Ordinals and cardinals in HOL

Andrei Popescu & Dmitriy Traytel

Contents

Abstract

We develop a basic theory of ordinals and cardinals in Isabelle/HOL, up to the point where some cardinality facts relevant for the "working mathematician" become available. Unlike in set theory, here we do not have at hand canonical notions of ordinal and cardinal. Therefore, here an ordinal is merely a well-order relation and a cardinal is an ordinal minim w.r.t. order embedding on its field.

1 Introduction

In set theory (under formalizations such as Zermelo-Fraenkel or Von Neumann-Bernays-Gödel), an *ordinal* is a special kind of well-order, namely one whose strict version is the restriction of the membership relation to a set. In particular, the field of a set-theoretic ordinal is a transitive set, and the non-strict version of an ordinal relation is set inclusion. Set-theoretic ordinals enjoy the nice properties of membership on transitive sets, while at the same time forming a complete class of representatives for well-orders (since any well-order turns out isomorphic to an ordinal). Moreover, the class of ordinals is itself transitive and well-ordered by membership as the strict relation and inclusion as the non-strict relation. Also knowing that any set can be well-ordered (in the presence of the axiom of choice), one then defines the *cardinal* of a set to be the smallest ordinal isomorphic to a well-order on that set. This makes the class of cardinals a complete set of representatives for the intuitive notion of set cardinality.[1] The ability to produce *canonical well-orders* from the membership relation (having the aforementioned convenient properties) allows for a harmonious development of the theory of cardinals in set-theoretic settings. Non-trivial cardinality results, such as A being equipollent to $A \times A$ for any infinite A, follow rather quickly within this theory.

However, a canonical notion of well-order is *not* available in HOL. Here, one has to do with well-order "as is", but otherwise has all the necessary infrastructure (including Hilbert choice) to "climb" well-orders recursively and to well-oder arbitrary sets.

The current work, formalized in Isabelle/HOL, develops the basic theory of ordinals and cardinals up to the point where there are inferred a collection of non-trivial cardinality facts useful for the "working mathematician", among which:

[1]The "intuitive" cardinality of a set A is the class of all sets equipollent to A, i.e., being in bijection with A.

- Given any two sets (on any two given types)[2], one is injectable in the other.
- If at least one of two sets is infinite, then their sum and their Cartesian product are equipollent to the larger of the two.
- The set of lists (and also the set of finite sets) with element from an infinite set is equipollent with that set.

Our development emulates the standard one from set-theory, but keeps everything *up to order isomorphism.* An (HOL) ordinal is merely a well-order. An *ordinal embedding* is an injective and order-compatible function which maps its source into an initial segment (i.e., order filter) of its target. Now, a *cardinal* (called in this work a *cardinal order*) is an ordinal minim w.r.t. the existence of embeddings among all well-orders on its field. After showing the existence of cardinals on any given set, we define the cardinal of a set A, denoted $|A|$, to be *some* cardinal order on A. This concept is unique only up to order isomorphism (denoted $=o$), but meets its purpose: any two sets A and B (laying at potentially distinct types) are in bijection if and only if $|A| =o |B|$. Moreover, we also show that numeric cardinals assigned to finite sets[3] are *conservatively extended* by our general (order-theoretic) notion of cardinal. We study the interaction of cardinals with standard set-theoretic constructions such as powersets, products, sums and lists. These constructions are shown to preserve the "cardinal identity" $=o$ and also to be monotonic w.r.t. $\leq o$, the ordinal embedding relation. By studying the interaction between these constructions, infinite sets and cardinals, we obtain the aforementioned results for "working mathematicians".

For this development, we did not follow closely any particular textbook, and in fact are not aware of such basic theory of cardinals previously developed in HOL.[4] On the other hand, the set-theoretic versions of the facts proved here are folklore in set theory, and can be found, e.g., in the textbook [1]. Beyond taking care of some locality aspects concerning the spreading of our concepts throughout types, we have not departed much from the techniques used in set theory for establishing these facts – for instance, in the proof of one of our major theorems, *Card-order-Times-same-infinite* from Section 8.4,[5] we have essentially applied the technique described, e.g., in the proof of theorem 1.5.11 from [1], page 47.

Here is the structure of the rest of this document.

[2]Recall that, in HOL, a set on a type α is modeled, just like a predicate, as a function from α to bool.

[3]Numeric cardinals of finite sets are already formalized in Isabelle/HOL.

[4]After writing this formalization, we became aware of Paul Taylor's membership-free development of the theory of ordinals [2].

[5]This theorem states that, for any infinite cardinal r on a set A, $|A \times A|$ is not larger than r.

The next three sections, 2-4, develop some mathematical prerequisites. In Section 2, a large collection of simple facts about injections, bijections, inverses, (in)finite sets and numeric cardinals are proved, making life easier for later, when proving less trivial facts. Section 3 introduces upper and lower bounds operators for order-like relations and studies their basic properties. Section 4 states some useful variations of well-founded recursion and induction principles.

Then come the major sections, 5-8. Section 5 defines and studies, in the context of a well-order relation, the notions of minimum (of a set), maximum (of two elements), supremum, successor (of a set), and order filter (i.e., initial segment, i.e., downward-closed set). Section 6 defines and studies (well-order) embeddings, strict embeddings, isomorphisms, and compatible functions. Section 7 deals with various constructions on well-orders, and with the relations $\leq o$, $<o$ and $=o$ of well-order embedding, strict embedding, and isomorphism, respectively. Section 8 defines and studies cardinal order relations, the cardinal of a set, the connection of cardinals with set-theoretic constructs, the canonical cardinal of natural numbers and finite cardinals, the successor of a cardinal, as well as regular cardinals. (The latter play a crucial role in the development of a new (co)datatype package in HOL.)

Finally, section 9 provides an abstraction of the previous developments on cardinals, to provide a simpler user interface to cardinals, which in most of the cases allows to forget that cardinals are represented by orders and use them through defined arithmetic operators.

More informal details are provided at the beginning of each section, and also at the beginning of some of the subsections.

References

[1] M. Holz, K. Steffens, and E. Weitz. *Introduction to Cardinal Arithmetic.* Birkhäuser, 1999.

[2] Paul Taylor. Intuitionistic sets and ordinals. *J. Symb. Log.*, 61(3):705–744, 1996.

2 More on injections, bijections and inverses

theory *Fun2* **imports** *~~/src/HOL/Library/Infinite-Set*
begin

This section proves more facts (additional to those in *Fun.thy*, *Hilbert-Choice.thy* *Finite-Set.thy* and *Infinite-Set.thy*), mainly concerning injections, bijections, inverses and (numeric) cardinals of finite sets.

2.1 Purely functional properties

lemma *UNIV-Times*:
$(\mathit{UNIV} :: ('a * 'b)\ \mathit{set}) = (\mathit{UNIV} :: 'a\ \mathit{set}) <*> (\mathit{UNIV} :: 'b\ \mathit{set})$
by *simp*

lemma *UNIV-Plus*:
$(\mathit{UNIV} :: ('a + 'b)\ \mathit{set}) = (\mathit{UNIV} :: 'a\ \mathit{set}) <+> (\mathit{UNIV} :: 'b\ \mathit{set})$
by *simp*

lemma *bij-bij-betw*: $\mathit{bij}\ f = \mathit{bij\text{-}betw}\ f\ \mathit{UNIV}\ \mathit{UNIV}$
unfolding *bij-betw-def bij-def surj-def* **by** *auto*

lemma *bij-betw-empty1*:
assumes $\mathit{bij\text{-}betw}\ f\ \{\}\ A$
shows $A = \{\}$
using *assms* **unfolding** *bij-betw-def* **by** *blast*

lemma *bij-betw-empty2*:
assumes $\mathit{bij\text{-}betw}\ f\ A\ \{\}$
shows $A = \{\}$
using *assms* **unfolding** *bij-betw-def* **by** *blast*

lemma *inj-on-imp-bij-betw*:
$\mathit{inj\text{-}on}\ f\ A \Longrightarrow \mathit{bij\text{-}betw}\ f\ A\ (f\ `\ A)$
unfolding *bij-betw-def inj-on-def* **by** *blast*

lemma *inj-on-cong*[*fundef-cong*]:
$(\bigwedge a.\ a : A \Longrightarrow f\ a = g\ a) \Longrightarrow \mathit{inj\text{-}on}\ f\ A = \mathit{inj\text{-}on}\ g\ A$
unfolding *inj-on-def* **by** *auto*

lemma *inj-on-strict-subset*:
$[\![\mathit{inj\text{-}on}\ f\ B;\ A < B]\!] \Longrightarrow f`A < f`B$
unfolding *inj-on-def* **unfolding** *image-def* **by** *blast*

lemma *bij-betw-cong*[*fundef-cong*]:
$(\bigwedge a.\ a \in A \Longrightarrow f\ a = g\ a) \Longrightarrow \mathit{bij\text{-}betw}\ f\ A\ A' = \mathit{bij\text{-}betw}\ g\ A\ A'$
unfolding *bij-betw-def inj-on-def* **by** *force*

lemma *bij-betw-id*: $\mathit{bij\text{-}betw}\ \mathit{id}\ A\ A$
unfolding *bij-betw-def id-def* **by** *auto*

