamath and formulated in the complex0 context. This serves two purposes: to give an overview of the material covered in the translated theorems and to provide examples of how to take a translated theorem (proven in the MMIsar0 context) and transfer it to the complex0 context. The typical procedure for moving a theorem from MMIsar0 to complex0 is as follows: First we define certain aliases that map names defined in the complex0 to their corresponding names in the MMIsar0 context. This makes it easy to copy and paste the statement of the theorem as displayed with ProofGeneral. Then we run the Isabelle from ProofGeneral up to the theorem we want to move. When the theorem is verified ProofGeneral displays the statement in the raw set theory notation, stripped from any notation defined in the MMIsar0 locale. This is what we copy to the proof in the complex0 locale. After that we just can write "then have ?thesis by simp" and the simplifier translates the raw set theory notation to the one used in complex0.

82.1 Extended reals and order

In this sectin we import a couple of theorems about the extended real line and the linear order on it.

Metamath uses the set of real numbers extended with $+\infty$ and $-\infty$. The $+\infty$ and $-\infty$ symbols are defined quite arbitrarily as $\mathbb{C}$ and $\{\mathbb{C}\}$, respectively. The next lemma that corresponds to Metamath's renfdisj states that $+\infty$ and $-\infty$ are not elements of $\mathbb{R}$.

lemma (in complex0) renfdisj: shows $\mathbb{R} \cap -+\infty,-\infty'' = 0$
proof -
  let real = $\mathbb{R}$
  let complex = $\mathbb{C}$
  let one = $\mathbf{1}$
  let zero = $\mathbf{0}$
  let iunit = i
  let caddset = CplxAdd(R,A)

```
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar˙valid by simp
  then have real ∩ –complex, –complex˝˝ = 0
    by (rule MMIsar0.MMI˙renfdisj)
  thus ℝ ∩ –+∞,–∞˝ = 0 by simp
qed
```

The order relation used most often in Metamath is defined on the set of complex reals extended with $+\infty$ and $-\infty$. The next lemma allows to use Metamath's xrltso that states that the ¡ relations is a strict linear order on the extended set.

```
lemma (in complex0) xrltso: shows ¡ Orders ℝ*
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar˙valid by simp
  then have
    (lessrrel ∩ real × real ∪
    –⟨–complex˝, complex⟩˝ ∪ real × –complex˝ ∪
      —complex˝˝ × real) Orders (real ∪ –complex, –complex˝˝)
    by (rule MMIsar0.MMI˙xrltso)
  moreover have lessrrel ∩ real × real = lessrrel
    using cplx˙strict˙ord˙on˙cplx˙reals by auto
  ultimately show ¡ Orders ℝ* by simp
qed
```

Metamath defines the usual $<$ and $\leq$ ordering relations for the extended real line, including $+\infty$ and $-\infty$.

```
lemma (in complex0) xrrebndt: assumes A1: x ∈ ℝ*
  shows x ∈ ℝ ⟷ ( −∞ ¡ x ∧ x ¡ +∞ )
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
```

```
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar˙valid by simp
  then have x ∈ ℝ ∪ –ℂ, –ℂ˝˝ ⟶
    x ∈ ℝ ⟷ ⟨–ℂ˝, x⟩ ∈ lessrrel ∩ ℝ × ℝ ∪ –⟨–ℂ˝, ℂ⟩˝ ∪
    ℝ × –ℂ˝ ∪ —ℂ˝˝ × ℝ ∧
    ⟨x, ℂ⟩ ∈ lessrrel ∩ ℝ × ℝ ∪ –⟨–ℂ˝, ℂ⟩˝ ∪
    ℝ × –ℂ˝ ∪ —ℂ˝˝ × ℝ
    by (rule MMIsar0.MMI˙xrrebndt)
  then have x ∈ ℝ* ⟶ ( x ∈ ℝ ⟷ ( −∞ ¡ x ∧ x ¡ +∞ ) )
    by simp
  with A1 show thesis by simp
qed
```