lemma *bij-betw-id-iff*:
$(A = B) = (\textit{bij-betw id } A\ B)$
by(*auto simp add*: *bij-betw-def*)

lemma *bij-betw-byWitness*:
assumes *LEFT*: $\forall\, a \in A.\ f'(f\ a) = a$ **and**
RIGHT: $\forall\, a' \in A'.\ f(f'\ a') = a'$ **and**
IM1: $f\ `\ A \leq A'$ **and** *IM2*: $f'\ `\ A' \leq A$
shows *bij-betw* $f\ A\ A'$
using *assms*
proof(*unfold bij-betw-def inj-on-def*, *auto*)
fix $a\ b$ **assume** $*$: $a \in A\ b \in A$ **and** $**$: $f\ a = f\ b$
have $a = f'(f\ a) \wedge b = f'(f\ b)$ **using** $*$ *LEFT* **by** *auto*
with $**$ **show** $a = b$ **by** *simp*
next
fix a' **assume** $*$: $a' \in A'$
hence $f'\ a' \in A$ **using** *IM2* **by** *auto*
moreover
have $a' = f(f'\ a')$ **using** $*$ *RIGHT* **by** *auto*
ultimately show $a' \in f\ `\ A$ **by** *blast*
qed

lemma *Int-inj-on*: $[\![\textit{inj-on } f\ A;\ \textit{inj-on } f\ B]\!] \Longrightarrow \textit{inj-on } f\ (A\ \textit{Int}\ B)$
unfolding *inj-on-def* **by** *blast*

lemma *INTER-inj-on*:
$[\![I \neq \{\};\ \bigwedge\, i.\ i \in I \Longrightarrow \textit{inj-on } f\ (A\ i)]\!] \Longrightarrow \textit{inj-on } f\ (\bigcap\ i \in I.\ A\ i)$
unfolding *inj-on-def* **by** *blast*

lemma *Inter-inj-on*:
$[\![S \neq \{\};\ \bigwedge\, A.\ A \in S \Longrightarrow \textit{inj-on } f\ A]\!] \Longrightarrow \textit{inj-on } f\ (\textit{Inter}\ S)$
unfolding *inj-on-def* **by** *blast*

lemma *UNION-inj-on*:
assumes *CH*: $\bigwedge\, i\ j.\ [\![i \in I;\ j \in I]\!] \Longrightarrow A\ i \leq A\ j \vee A\ j \leq A\ i$ **and**
INJ: $\bigwedge\, i.\ i \in I \Longrightarrow \textit{inj-on } f\ (A\ i)$
shows *inj-on* $f\ (\bigcup\ i \in I.\ A\ i)$
proof(*unfold inj-on-def UNION-eq*, *auto*)
fix $i\ j\ x\ y$
assume $*$: $i \in I\ j \in I$ **and** $**$: $x \in A\ i\ y \in A\ j$
and $***$: $f\ x = f\ y$
show $x = y$

proof−
{**assume** $A\ i \leq A\ j$
with $**$ **have** $x \in A\ j$ **by** *auto*
with *INJ* $*$ $**$ $***$ **have** *?thesis*
by(*auto simp add*: *inj-on-def*)
}
moreover
{**assume** $A\ j \leq A\ i$
with $**$ **have** $y \in A\ i$ **by** *auto*
with *INJ* $*$ $**$ $***$ **have** *?thesis*
by(*auto simp add*: *inj-on-def*)
}
ultimately show *?thesis* **using** *CH* $*$ **by** *blast*
qed
qed

lemma *bij-betw-comp*:
$[\![$*bij-betw f A A'*; *bij-betw f' A' A''*$]\!] \Longrightarrow$ *bij-betw* $(f'\ o\ f)\ A\ A''$
using *comp-inj-on*[*of f A f'*]
by(*auto simp add*: *bij-betw-def comp-def*)

lemma *UNION-bij-betw*:
assumes *CH*: $\bigwedge i\ j.\ [\![i \in I;\ j \in I]\!] \Longrightarrow A\ i \leq A\ j \vee A\ j \leq A\ i$ **and**
BIJ: $\bigwedge i.\ i \in I \Longrightarrow$ *bij-betw f* $(A\ i)\ (A'\ i)$
shows *bij-betw f* $(\bigcup\ i \in I.\ A\ i)\ (\bigcup\ i \in I.\ A'\ i)$
proof(*unfold bij-betw-def*, *auto simp add*: *image-def*)
have $\bigwedge i.\ i \in I \Longrightarrow$ *inj-on f* $(A\ i)$
using *BIJ bij-betw-def*[*of f*] **by** *auto*
thus *inj-on f* $(\bigcup\ i \in I.\ A\ i)$
using *CH UNION-inj-on*[*of I A f*] **by** *auto*
next
fix $i\ x$
assume $*$: $i \in I\ x \in A\ i$
hence $f\ x \in A'\ i$ **using** *BIJ bij-betw-def*[*of f*] **by** *auto*
thus $\exists j \in I.\ f\ x \in A'\ j$ **using** $*$ **by** *blast*
next
fix $i\ x'$
assume $*$: $i \in I\ x' \in A'\ i$
hence $\exists x \in A\ i.\ x' = f\ x$ **using** *BIJ bij-betw-def*[*of f*] **by** *blast*
thus $\exists j \in I.\ \exists x \in A\ j.\ x' = f\ x$
using $*$ **by** *blast*
qed

lemma *Disj-Un-bij-betw*:
assumes *DISJ*: A *Int* $B = \{\}$ **and** *DISJ'*: A' *Int* $B' = \{\}$ **and**
B1: *bij-betw f A A'* **and** *B2*: *bij-betw f B B'*

shows *bij-betw f* $(A \cup B)$ $(A' \cup B')$
proof−
  have *1*: *inj-on f A* $\wedge$ *inj-on f B*
  using *B1 B2* **by** (*auto simp add*: *bij-betw-def*)
  have *2*: $f`A = A' \wedge f`B = B'$
  using *B1 B2* **by** (*auto simp add*: *bij-betw-def*)
  hence $f`(A - B)$ *Int* $f`(B - A) = \{\}$
  using *DISJ DISJ'* **by** *blast*
  hence *inj-on f* $(A \cup B)$
  using *1* **by** (*auto simp add*: *inj-on-Un*)

  moreover
  have $f`(A \cup B) = A' \cup B'$
  using *2* **by** *auto*
  ultimately show *?thesis*
  unfolding *bij-betw-def* **by** *auto*
qed

corollary *notIn-Un-bij-betw*:
assumes *NIN*: $b \notin A$ **and** *NIN'*: $f\ b \notin A'$ **and**
    BIJ: *bij-betw f A A'*
shows *bij-betw f* $(A \cup \{b\})$ $(A' \cup \{f\ b\})$
proof−
  have *bij-betw f* $\{b\}$ $\{f\ b\}$
  unfolding *bij-betw-def inj-on-def* **by** *auto*
  with *assms* **show** *?thesis*
  using *Disj-Un-bij-betw*[*of A* $\{b\}$ *A'* $\{f\ b\}$ *f*] **by** *blast*
qed

lemma *bij-betw-subset*:
assumes *BIJ*: *bij-betw f A A'* **and**
    SUB: $B \le A$ **and** *IM*: $f\ `\ B = B'$
shows *bij-betw f B B'*
using *assms*
by(*unfold bij-betw-def inj-on-def*, *auto simp add*: *inj-on-def*)

lemma *notIn-Un-bij-betw2*:
assumes *NIN*: $b \notin A$ **and** *NIN'*: $b' \notin A'$ **and**
    BIJ: *bij-betw f A A'*
shows *bij-betw f* $(A \cup \{b\})$ $(A' \cup \{b'\}) = (f\ b = b')$
proof
  assume $f\ b = b'$
  thus *bij-betw f* $(A \cup \{b\})$ $(A' \cup \{b'\})$
  using *assms notIn-Un-bij-betw*[*of b A f A'*] **by** *auto*
next
  assume $*$: *bij-betw f* $(A \cup \{b\})$ $(A' \cup \{b'\})$

hence $f\ b \in A' \cup \{b'\}$
unfolding *bij-betw-def* **by** *auto*
moreover
{assume $f\ b \in A'$
then obtain *b1* **where** *1*: $b1 \in A$ **and** *2*: $f\ b1 = f\ b$ **using** *BIJ*
by (*auto simp add*: *bij-betw-def*)
hence $b = b1$ **using** $*$
by (*auto simp add*: *bij-betw-def inj-on-def*)
with *1 NIN* **have** *False* **by** *auto*
}
ultimately show $f\ b = b'$ **by** *blast*
qed