A quite involved inequality.

```
lemma (in complex0) lt2mul2divt:
  assumes A1: a ∈ ℝ  b ∈ ℝ  c ∈ ℝ  d ∈ ℝ and
  A2: 𝟬 ¡ b  𝟬 ¡ d
  shows a·b ¡ c·d ⟷ a/d ¡ c/b
proof -
  let real = ℝ
  let complex = ℂ
  let one = 𝟭
  let zero = 𝟬
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar˙valid by simp
  then have
    (a ∈ real ∧ b ∈ real) ∧
    (c ∈ real ∧ d ∈ real) ∧
    ⟨zero, b⟩ ∈ lessrrel ∩ real × real ∪
    –⟨–complex˝, complex⟩˝ ∪ real × –complex˝ ∪ —complex˝˝ × real ∧
    ⟨zero, d⟩ ∈ lessrrel ∩ real × real ∪
    –⟨–complex˝, complex⟩˝ ∪ real × –complex˝ ∪ —complex˝˝ × real ⟶
    ⟨cmulset  ⟨a, b⟩, cmulset  ⟨c, d⟩⟩ ∈
    lessrrel ∩ real × real ∪ –⟨–complex˝, complex⟩˝ ∪
    real × –complex˝ ∪ —complex˝˝ × real ⟷
    ⟨⋃–x ∈ complex . cmulset  ⟨d, x⟩ = a˝,
    ⋃–x ∈ complex . cmulset  ⟨b, x⟩ = c˝⟩ ∈
    lessrrel ∩ real × real ∪ –⟨–complex˝, complex⟩˝ ∪
    real × –complex˝ ∪ —complex˝˝ × real
    by (rule MMIsar0.MMI˙lt2mul2divt)
  with A1 A2 show thesis by simp
qed
```

A real number is smaller than its half iff it is positive.

```
lemma (in complex0) halfpos: assumes A1: a ∈ ℝ
  shows 𝟬 ¡ a ⟷ a/𝟮 ¡ a
proof -
  let real = ℝ
  let complex = ℂ
  let one = 𝟭
  let zero = 𝟬
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  from A1 have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    and a ∈ real
    using MMIsar˙valid by auto
  then have
    ⟨zero, a⟩ ∈
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪ —complex″″ × real ⟷
    ⟨⋃–x ∈ complex . cmulset  ⟨caddset  ⟨one, one⟩, x⟩ = a″, a⟩ ∈
    lessrrel ∩ real × real ∪
    –⟨–complex″, complex⟩″ ∪ real × –complex″ ∪ —complex″″ × real
    by (rule MMIsar0.MMI˙halfpos)
  then show thesis by simp
qed
```

One more inequality.

```
lemma (in complex0) ledivp1t:
  assumes A1:  a ∈ ℝ   b ∈ ℝ and
  A2: 𝟬 ≤ a  𝟬 ≤ b
  shows (a/(b + 𝟭))·b ≤ a
proof -
  let real = ℝ
  let complex = ℂ
  let one = 𝟭
  let zero = 𝟬
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar˙valid by simp
  then have
    (a ∈ real ∧ ⟨a, zero⟩ ∉
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪ —complex″″ × real) ∧
    b ∈ real ∧ ⟨b, zero⟩ ∉ lessrrel ∩ real × real ∪
```

```
    –⟨–complex″, complex⟩″ ∪ real × –complex″ ∪
    —complex″″ × real ⟶
    ⟨a,cmulset⟨⋃–x ∈ complex . cmulset⟨caddset⟨b, one⟩, x⟩ = a″, b⟩⟩ ∉
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪ —complex″″ × real
    by (rule MMIsar0.MMI˙ledivp1t)
  with A1 A2 show thesis by simp
qed
```

82.2 Natural real numbers

In standard mathematics natural numbers are treated as a subset of real numbers. From the set theory point of view however those are quite different objects. In this section we talk about "real natural" numbers i.e. the conterpart of natural numbers that is a subset of the reals.

Two ways of saying that there are no natural numbers between n and $n+1$.

```
lemma (in complex0) no˙nats˙between:
  assumes A1: n ∈ ℕ  k ∈ ℕ
  shows
  n≤k ⟷ n ¡ k+𝟏
  n ¡ k ⟷ n + 𝟏 ≤ k
proof -
  let real = ℝ
  let complex = ℂ
  let one = 𝟏
  let zero = 𝟎
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have I: MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar˙valid by simp
  then have
    n ∈ ⋂–N ∈ Pow(real) . one ∈ N ∧
    (∀n. n ∈ N ⟶ caddset  ⟨n, one⟩ ∈ N)″ ∧
    k ∈ ⋂–N ∈ Pow(real) . one ∈ N ∧
    (∀n. n ∈ N ⟶ caddset  ⟨n, one⟩ ∈ N)″ ⟶
    ⟨k, n⟩ ∉
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪ real × –complex″ ∪
    —complex″″ × real ⟷
    ⟨n, caddset  ⟨k, one⟩⟩ ∈
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪ real × –complex″ ∪
    —complex″″ × real by (rule MMIsar0.MMI˙nnleltp1t)
  then have n ∈ ℕ ∧ k ∈ ℕ ⟶ n ≤ k ⟷ n ¡ k + 𝟏
    by simp
  with A1 show n≤k ⟷ n ¡ k+𝟏 by simp
  from I have
```

```
    n ∈ ⋂–N ∈ Pow(real) . one ∈ N ∧
    (∀n. n ∈ N ⟶ caddset ⟨n, one⟩ ∈ N)″ ∧
    k ∈ ⋂–N ∈ Pow(real) . one ∈ N ∧
    (∀n. n ∈ N ⟶ caddset ⟨n, one⟩ ∈ N)″ ⟶
    ⟨n, k⟩ ∈
    lessrrel ∩ real × real ∪
    –⟨–complex″, complex⟩″ ∪ real × –complex″ ∪
    —complex″″ × real ⟷ ⟨k, caddset ⟨n, one⟩⟩ ∉
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪ real × –complex″ ∪
    —complex″″ × real by (rule MMIsar0.MMI˙nnltp1let)
  then have n ∈ ℕ ∧ k ∈ ℕ ⟶ n ¡ k ⟷ n + 𝟏 ≤ k
    by simp
  with A1 show n ¡ k ⟷ n + 𝟏 ≤ k by simp
qed
```