lemma *notIn-Un-bij-betw3*:
assumes *NIN*: $b \notin A$ **and** *NIN'*: $f\ b \notin A'$
shows *bij-betw f A A'* $=$ *bij-betw f* $(A \cup \{b\})$ $(A' \cup \{f\ b\})$
proof
assume *bij-betw f A A'*
thus *bij-betw f* $(A \cup \{b\})$ $(A' \cup \{f\ b\})$
using *assms notIn-Un-bij-betw*[*of b A f A'*] **by** *auto*
next
assume $*$: *bij-betw f* $(A \cup \{b\})$ $(A' \cup \{f\ b\})$
have f ‘ $A = A'$
proof(*auto*)
fix *a* **assume** $**$: $a \in A$
hence $f\ a \in A' \cup \{f\ b\}$ **using** $*$
by (*auto simp add*: *bij-betw-def*)
moreover
{assume $f\ a = f\ b$
hence $a = b$ **using** $*$ $**$
by(*auto simp add*: *bij-betw-def inj-on-def*)
with *NIN* $**$ **have** *False* **by** *auto*
}
ultimately show $f\ a \in A'$ **by** *blast*
next
fix a' **assume** $**$: $a' \in A'$
hence $a' \in f$‘$(A \cup \{b\})$
using $*$ **by** (*auto simp add*: *bij-betw-def*)
then obtain *a* **where** *1*: $a \in A \cup \{b\} \wedge f\ a = a'$ **by** *blast*
moreover
{assume $a = b$ **with** *1* $**$ *NIN'* **have** *False* **by** *blast*
}
ultimately have $a \in A$ **by** *blast*
with *1* **show** $a' \in f$ ‘ A **by** *auto*
qed
thus *bij-betw f A A'* **using** $*$ *bij-betw-subset*[*of f* $A \cup \{b\}$ - A] **by** *auto*
qed

lemma *bij-betw-diff-singl*:
assumes *BIJ*: *bij-betw f A A'* **and** *IN*: $a \in A$
shows *bij-betw f* $(A - \{a\})$ $(A' - \{f\ a\})$
proof−
  let $?B = A - \{a\}$ **let** $?B' = A' - \{f\ a\}$
  have $f\ a \in A'$ **using** *IN BIJ* **unfolding** *bij-betw-def* **by** *auto*
  hence $a \notin ?B \wedge f\ a \notin ?B' \wedge A = ?B \cup \{a\} \wedge A' = ?B' \cup \{f\ a\}$
  using *IN* **by** *blast*
  thus *?thesis* **using** *notIn-Un-bij-betw3*[*of a ?B f ?B'*] *BIJ* **by** *auto*
qed

lemma *comp-inj-on2*:
inj-on f A $\Longrightarrow$ *inj-on f'* $(f\ `\ A)$ = *inj-on* $(f'\ o\ f)$ *A*
by(*auto simp add*: *comp-inj-on inj-on-def*)

lemma *comp-inj-on3*:
inj-on $(f'\ o\ f)$ *A* $\Longrightarrow$ *inj-on f A*
by(*auto simp add*: *comp-inj-on inj-on-def*)

lemma *comp-bij-betw2*:
bij-betw f A A' $\Longrightarrow$ *bij-betw f' A' A''* = *bij-betw* $(f'\ o\ f)$ *A A''*
by(*auto simp add*: *bij-betw-def inj-on-def*)

lemma *comp-bij-betw3*:
assumes *BIJ*: *bij-betw f' A' A''* **and** *IM*: $f\ `\ A \leq A'$
shows *bij-betw f A A'* = *bij-betw* $(f'\ o\ f)$ *A A''*
using *assms*
proof(*auto simp add*: *bij-betw-comp*)
  assume $*$: *bij-betw* $(f' \circ f)$ *A A''*
  thus *bij-betw f A A'*
  using *IM*
  proof(*auto simp add*: *bij-betw-def*)
    assume *inj-on* $(f' \circ f)$ *A*
    thus *inj-on f A* **using** *comp-inj-on3* **by** *blast*
  next
    fix a' **assume** $**$: $a' \in A'$
    hence $f'\ a' \in A''$ **using** *BIJ* **unfolding** *bij-betw-def* **by** *auto*
    then obtain *a* **where** *1*: $a \in A \wedge f'(f\ a) = f'\ a'$ **using** $*$
    unfolding *bij-betw-def* **by** *force*
    hence $f\ a \in A'$ **using** *IM* **by** *auto*
    hence $f\ a = a'$ **using** *BIJ* $**$ *1* **unfolding** *bij-betw-def inj-on-def* **by** *auto*
    thus $a' \in f\ `\ A$ **using** *1* **by** *auto*
  qed
qed

```
lemma bij-betw-ball:
assumes BIJ: bij-betw f A B
shows (∀ b ∈ B. phi b) = (∀ a ∈ A. phi(f a))
using assms unfolding bij-betw-def inj-on-def by blast
```

2.2 Properties involving finite and infinite sets

```
lemma inj-on-finite:
assumes INJ: inj-on f A and SUB: f ‘ A ≤ B and
        FIN: finite B
shows finite A
proof−
  have finite B ⟹ (∀ (A::'a set) f. inj-on f A ∧ f ‘ A ≤ B ⟶ finite A)
  proof(erule finite-induct, auto)
    fix x B and A::'a set and f
    assume 1: finite B and 2: x ∉ B and
           3: inj-on f A and 4: f ‘ A ⊆ insert x B and
           IH: ∀ (A::'a set). (∃ g. inj-on g A ∧ g ‘ A ⊆ B) ⟶ finite A
    show finite A
    proof(cases f ‘ A ≤ B)
      assume Case1: f ‘ A ≤ B
      thus ?thesis using 3 IH by blast
    next
      assume Case2: ¬ f ‘ A ≤ B
      then obtain a where 5: a ∈ A ∧ f a = x using 4 by blast
      let ?A' = A − {a}
      have inj-on f ?A' using 3 subset-inj-on[of f A ?A'] by blast
      moreover
      have f ‘ ?A' ≤ B
      proof(auto)
        fix a' assume ∗: a' ∈ A and f a' ∉ B
        hence f a' = x using 4 by auto
        thus a' = a using ∗ 5 3 unfolding inj-on-def by auto
      qed
      ultimately have finite ?A' using IH by blast
      thus ?thesis using finite-insert by auto
    qed
  qed
  thus ?thesis using assms by blast
qed
```

```
lemma bij-betw-finite:
assumes bij-betw f A B
shows finite A = finite B
using assms unfolding bij-betw-def
using inj-on-finite[of f A B] by auto
```

```
lemma infinite-imp-bij-betw:
assumes INF: infinite A
shows ∃h. bij-betw h A (A − {a})
proof(cases a ∈ A)
  assume Case1: a ∉ A  hence A − {a} = A by blast
  thus ?thesis using bij-betw-id[of A] by auto
next
  assume Case2: a ∈ A
  have infinite (A − {a}) using INF infinite-remove by auto
  with infinite-iff-countable-subset[of A − {a}] obtain f::nat ⇒ 'a
  where 1: inj f and 2: f ` UNIV ≤ A − {a} by blast
  obtain g where g-def: g = (λ n. if n = 0 then a else f (Suc n)) by blast
  obtain A' where A'-def: A' = g ` UNIV by blast
  have temp: ∀ y. f y ≠ a using 2 by blast
  have 3: inj-on g UNIV ∧ g ` UNIV ≤ A ∧ a ∈ g ` UNIV
  proof(auto simp add: Case2 g-def, unfold inj-on-def, intro ballI impI,
        case-tac x = 0, auto simp add: 2)
    fix y  assume a = (if y = 0 then a else f (Suc y))
    thus y = 0 using temp by (case-tac y = 0, auto)
  next
    fix x y
    assume f (Suc x) = (if y = 0 then a else f (Suc y))
    thus x = y using 1 temp unfolding inj-on-def by (case-tac y = 0, auto)
  next
    fix n show f (Suc n) ∈ A using 2 by blast
  qed
  hence 4: bij-betw g UNIV A' ∧ a ∈ A' ∧ A' ≤ A
  using inj-on-imp-bij-betw[of g] unfolding A'-def by auto
  hence 5: bij-betw (inv g) A' UNIV
  by (auto simp add: bij-betw-inv-into)

  obtain n where g n = a using 3 by auto
  hence 6: bij-betw g (UNIV − {n}) (A' − {a})
  using 4 bij-betw-diff-singl[of g] by blast

  obtain v where v-def: v = (λ m. if m < n then m else Suc m) by blast
  have 7: bij-betw v UNIV (UNIV − {n})
  proof(unfold bij-betw-def inj-on-def, intro conjI, clarify)
    fix m1 m2 assume v m1 = v m2
    thus m1 = m2
    by(case-tac m1 < n, case-tac m2 < n,
       auto simp add: inj-on-def v-def, case-tac m2 < n, auto)
  next
    show v ` UNIV = UNIV − {n}
    proof(auto simp add: v-def)
      fix m assume *: m ≠ n and **: m ∉ Suc ` {m'. ¬ m' < n}
      {assume n ≤ m with * have 71: Suc n ≤ m by auto
```

```
      then obtain m' where 72: m = Suc m' using Suc-le-D by auto
      with 71 have n ≤ m' by auto
      with 72 ** have False by auto
     }
     thus m < n by force
   qed
 qed

 obtain h' where h'-def: h' = g o v o (inv g) by blast
 hence 8: bij-betw h' A' (A' − {a}) using 5 7 6
 by (auto simp add: bij-betw-comp)

 obtain h where h-def: h = (λ b. if b ∈ A' then h' b else b) by blast
 have ∀ b ∈ A'. h b = h' b unfolding h-def by auto
 hence bij-betw h  A' (A' − {a}) using 8 bij-betw-cong[of A' h] by auto
 moreover
 {have ∀ b ∈ A − A'. h b = b unfolding h-def by auto
  hence bij-betw h  (A − A') (A − A')
  using bij-betw-cong[of A − A' h id] bij-betw-id[of A − A'] by auto
 }
 moreover
 have (A' Int (A − A') = {} ∧ A' ∪ (A − A') = A) ∧
       ((A' − {a}) Int (A − A') = {} ∧ (A' − {a}) ∪ (A − A') = A − {a})
 using 4 by blast
 ultimately have bij-betw h A (A − {a})
 using Disj-Un-bij-betw[of A' A − A' A' − {a} A − A' h] by auto
 thus ?thesis by blast
qed


lemma infinite-imp-bij-betw2:
assumes INF: infinite A
shows ∃ h. bij-betw h A (A ∪ {a})
proof(cases a ∈ A)
 assume Case1: a ∈ A  hence A ∪ {a} = A by blast
 thus ?thesis using bij-betw-id[of A] by auto
next
 let ?A' = A ∪ {a}
 assume Case2: a ∉ A hence A = ?A' − {a} by blast
 moreover have infinite ?A' using INF by auto
 ultimately obtain f where bij-betw f ?A' A
 using infinite-imp-bij-betw[of ?A' a] by auto
 hence bij-betw(inv-into ?A' f) A ?A' using bij-betw-inv-into by blast
 thus ?thesis by auto
qed


lemma bij-betw-imp-card:
assumes FIN: finite A and BIJ: bij-betw f A B
```

shows *card A* = *card B*
proof−
  have *finite A* $\Longrightarrow \forall B$. *bij-betw f A B* $\longrightarrow$ *card A* = *card B*
  proof (*erule finite.induct*, *auto*)
    fix *B* **assume** *bij-betw f* {} *B*
    thus *card B* = *0* **using** *bij-betw-empty1 card-empty* **by** *blast*
  next
    fix *A a B′*
    assume ∗: *finite A* **and** ∗∗: *bij-betw f* (*insert a A*) *B′* **and**
        IH: $\forall B$. *bij-betw f A B* $\longrightarrow$ *card A* = *card B*
    show *card* (*insert a A*) = *card B′*
    proof(*cases a* $\in$ *A*)
      assume *a* $\in$ *A* **hence** *1*: *insert a A* = *A* **by** *auto*
      hence *bij-betw f A B′* **using** ∗∗ **by** *auto*
      thus *?thesis* **using** *IH* ∗ *1* **by** *auto*
    next
      assume ∗∗∗: *a* $\notin$ *A*
      hence *2*: *card*(*insert a A*) = *card A* + *1* **using** ∗ **by** *auto*
      obtain *b* **and** *B* **where** *b-def*: *b* = *f a* **and** *B-def*: *B* = *B′* − {*b*} **by** *blast*
      have *3*: *b* $\in$ *B′* **using** ∗∗ **unfolding** *bij-betw-def b-def* **by** *auto*
      have (*insert a A*) − {*a*} = *A* **using** ∗∗∗ **by** *auto*
      hence *bij-betw f A B* **unfolding** *B-def b-def*
      using ∗∗ *bij-betw-diff-singl*[*of f insert a A B′ a*] **by** *auto*
      hence *5*: *card A* = *card B* **using** ∗ *IH* **by** *auto*
      have *B′* = *insert b B* $\wedge$ *b* $\notin$ *B* **unfolding** *B-def* **using** *insert-Diff 3* **by** *blast*
      moreover have *finite B* **unfolding** *B-def*
      using *bij-betw-finite*[*of f - B′*] *finite-subset*[*of B B′*] ∗ ∗∗ **by** *auto*
      ultimately have *card B′* = *card B* + *1* **by** *auto*

      with *2 5* **show** *?thesis* **by** *auto*
    qed
  qed
  thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *bij-betw-iff-card*:
assumes *FIN*: *finite A* **and** *FIN′*: *finite B*
shows *BIJ*: ($\exists f$. *bij-betw f A B*) = (*card A* = *card B*)
using *assms*
proof(*auto simp add*: *bij-betw-imp-card*)
  assume ∗: *card A* = *card B*
  obtain *f* **where** *bij-betw f A* {*0* ..< *card A*}
  using *FIN ex-bij-betw-finite-nat* **by** *blast*
  moreover obtain *g* **where** *bij-betw g* {*0* ..< *card B*} *B*
  using *FIN′ ex-bij-betw-nat-finite* **by** *blast*
  ultimately have *bij-betw* (*g o f*) *A B*
  using ∗ **by** (*auto simp add*: *bij-betw-comp*)
  thus ($\exists f$. *bij-betw f A B*) **by** *blast*

qed

lemma *inj-on-iff-card*:
assumes *FIN*: *finite A* **and** *FIN'*: *finite B*
shows $(\exists f.$ *inj-on f A* $\wedge f$ ‘ $A \leq B) = ($*card A* $\leq$ *card B*$)$
using *assms*
proof(*auto simp add*: *card-inj-on-le*)
  assume $*$: *card A* $\leq$ *card B*
  obtain *f* **where** *1*: *inj-on f A* **and** *2*: *f* ‘ $A = \{0 ..<$ *card A*$\}$
  using *FIN ex-bij-betw-finite-nat* **unfolding** *bij-betw-def* **by** *force*
  moreover obtain *g* **where** *inj-on g* $\{0 ..<$ *card B*$\}$ **and** *3*: *g* ‘ $\{0 ..<$ *card B*$\}$ $= B$
  using *FIN' ex-bij-betw-nat-finite* **unfolding** *bij-betw-def* **by** *force*
  ultimately have *inj-on g* (*f* ‘ *A*) **using** *subset-inj-on*[*of g - f* ‘ *A*] $*$ **by** *force*
  hence *inj-on* (*g o f*) *A* **using** *1 comp-inj-on* **by** *blast*
  moreover
  {have $\{0 ..<$ *card A*$\} \leq \{0 ..<$ *card B*$\}$ **using** $*$ **by** *force*
   with *2* **have** *f* ‘ $A \ \leq \{0 ..<$ *card B*$\}$ **by** *blast*
   hence (*g o f*) ‘ $A \leq B$ **unfolding** *comp-def* **using** *3* **by** *force*
  }
  ultimately show $(\exists f.$ *inj-on f A* $\wedge f$ ‘ $A \leq B)$ **by** *blast*
qed

lemma *inj-on-image-Pow*:
assumes *inj-on f A*
shows *inj-on* (*image f*) (*Pow A*)
unfolding *Pow-def inj-on-def* **proof**(*clarsimp*)
  fix *X Y* **assume** $*$: $X \leq A$ **and** $**$: $Y \leq A$ **and**
          $***$: *f* ‘ $X =$ *f* ‘ Y
  show $X = Y$
  proof(*auto*)
    fix *x* **assume** $****$: $x \in X$
    with $***$ **obtain** *y* **where** $y \in Y \wedge f\,x = f\,y$ **by** *blast*
    with $****$ $*$ $**$ *assms* **show** $x \in Y$
    unfolding *inj-on-def* **by** *auto*
  next
    fix *y* **assume** $****$: $y \in Y$
    with $***$ **obtain** *x* **where** $x \in X \wedge f\,x = f\,y$ **by** *force*
    with $****$ $*$ $**$ *assms* **show** $y \in X$
    unfolding *inj-on-def* **by** *auto*
  qed
qed

lemma *image-Pow-mono*:
assumes *f* ‘ $A \leq B$
shows (*image f*) ‘ (*Pow A*) $\leq$ *Pow B*

using *assms* **by** *blast*

lemma *image-Pow-surjective*:
assumes f ‘ $A = B$
shows $(image\ f)$ ‘ $(Pow\ A) = Pow\ B$
using *assms* **unfolding** *Pow-def* **proof**(*auto*)
  fix Y **assume** $*$: $Y \le f$ ‘ A
  obtain X **where** *X-def*: $X = \{x \in A.\ f\ x \in Y\}$ **by** *blast*
  have f ‘ $X = Y \wedge X \le A$ **unfolding** *X-def* **using** $*$ **by** *auto*
  thus $Y \in (image\ f)$ ‘ $\{X.\ X \le A\}$ **by** *blast*
qed

lemma *bij-betw-image-Pow*:
assumes *bij-betw f A B*
shows *bij-betw* $(image\ f)$ $(Pow\ A)$ $(Pow\ B)$
using *assms* **unfolding** *bij-betw-def*
by (*auto simp add*: *inj-on-image-Pow image-Pow-surjective*)