Metamath has some very complicated and general version of induction on (complex) natural numbers that I can't even understand. As an exercise I derived a more standard version that is imported to the complex0 context below.

```
lemma (in complex0) cplx˙nat˙ind: assumes A1: ψ(𝟏) and
  A2: ∀k ∈ ℕ. ψ(k) ⟶ ψ(k+𝟏) and
  A3: n ∈ ℕ
  shows ψ(n)
proof -
  let real = ℝ
  let complex = ℂ
  let one = 𝟏
  let zero = 𝟎
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have I: MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar˙valid by simp
  moreover from A1 A2 A3 have
    ψ(one)
    ∀k∈⋂–N ∈ Pow(real) . one ∈ N ∧
    (∀n. n ∈ N ⟶ caddset ⟨n, one⟩ ∈ N)″.
    ψ(k) ⟶ ψ(caddset ⟨k, one⟩)
    n ∈ ⋂–N ∈ Pow(real) . one ∈ N ∧
    (∀n. n ∈ N ⟶ caddset ⟨n, one⟩ ∈ N)″
    by auto
  ultimately show ψ(n) by (rule MMIsar0.nnind1)
qed
```

Some simple arithmetics.

```
lemma (in complex0) arith: shows
  𝟐 + 𝟐 = 𝟒
```

```
  2·2 = 4
  3·2 = 6
  3·3 = 9
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have I: MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar˙valid by simp
  then have
    caddset ⟨caddset ⟨one, one⟩, caddset ⟨one, one⟩⟩ =
    caddset ⟨caddset ⟨caddset ⟨one, one⟩, one⟩, one⟩
    by (rule MMIsar0.MMI˙2p2e4)
  thus 2 + 2 = 4 by simp
  from I have
    cmulset⟨caddset⟨one, one⟩, caddset⟨one, one⟩⟩ =
    caddset⟨caddset⟨caddset⟨one, one⟩, one⟩, one⟩
    by (rule MMIsar0.MMI˙2t2e4)
  thus 2·2 = 4 by simp
  from I have
    cmulset⟨caddset⟨caddset⟨one, one⟩, one⟩, caddset⟨one, one⟩⟩ =
    caddset ⟨caddset⟨caddset⟨caddset⟨caddset
    ⟨one, one⟩, one⟩, one⟩, one⟩, one⟩
    by (rule MMIsar0.MMI˙3t2e6)
  thus 3·2 = 6 by simp
  from I have cmulset
    ⟨caddset⟨caddset⟨one, one⟩, one⟩,
    caddset⟨caddset⟨one, one⟩, one⟩⟩ =
    caddset⟨caddset⟨caddset ⟨caddset
    ⟨caddset⟨caddset⟨caddset⟨caddset⟨one, one⟩, one⟩, one⟩, one⟩,
    one⟩, one⟩, one⟩, one⟩
    by (rule MMIsar0.MMI˙3t3e9)
  thus 3·3 = 9 by simp
qed
```

82.3 Infimum and supremum in real numbers

Real numbers form a complete ordered field. Here we import a couple of Metamath theorems about supremu and infimum.

If a set S has a smallest element, then the infimum of S belongs to it.

```
lemma (in complex0) lbinfmcl: assumes A1: S ⊆ ℝ and
  A2: ∃x∈S. ∀y∈S. x ≤ y
```

```
  shows  Infim(S,ℝ,¡) ∈ S
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have I: MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar˙valid by simp
  then have
    S ⊆ real ∧ (∃x∈S. ∀y∈S. ⟨y, x⟩ ∉
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪ —complex″″ × real) ⟶
    Sup(S, real,
    converse(lessrrel ∩ real × real ∪
    –⟨–complex″, complex⟩″ ∪ real × –complex″ ∪
    —complex″″ × real)) ∈ S
    by (rule MMIsar0.MMI˙lbinfmcl)
  then have
    S ⊆ℝ ∧ ( ∃x∈S. ∀y∈S. x ≤ y) ⟶
    Sup(S,ℝ,converse(¡)) ∈ S by simp
  with A1 A2 show thesis using Infim˙def by simp
qed
```