2.3 Properties involving Hilbert choice

lemma *bij-betw-inv-into-left*:
assumes *BIJ*: *bij-betw f A A′* **and** *IN*: $a \in A$
shows $(inv\text{-}into\ A\ f)\ (f\ a) = a$
proof(*unfold inv-into-def*)
  let $?phi = (\lambda\ b.\ b \in A \wedge f\ b = f\ a)$
  have $?phi\ a$ **using** *IN* **by** *auto*
  moreover
  have $\bigwedge\ b.\ ?phi\ b \Longrightarrow b = a$
  using *assms* **by** (*auto simp add*: *bij-betw-def inj-on-def*)
  ultimately
  show $(SOME\ a.\ ?phi\ a) = a$
  by (*auto simp add*: *some-equality*)
qed

lemma *bij-betw-inv-into-right*:
assumes *BIJ*: *bij-betw f A A′* **and** *IN*: $a' \in A'$
shows $f(inv\text{-}into\ A\ f\ a') = a'$
proof−
  let $?f' = (inv\text{-}into\ A\ f)$
  have *1*: *bij-betw* $?f'\ A'\ A$
  using *BIJ* **by** (*auto simp add*: *bij-betw-inv-into*)
  hence *2*: $?f'\ a' \in A$
  using *IN* **by** (*auto simp add*: *bij-betw-def*)
  hence $?f'(f(?f'\ a')) = ?f'\ a'$
  using *BIJ* **by** (*auto simp add*: *bij-betw-inv-into-left*)
  moreover

have $f(?f'\ a') \in A'$
using *BIJ 2* **by** (*auto simp add*: *bij-betw-def*)
ultimately show $f(?f'\ a') = a'$
using *IN 1* **by** (*auto simp add*: *bij-betw-def inj-on-def*)
qed

lemma *bij-betw-inv-into-LEFT*:
assumes *BIJ*: *bij-betw f A A'* **and** *SUB*: $B \leq A$
shows $(inv\text{-}into\ A\ f)`(f\ `\ B) = B$
using *assms*
proof(*auto simp add*: *bij-betw-inv-into-left*)
let $?f' = (inv\text{-}into\ A\ f)$
fix a **assume** $*$: $a \in B$
hence $a \in A$ **using** *SUB* **by** *auto*
hence $a = ?f'\ (f\ a)$
using *BIJ* **by** (*auto simp add*: *bij-betw-inv-into-left*)
thus $a \in ?f'\ `\ (f\ `\ B)$ **using** $*$ **by** *blast*
qed

lemma *bij-betw-inv-into-RIGHT*:
assumes *BIJ*: *bij-betw f A A'* **and** *SUB*: $B' \leq A'$
shows $f\ `((inv\text{-}into\ A\ f)`B') = B'$
using *assms*
proof(*auto simp add*: *bij-betw-inv-into-right*)
let $?f' = (inv\text{-}into\ A\ f)$
fix a' **assume** $*$: $a' \in B'$
hence $a' \in A'$ **using** *SUB* **by** *auto*
hence $a' = f\ (?f'\ a')$
using *BIJ* **by** (*auto simp add*: *bij-betw-inv-into-right*)
thus $a' \in f\ `\ (?f'\ `\ B')$ **using** $*$ **by** *blast*
qed

lemma *bij-betw-inv-into-LEFT-RIGHT*:
assumes *BIJ*: *bij-betw f A A'* **and** *SUB*: $B \leq A$ **and**
IM: $f\ `\ B = B'$
shows $(inv\text{-}into\ A\ f)\ `\ B' = B$
proof−
have $(inv\text{-}into\ A\ f)`(f\ `\ B) = B$
using *assms bij-betw-inv-into-LEFT*[*of f A A' B*] **by** *auto*
thus *?thesis* **using** *IM* **by** *auto*
qed

lemma *bij-betw-inv-into-RIGHT-LEFT*:
assumes *BIJ*: *bij-betw f A A'* **and** *SUB*: $B' \leq A'$ **and**
IM: $(inv\text{-}into\ A\ f)\ `\ B' = B$

```
shows f ‘ B = B′
proof−
  have f‘((inv-into A f)‘ B′) = B′
  using assms bij-betw-inv-into-RIGHT[of f A A′ B′] by auto
  thus ?thesis using IM by auto
qed


lemma bij-betw-inv-into-subset:
assumes BIJ: bij-betw f A A′ and
        SUB: B ≤ A and IM: f ‘ B = B′
shows bij-betw (inv-into A f) B′ B
proof−
  let ?f′ = (inv-into A f)
  have ?f′ ‘ B′ = B  using assms
  by (auto simp add: bij-betw-inv-into-LEFT-RIGHT)
  moreover
  {have bij-betw ?f′ A′ A
   using BIJ by (auto simp add: bij-betw-inv-into)
   hence inj-on ?f′ A′ unfolding bij-betw-def by auto
   moreover have B′ ≤ A′
   using SUB IM BIJ by (auto simp add: bij-betw-def)
   ultimately have inj-on ?f′ B′ using SUB
   by (auto simp add: subset-inj-on)
  }
  ultimately show ?thesis
  unfolding bij-betw-def by blast
qed


lemma bij-betw-inv-into-twice:
assumes bij-betw f A A′
shows ∀ a ∈ A. inv-into A′ (inv-into A f) a = f a
proof
  let ?f′ = inv-into A f   let ?f′′ = inv-into A′ ?f′
  have 1: bij-betw ?f′ A′ A using assms
  by (auto simp add: bij-betw-inv-into)
  fix a assume ∗: a ∈ A
  then obtain a′ where 2: a′ ∈ A′ and 3: ?f′ a′ = a
  using 1 unfolding bij-betw-def by force
  hence ?f′′ a = a′
  using ∗ 1 3 by (auto simp add: bij-betw-inv-into-left)
  moreover have f a = a′ using assms 2 3
  by (auto simp add: bij-betw-inv-into-right)
  ultimately show ?f′′ a = f a by simp
qed


lemma inj-on-iff-surjective:
```

assumes $A \neq \{\}$
shows $(\exists f.\ \textit{inj-on } f\ A \wedge f\ `\ A \leq A') = (\exists g.\ g\ `\ A' = A)$
proof(*safe*)
  fix f **assume** *INJ*: *inj-on f A* **and** *INCL*: $f\ `\ A \leq A'$
  let $?phi = \lambda a'\ a.\ a \in A \wedge f\ a = a'$ **let** $?csi = \lambda a.\ a \in A$
  let $?g = \lambda a'.\ \textit{if } a' \in f\ `\ A\ \textit{then } (\textit{SOME } a.\ ?phi\ a'\ a)\ \textit{else } (\textit{SOME } a.\ ?csi\ a)$
  have $?g\ `\ A' = A$
  proof
    show $?g\ `\ A' \leq A$
    proof(*clarify*)
      fix a' **assume** $*$: $a' \in A'$
      show $?g\ a' \in A$
      proof(*cases* $a' \in f\ `\ A$)
        assume *Case1*: $a' \in f\ `\ A$
        then obtain a **where** $?phi\ a'\ a$ **by** *blast*
        hence $?phi\ a'\ (\textit{SOME } a.\ ?phi\ a'\ a)$ **using** *someI*[*of* $?phi\ a'\ a$] **by** *blast*
        with *Case1* **show** *?thesis* **by** *auto*
      next
        assume *Case2*: $a' \notin f\ `\ A$
        hence $?csi\ (\textit{SOME } a.\ ?csi\ a)$ **using** *assms someI-ex*[*of ?csi*] **by** *blast*
        with *Case2* **show** *?thesis* **by** *auto*
      qed
    qed
  next
    show $A \leq ?g\ `\ A'$
    proof−
      {**fix** a **assume** $*$: $a \in A$
       let $?b = \textit{SOME } aa.\ ?phi\ (f\ a)\ aa$
       have $?phi\ (f\ a)\ a$ **using** $*$ **by** *auto*
       hence *1*: $?phi\ (f\ a)\ ?b$ **using** *someI*[*of* $?phi(f\ a)\ a$] **by** *blast*
       hence $?g(f\ a) = ?b$ **using** $*$ **by** *auto*
       moreover have $a = ?b$ **using** *1 INJ* $*$ **by** (*auto simp add*: *inj-on-def*)
       ultimately have $?g(f\ a) = a$ **by** *simp*
       with *INCL* $*$ **have** $?g(f\ a) = a \wedge f\ a \in A'$ **by** *auto*
      }
      thus *?thesis* **by** *force*
    qed
  qed
  thus $\exists g.\ g\ `\ A' = A$ **by** *blast*
next
  fix g **let** $?f = \textit{inv-into } A'\ g$
  have *inj-on* $?f\ (g\ `\ A')$
  by (*auto simp add*: *inj-on-inv-into*)
  moreover
  {**fix** a' **assume** $*$: $a' \in A'$
   let $?phi = \lambda\ b'.\ b' \in A' \wedge g\ b' = g\ a'$
   have $?phi\ a'$ **using** $*$ **by** *auto*
   hence $?phi(\textit{SOME } b'.\ ?phi\ b')$ **using** *someI*[*of ?phi*] **by** *blast*
   hence $?f(g\ a') \in A'$ **unfolding** *inv-into-def* **by** *auto*

}
ultimately show $\exists f.\ inj\text{-}on\ f\ (g\ `\ A') \wedge f\ `\ g\ `\ A' \subseteq A'$ **by** *auto*
qed

lemma *UNION-inj-on-Sigma*:
$\exists f.\ (inj\text{-}on\ f\ (\bigcup\ i \in I.\ A\ i) \wedge f\ `\ (\bigcup\ i \in I.\ A\ i) \leq (SIGMA\ i : I.\ A\ i))$
proof
let $?phi = \lambda\ a\ i.\ i \in I \wedge a \in A\ i$
let $?sm = \lambda\ a.\ SOME\ i.\ ?phi\ a\ i$
let $?f = \lambda a.\ (?sm\ a,\ a)$
have $inj\text{-}on\ ?f\ (\bigcup\ i \in I.\ A\ i)$ **unfolding** *inj-on-def* **by** *auto*
moreover
{{**fix** *i a* **assume** $i \in I$ **and** $a \in A\ i$
hence $?sm\ a \in I \wedge a \in A(?sm\ a)$ **using** *someI*[*of ?phi a i*] **by** *auto*
}
hence $?f\ `\ (\bigcup\ i \in I.\ A\ i) \leq (SIGMA\ i : I.\ A\ i)$ **by** *auto*
}
ultimately
show $inj\text{-}on\ ?f\ (\bigcup\ i \in I.\ A\ i) \wedge\ ?f\ `\ (\bigcup\ i \in I.\ A\ i) \leq (SIGMA\ i : I.\ A\ i)$
by *auto*
qed

2.4 Cantor's Paradox

lemma *Cantors-paradox*:
$\neg(\exists f.\ f\ `\ A = Pow\ A)$
proof(*clarify*)
fix *f* **assume** $f\ `\ A = Pow\ A$ **hence** $*$: $Pow\ A \leq f\ `\ A$ **by** *blast*
let $?X = \{a \in A.\ a \notin f\ a\}$
have $?X \in Pow\ A$ **unfolding** *Pow-def* **by** *auto*
with $*$ **obtain** *x* **where** $x \in A \wedge f\ x = ?X$ **by** *blast*
thus *False* **by** *best*
qed

2.5 The Cantor-Bernstein Theorem

lemma *Cantor-Bernstein-aux*:
shows $\exists A'\ h.\ A' \leq A \wedge$
$(\forall a \in A'.\ a \notin g`(B - f\ `\ A')) \wedge$
$(\forall a \in A'.\ h\ a = f\ a) \wedge$
$(\forall a \in A - A'.\ h\ a \in B - (f\ `\ A') \wedge a = g(h\ a))$
proof−
obtain *H* **where** *H-def*: $H = (\lambda\ A'.\ A - (g`(B - (f\ `\ A'))))$ **by** *blast*
have *0*: *mono H* **unfolding** *mono-def H-def* **by** *blast*
then obtain A' **where** *1*: $H\ A' = A'$ **using** *lfp-unfold* **by** *blast*
hence *2*: $A' = A - (g`(B - (f\ `\ A')))$ **unfolding** *H-def* **by** *simp*
hence *3*: $A' \leq A$ **by** *blast*
have *4*: $\forall a \in A'.\ \ a \notin g`(B - f\ `\ A')$
using *2* **by** *blast*

have *5*: $\forall\, a \in A - A'.\ \exists\, b \in B - (f \text{ ` } A').\ a = g\ b$
using *2* **by** *blast*

obtain *h* **where** *h-def*:
$h = (\lambda\ a.\ \mathit{if}\ a \in A'\ \mathit{then}\ f\ a\ \mathit{else}\ (\mathit{SOME}\ b.\ b \in B - (f \text{ ` } A') \wedge a = g\ b))$ **by** *blast*
hence $\forall\, a \in A'.\ h\ a = f\ a$ **by** *auto*
moreover
have $\forall\, a \in A - A'.\ h\ a \in B - (f \text{ ` } A') \wedge a = g(h\ a)$
proof
  fix *a* **assume** $*$: $a \in A - A'$
  let $?phi = \lambda\ b.\ b \in B - (f \text{ ` } A') \wedge a = g\ b$
  have $h\ a = (\mathit{SOME}\ b.\ ?phi\ b)$ **using** *h-def* $*$ **by** *auto*
  moreover have $\exists\, b.\ ?phi\ b$ **using** *5* $*$ **by** *auto*
  ultimately show $?phi\ (h\ a)$ **using** *someI-ex*[*of ?phi*] **by** *auto*
qed
ultimately show *?thesis* **using** *3 4* **by** *blast*
qed

theorem *Cantor-Bernstein*:
assumes *INJ1*: *inj-on f A* **and** *SUB1*: $f \text{ ` } A \leq B$ **and**
    INJ2: *inj-on g B* **and** *SUB2*: $g \text{ ` } B \leq A$
shows $\exists\, h.$ *bij-betw h A B*
proof−
  obtain A' **and** *h* **where** *0*: $A' \leq A$ **and**
  1: $\forall\, a \in A'.\ a \notin g\text{`}(B - f \text{ ` } A')$ **and**
  2: $\forall\, a \in A'.\ h\ a = f\ a$ **and**
  3: $\forall\, a \in A - A'.\ h\ a \in B - (f \text{ ` } A') \wedge a = g(h\ a)$
  using *Cantor-Bernstein-aux*[*of A g B f*] **by** *blast*
  have *inj-on h A*
  proof(*unfold inj-on-def*, *auto*)
    fix *a1 a2*
    assume *4*: $a1 \in A$ **and** *5*: $a2 \in A$ **and** *6*: $h\ a1 = h\ a2$
    show $a1 = a2$
    proof(*cases* $a1 \in A'$)
      assume *Case1*: $a1 \in A'$
      show *?thesis*
      proof(*cases* $a2 \in A'$)
        assume *Case11*: $a2 \in A'$
        hence $f\ a1 = f\ a2$ **using** *Case1 2 6* **by** *auto*
        thus *?thesis* **using** *INJ1 Case1 Case11 0*
        unfolding *inj-on-def* **by** *blast*
      next
        assume *Case12*: $a2 \notin A'$
        hence *False* **using** *3 5 2 6 Case1* **by** *force*
        thus *?thesis* **by** *simp*
      qed
    next
    assume *Case2*: $a1 \notin A'$

```
      show ?thesis
      proof(cases a2 ∈ A′)
        assume Case21: a2 ∈ A′
        hence False using 3 4 2 6 Case2 by auto
        thus ?thesis by simp
      next
        assume Case22: a2 ∉ A′
        hence a1 = g(h a1) ∧ a2 = g(h a2) using Case2 4 5 3 by auto
        thus ?thesis using 6 by simp
      qed
    qed
  qed

  moreover
  have h ‘ A = B
  proof(auto)
    fix a assume a ∈ A
    thus h a ∈ B using SUB1 2 3 by (case-tac a ∈ A′, auto)
  next
    fix b assume ∗: b ∈ B
    show b ∈ h ‘ A
    proof(cases b ∈ f ‘ A′)
      assume Case1: b ∈ f ‘ A′
      then obtain a where a ∈ A′ ∧ b = f a by blast
      thus ?thesis using 2 0 by force
    next
      assume Case2: b ∉ f ‘ A′
      hence g b ∉ A′ using 1 ∗ by auto
      hence 4: g b ∈ A − A′ using ∗ SUB2 by auto
      hence h(g b) ∈ B ∧ g(h(g b)) = g b
      using 3 by auto
      hence h(g b) = b using ∗ INJ2 unfolding inj-on-def by auto
      thus ?thesis using 4 by force
    qed
  qed

  ultimately show ?thesis unfolding bij-betw-def by auto
qed
```

2.6 Other facts

```
lemma Pow-not-empty: Pow A ≠ {}
using Pow-top by blast
```

```
lemma atLeastLessThan-injective:
assumes {0 ..< m::nat} = {0 ..< n}
shows m = n
proof−
```

{assume $m < n$
hence $m \in \{0 ..< n\}$ **by** *auto*
hence $\{0 ..< m\} < \{0 ..< n\}$ **by** *auto*
hence *False* **using** *assms* **by** *blast*
}
moreover
{assume $n < m$
hence $n \in \{0 ..< m\}$ **by** *auto*
hence $\{0 ..< n\} < \{0 ..< m\}$ **by** *auto*
hence *False* **using** *assms* **by** *blast*
}
ultimately show *?thesis* **by** *force*
qed

lemma *atLeastLessThan-injective2*:
bij-betw $f\ \{0 ..< m{::}nat\}\ \{0 ..< n\} \Longrightarrow m = n$
using *finite-atLeastLessThan[of m] finite-atLeastLessThan[of n]*
card-atLeastLessThan[of m] card-atLeastLessThan[of n]
bij-betw-iff-card[of $\{0 ..< m\}\ \{0 ..< n\}$*]* **by** *auto*

lemma *atLeastLessThan-less-eq*:
$(\{0..{<}m\} \leq \{0..{<}n\}) = ((m{::}nat) \leq n)$
unfolding *ivl-subset* **by** *arith*

lemma *atLeastLessThan-less-eq2*:
assumes *inj-on* $f\ \{0..{<}(m{::}nat)\} \land f\ `\ \{0..{<}m\} \leq \{0..{<}n\}$
shows $m \leq n$
using *assms*
using *finite-atLeastLessThan[of m] finite-atLeastLessThan[of n]*
card-atLeastLessThan[of m] card-atLeastLessThan[of n]
card-inj-on-le[of f $\{0 ..< m\}\ \{0 ..< n\}$*]* **by** *auto*

lemma *atLeastLessThan-less-eq3*:
$(\exists f.$ *inj-on* $f\ \{0..{<}(m{::}nat)\} \land f\ `\ \{0..{<}m\} \leq \{0..{<}n\}) = (m \leq n)$
using *atLeastLessThan-less-eq2*
proof(*auto*)
assume $m \leq n$
hence *inj-on id* $\{0..{<}m\} \land id\ `\ \{0..{<}m\} \subseteq \{0..{<}n\}$ **unfolding** *inj-on-def* **by** *force*
thus $\exists f.$ *inj-on* $f\ \{0..{<}m\} \land f\ `\ \{0..{<}m\} \subseteq \{0..{<}n\}$ **by** *blast*
qed