Supremum of any subset of reals that is bounded above is real.

```
lemma (in complex0) sup˙is˙real:
  assumes A ⊆ ℝ  and A ≠ 0 and ∃x∈ℝ. ∀y∈A. y ≤ x
  shows Sup(A,ℝ,¡) ∈ ℝ
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar˙valid by simp
  then have
    A ⊆ real ∧ A ≠ 0 ∧ (∃x∈real.  ∀y∈A. ⟨x, y⟩ ∉
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪ —complex″″ × real) ⟶
    Sup(A, real,
```

```
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪ —complex″″ × real) ∈ real
    by (rule MMIsar0.MMI˙suprcl)
  with assms show thesis by simp
qed
```

If a real number is smaller that the supremum of A, then we can find an element of A greater than it.

```
lemma (in complex0) suprlub:
  assumes A ⊆ℝ and A ≠ 0 and ∃x∈ℝ. ∀y∈A. y ≤ x
  and B ∈ ℝ  and B ¡ Sup(A,ℝ,¡)
  shows ∃z∈A. B ¡ z
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar˙valid by simp
  then have (A ⊆ real ∧ A ≠ 0 ∧ (∃x∈real. ∀y∈A. ⟨x, y⟩ ∉
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪
    —complex″″ × real)) ∧ B ∈ real ∧ ⟨B, Sup(A, real,
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪
    —complex″″ × real)⟩ ∈ lessrrel ∩ real × real ∪
    –⟨–complex″, complex⟩″ ∪ real × –complex″ ∪
    —complex″″ × real ⟶
    (∃z∈A. ⟨B, z⟩ ∈ lessrrel ∩ real × real ∪
    –⟨–complex″, complex⟩″ ∪ real × –complex″ ∪
    —complex″″ × real)
    by (rule MMIsar0.MMI˙suprlub)
  with assms show thesis by simp
qed
```

Something a bit more interesting: infimum of a set that is bounded below is real and equal to the minus supremum of the set flipped around zero.

```
lemma (in complex0) infmsup:
  assumes A ⊆ ℝ and A ≠ 0 and ∃x∈ℝ. ∀y∈A. x ≤ y
  shows
  Infim(A,ℝ,¡) ∈ ℝ
  Infim(A,ℝ,¡) = ( -Sup(–z ∈ ℝ. (-z) ∈ A ″,ℝ,¡) )
proof -
  let real = ℝ
```

```
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have I: MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar˙valid by simp
  then have
    A ⊆ real ∧ A ≠ 0 ∧ (∃x∈real. ∀y∈A. ⟨y, x⟩ ∉
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪
    —complex″″ × real) ⟶ Sup(A, real, converse
    (lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪
    —complex″″ × real)) =
    ⋃–x ∈ complex . caddset
    ⟨Sup(–z ∈ real . ⋃–x ∈ complex . caddset⟨z, x⟩ = zero″ ∈ A″, real,
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪ —complex″″ × real), x⟩ = zero″
    by (rule MMIsar0.MMI˙infmsup)
  then have A ⊆ℝ ∧ ¬(A = 0) ∧ ( ∃x∈ℝ. ∀y∈A. x ≤ y) ⟶
    Sup(A,ℝ,converse(¡)) = ( -Sup(–z ∈ ℝ. (-z) ∈ A ″,ℝ,¡) )
    by simp
  with assms show
    Infim(A,ℝ,¡) = ( -Sup(–z ∈ ℝ. (-z) ∈ A ″,ℝ,¡) )
    using Infim˙def by simp
  from I have
    A ⊆ real ∧ A ≠ 0 ∧ (∃x∈real. ∀y∈A. ⟨y, x⟩ ∉
    lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪
    —complex″″ × real) ⟶ Sup(A, real, converse
    (lessrrel ∩ real × real ∪ –⟨–complex″, complex⟩″ ∪
    real × –complex″ ∪ —complex″″ × real)) ∈ real
    by (rule MMIsar0.MMI˙infmrcl)
  with assms show Infim(A,ℝ,¡) ∈ ℝ
    using Infim˙def by simp
qed

end
```

References

[1] N. A'Campo. A natural construction for the real numbers. 2003.

[2] R. D. Arthan. The Eudoxus Real Numbers. 2004.

[3] R. Street at al. The Efficient Real Numbers. 2003.

[4] Strecker G.E. Herrlich H. When is $\mathbb{N}$ lindelöf? Comment. Math. Univ. Carolinae, 1997.

[5] I. L. Reilly and M. K. Vamanamurthy. Some topological anti-properties. Illinois J. Math., 24:382–389, 1980.