lemma *atLeastLessThan-less*:
$(\{0..{<}m\} < \{0..{<}n\}) = ((m{::}nat) < n)$

proof−
  have $(\{0..<m\} < \{0..<n\}) = (\{0..<m\} \leq \{0..<n\} \wedge \{0..<m\} \mathbin{\tilde{}}= \{0..<n\})$
  using *subset-iff-psubset-eq* **by** *blast*
  also have $\ldots = (m \leq n \wedge m \mathbin{\tilde{}}= n)$
  using *atLeastLessThan-less-eq atLeastLessThan-injective* **by** *blast*
  also have $\ldots = (m < n)$ **by** *auto*
  finally show *?thesis* **.**
qed

end

3 Basics on order-like relations

theory *Order-Relation2*
imports *~~/src/HOL/Library/Order-Relation*
begin

In this section, we develop basic concepts and results pertaining to order-like relations, i.e., to reflexive and/or transitive and/or symmetric and/or total relations. The development is placed on top of the definitions from the theory *Order-Relation*. We also further define upper and lower bounds operators.

type-synonym $'a\ rel = ('a * 'a)\ set$

locale *rel* = **fixes** $r :: 'a\ rel$

The following context encompasses all this section, except for its last subsection. In other words, for the rest of this section except its last subsection, we consider a fixed relation r.

context *rel*
begin

3.1 Auxiliaries

lemma *refl-on-domain*:
$[\![\mathit{refl\text{-}on}\ A\ r;\ (a,b) : r]\!] \Longrightarrow a \in A \wedge b \in A$
by(*auto simp add*: *refl-on-def*)

corollary *well-order-on-domain*:
$[\![\mathit{well\text{-}order\text{-}on}\ A\ r;\ (a,b) \in r]\!] \Longrightarrow a \in A \wedge b \in A$
by(*auto simp add*: *refl-on-domain order-on-defs*)

lemma *well-order-on-Field*:
well-order-on A r $\Longrightarrow$ *A* = *Field r*
by(*auto simp add*: *refl-on-def Field-def order-on-defs*)

lemma *well-order-on-Well-order*:
well-order-on A r $\Longrightarrow$ *A* = *Field r* $\wedge$ *Well-order r*
using *well-order-on-Field* **by** *auto*

lemma *Total-Id-Field*:
assumes *TOT*: *Total r* **and** *NID*: $\neg$ (*r* <= *Id*)
shows *Field r* = *Field*(*r* − *Id*)
using *mono-Field*[*of r* − *Id r*] *Diff-subset*[*of r Id*]
proof(*auto*)
  have *r* $\neq$ {} **using** *NID* **by** *auto*
  then obtain *b* **and** *c* **where** *b* $\neq$ *c* $\wedge$ (*b*,*c*) $\in$ *r* **using** *NID* **by** *auto*
  hence *1*: *b* $\neq$ *c* $\wedge$ {*b*,*c*} $\leq$ *Field r* **unfolding** *Field-def* **by** *auto*

  fix *a* **assume** ∗: *a* $\in$ *Field r*
  obtain *d* **where** *2*: *d* $\in$ *Field r* **and** *3*: *d* $\neq$ *a*
  using ∗ *1* **by** *blast*
  hence (*a*,*d*) $\in$ *r* $\vee$ (*d*,*a*) $\in$ *r* **using** ∗ *TOT*
  by (*auto simp add*: *total-on-def*)
  thus *a* $\in$ *Field*(*r* − *Id*) **using** *3* **unfolding** *Field-def* **by** *blast*
qed

lemma *Total-subset-Id*:
assumes *TOT*: *Total r* **and** *SUB*: *r* $\leq$ *Id*
shows *r* = {} $\vee$ ($\exists$ *a*. *r* = {(*a*,*a*)})
proof−
  {**assume** *r* $\neq$ {}
   then obtain *a b* **where** *1*: (*a*,*b*) $\in$ *r* **by** *auto*
   hence *a* = *b* **using** *SUB* **by** *blast*
   hence *2*: (*a*,*a*) $\in$ *r* **using** *1* **by** *auto*
   {**fix** *c d* **assume** (*c*,*d*) $\in$ *r*
    hence {*a*,*c*,*d*} $\leq$ *Field r* **using** *1* **unfolding** *Field-def* **by** *auto*
    hence ((*a*,*c*) $\in$ *r* $\vee$ (*c*,*a*) $\in$ *r* $\vee$ *a* = *c*) $\wedge$
          ((*a*,*d*) $\in$ *r* $\vee$ (*d*,*a*) $\in$ *r* $\vee$ *a* = *d*)
    using *TOT* **unfolding** *total-on-def* **by** *auto*
    hence *a* = *c* $\wedge$ *a* = *d* **using** *SUB* **by** *blast*
   }
   hence *r* $\leq$ {(*a*,*a*)} **by** *auto*
   with *2* **have** $\exists$ *a*. *r* = {(*a*,*a*)} **by** *blast*
  }
  thus *?thesis* **by** *auto*
qed

lemma *Linear-order-in-diff-Id*:
assumes *LI*: *Linear-order r* **and**
  IN1: $a \in$ *Field r* **and** *IN2*: $b \in$ *Field r*
shows $((a,b) \in r) = ((b,a) \notin r - Id)$
using *assms* **unfolding** *order-on-defs total-on-def antisym-def Id-def refl-on-def*
by *force*

3.2 The upper and lower bounds operators

Here we define upper ("above") and lower ("below") bounds operators. We think of r as a *non-strict* relation. The suffix "S" at the names of some operators indicates that the bounds are strict – e.g., *underS a* is the set of all strict lower bounds of a (w.r.t. r). Capitalization of the first letter in the name reminds that the operator acts on sets, rather than on individual elements.

definition *under*::$'a \Rightarrow 'a$ *set*
where *under* $a \equiv \{b.\ (b,a) \in r\}$

definition *underS*::$'a \Rightarrow 'a$ *set*
where *underS* $a \equiv \{b.\ b \neq a \wedge (b,a) \in r\}$

definition *Under*::$'a$ *set* $\Rightarrow 'a$ *set*
where *Under* $A \equiv \{b \in$ *Field* $r.\ \forall\, a \in A.\ (b,a) \in r\}$

definition *UnderS*::$'a$ *set* $\Rightarrow 'a$ *set*
where *UnderS* $A \equiv \{b \in$ *Field* $r.\ \forall\, a \in A.\ b \neq a \wedge (b,a) \in r\}$

definition *above*::$'a \Rightarrow 'a$ *set*
where *above* $a \equiv \{b.\ (a,b) \in r\}$

definition *aboveS*::$'a \Rightarrow 'a$ *set*
where *aboveS* $a \equiv \{b.\ b \neq a \wedge (a,b) \in r\}$

definition *Above*::$'a$ *set* $\Rightarrow 'a$ *set*
where *Above* $A \equiv \{b \in$ *Field* $r.\ \forall\, a \in A.\ (a,b) \in r\}$

definition *AboveS*::$'a$ *set* $\Rightarrow 'a$ *set*
where *AboveS* $A \equiv \{b \in$ *Field* $r.\ \forall\, a \in A.\ b \neq a \wedge (a,b) \in r\}$

Note: In the definitions of *Above*[S] and *Under*[S], we bounded comprehension by *Field r* in order to properly cover the case of A being empty.

lemma *underS-subset-under*: *underS* $a \leq$ *under* a
by(*auto simp add*: *underS-def under-def*)

lemma *UnderS-subset-Under*: *UnderS* $A \leq$ *Under* A

by(*auto simp add*: *UnderS-def Under-def*)

lemma *aboveS-subset-above*: *aboveS a* $\leq$ *above a*
by(*auto simp add*: *aboveS-def above-def*)

lemma *AboveS-subset-Above*: *AboveS A* $\leq$ *Above A*
by(*auto simp add*: *AboveS-def Above-def*)

lemma *underS-notIn*: *a* $\notin$ *underS a*
by(*auto simp add*: *underS-def*)

lemma *Refl-under-in*: $[\![$*Refl r*; *a* $\in$ *Field r*$]\!] \Longrightarrow$ *a* $\in$ *under a*
by(*auto simp add*: *refl-on-def under-def*)

lemma *UnderS-disjoint*: *A Int* (*UnderS A*) = {}
by(*auto simp add*: *UnderS-def*)

lemma *aboveS-notIn*: *a* $\notin$ *aboveS a*
by(*auto simp add*: *aboveS-def*)

lemma *AboveS-disjoint*: *A Int* (*AboveS A*) = {}
by(*auto simp add*: *AboveS-def*)

lemma *Refl-above-in*: $[\![$*Refl r*; *a* $\in$ *Field r*$]\!] \Longrightarrow$ *a* $\in$ *above a*
by(*auto simp add*: *refl-on-def above-def*)

lemma *in-Above-under*: *a* $\in$ *Field r* $\Longrightarrow$ *a* $\in$ *Above* (*under a*)
by(*auto simp add*: *Above-def under-def*)

lemma *in-Under-above*: *a* $\in$ *Field r* $\Longrightarrow$ *a* $\in$ *Under* (*above a*)
by(*auto simp add*: *Under-def above-def*)

lemma *in-AboveS-underS*: *a* $\in$ *Field r* $\Longrightarrow$ *a* $\in$ *AboveS* (*underS a*)
by(*auto simp add*: *AboveS-def underS-def*)

lemma *in-UnderS-aboveS*: *a* $\in$ *Field r* $\Longrightarrow$ *a* $\in$ *UnderS* (*aboveS a*)
by(*auto simp add*: *UnderS-def aboveS-def*)

lemma *subset-Above-Under*: $B \leq$ *Field* $r \Longrightarrow B \leq$ *Above* (*Under* B)
by(*auto simp add*: *Above-def Under-def*)

lemma *subset-Under-Above*: $B \leq$ *Field* $r \Longrightarrow B \leq$ *Under* (*Above* B)
by(*auto simp add*: *Under-def Above-def*)

lemma *subset-AboveS-UnderS*: $B \leq$ *Field* $r \Longrightarrow B \leq$ *AboveS* (*UnderS* B)
by(*auto simp add*: *AboveS-def UnderS-def*)

lemma *subset-UnderS-AboveS*: $B \leq$ *Field* $r \Longrightarrow B \leq$ *UnderS* (*AboveS* B)
by(*auto simp add*: *UnderS-def AboveS-def*)

lemma *Under-Above-Galois*:
$[\![B \leq$ *Field* r; $C \leq$ *Field* $r]\!] \Longrightarrow (B \leq$ *Above* $C) = (C \leq$ *Under* $B)$
by(*unfold Above-def Under-def*, *blast*)

lemma *UnderS-AboveS-Galois*:
$[\![B \leq$ *Field* r; $C \leq$ *Field* $r]\!] \Longrightarrow (B \leq$ *AboveS* $C) = (C \leq$ *UnderS* $B)$
by(*unfold AboveS-def UnderS-def*, *blast*)

lemma *Refl-under-underS*:
assumes *REFL*: *Refl* r **and** *IN*: $a \in$ *Field* r
shows *under* a = *underS* $a \cup \{a\}$
proof(*unfold under-def underS-def*, *auto*)
  show $(a,a) \in r$ **using** *REFL IN refl-on-def*[*of* - r] **by** *blast*
qed

lemma *Refl-above-aboveS*:
assumes *REFL*: *Refl* r **and** *IN*: $a \in$ *Field* r
shows *above* a = *aboveS* $a \cup \{a\}$
proof(*unfold above-def aboveS-def*, *auto*)
  show $(a,a) \in r$ **using** *REFL IN refl-on-def*[*of* - r] **by** *blast*
qed

lemma *Linear-order-under-aboveS-Field*:
assumes *LIN*: *Linear-order* r **and** *IN*: $a \in$ *Field* r
shows *Field* r = *under* $a \cup$ *aboveS* a
proof(*unfold under-def aboveS-def*, *auto*)
  assume $a \in$ *Field* r $(a,\ a) \notin r$

with *LIN IN order-on-defs*[*of* - *r*] *refl-on-def*[*of* - *r*]
show *False* **by** *auto*
next
fix b **assume** $b \in$ *Field* r $(b,\ a) \notin r$
with *LIN IN order-on-defs*[*of Field r r*] *total-on-def*[*of Field r r*]
have $(a,b) \in r \lor a = b$ **by** *blast*
thus $(a,b) \in r$
using *LIN IN order-on-defs*[*of* - *r*] *refl-on-def*[*of* - *r*] **by** *auto*
next
fix b **assume** $(b,\ a) \in r$
thus $b \in$ *Field* r
using *LIN order-on-defs*[*of* - *r*] *refl-on-def*[*of* - *r*] **by** *blast*
next
fix b **assume** $b \neq a$ $(a,\ b) \in r$
thus $b \in$ *Field* r
using *LIN order-on-defs*[*of Field r r*] *refl-on-def*[*of Field r r*] **by** *blast*
qed

lemma *Linear-order-underS-above-Field*:
assumes *LIN*: *Linear-order* r **and** *IN*: $a \in$ *Field* r
shows *Field* r = *underS* $a \cup$ *above* a
proof(*unfold underS-def above-def*, *auto*)
assume $a \in$ *Field* r $(a,\ a) \notin r$
with *LIN IN order-on-defs*[*of* - *r*] *refl-on-def*[*of* - *r*]
show *False* **by** *auto*
next
fix b **assume** $b \in$ *Field* r $(a,\ b) \notin r$
with *LIN IN order-on-defs*[*of Field r r*] *total-on-def*[*of Field r r*]
have $(b,a) \in r \lor b = a$ **by** *blast*
thus $(b,a) \in r$
using *LIN IN order-on-defs*[*of* - *r*] *refl-on-def*[*of* - *r*] **by** *auto*
next
fix b **assume** $b \neq a$ $(b,\ a) \in r$
thus $b \in$ *Field* r
using *LIN order-on-defs*[*of* - *r*] *refl-on-def*[*of* - *r*] **by** *blast*
next
fix b **assume** $(a,\ b) \in r$
thus $b \in$ *Field* r
using *LIN order-on-defs*[*of Field r r*] *refl-on-def*[*of Field r r*] **by** *blast*
qed

lemma *under-empty*: $a \notin$ *Field* $r \Longrightarrow$ *under* $a = \{\}$
unfolding *Field-def under-def* **by** *auto*

lemma *underS-empty*: $a \notin$ *Field* $r \Longrightarrow$ *underS* $a = \{\}$
unfolding *Field-def underS-def* **by** *auto*

lemma *under-Field*: *under a* $\leq$ *Field r*
by(*unfold under-def Field-def*, *auto*)

lemma *underS-Field*: *underS a* $\leq$ *Field r*
by(*unfold underS-def Field-def*, *auto*)

lemma *underS-Field2*:
a $\in$ *Field r* $\Longrightarrow$ *underS a* $<$ *Field r*
using *assms underS-notIn underS-Field* **by** *blast*

lemma *underS-Field3*:
Field r $\neq \{\}$ $\Longrightarrow$ *underS a* $<$ *Field r*
by(*cases a* $\in$ *Field r*, *simp add*: *underS-Field2*,
auto simp add: *underS-empty*)

lemma *Under-Field*: *Under A* $\leq$ *Field r*
by(*unfold Under-def Field-def*, *auto*)

lemma *UnderS-Field*: *UnderS A* $\leq$ *Field r*
by(*unfold UnderS-def Field-def*, *auto*)

lemma *above-Field*: *above a* $\leq$ *Field r*
by(*unfold above-def Field-def*, *auto*)

lemma *aboveS-Field*: *aboveS a* $\leq$ *Field r*
by(*unfold aboveS-def Field-def*, *auto*)

lemma *Above-Field*: *Above A* $\leq$ *Field r*
by(*unfold Above-def Field-def*, *auto*)

lemma *AboveS-Field*: *AboveS A* $\leq$ *Field r*
by(*unfold AboveS-def Field-def*, *auto*)

lemma *Refl-under-Under*:
assumes *REFL*: *Refl r* **and** *NE*: *A* $\neq \{\}$
shows *Under A* $= (\bigcap$ *a* $\in$ *A*. *under a*)
proof

show *Under A* $\subseteq$ ($\bigcap$ *a* $\in$ *A*. *under a*)
by(*unfold Under-def under-def*, *auto*)
next
show ($\bigcap$ *a* $\in$ *A*. *under a*) $\subseteq$ *Under A*
proof(*auto*)
fix *x*
assume $*$: $\forall$ *xa* $\in$ *A*. *x* $\in$ *under xa*
hence $\forall$ *xa* $\in$ *A*. (*x*,*xa*) $\in$ *r*
by (*simp add*: *under-def*)
moreover
{from *NE* **obtain** *a* **where** *a* $\in$ *A* **by** *blast*
with $*$ **have** *x* $\in$ *under a* **by** *simp*
hence *x* $\in$ *Field r*
using *under-Field*[*of a*] **by** *auto*
}
ultimately show *x* $\in$ *Under A*
unfolding *Under-def* **by** *auto*
qed
qed

lemma *Refl-underS-UnderS*:
assumes *REFL*: *Refl r* **and** *NE*: *A* $\neq$ {}
shows *UnderS A* = ($\bigcap$ *a* $\in$ *A*. *underS a*)
proof
show *UnderS A* $\subseteq$ ($\bigcap$ *a* $\in$ *A*. *underS a*)
by(*unfold UnderS-def underS-def*, *auto*)
next
show ($\bigcap$ *a* $\in$ *A*. *underS a*) $\subseteq$ *UnderS A*
proof(*auto*)
fix *x*
assume $*$: $\forall$ *xa* $\in$ *A*. *x* $\in$ *underS xa*
hence $\forall$ *xa* $\in$ *A*. *x* $\neq$ *xa* $\wedge$ (*x*,*xa*) $\in$ *r*
by (*auto simp add*: *underS-def*)
moreover
{from *NE* **obtain** *a* **where** *a* $\in$ *A* **by** *blast*
with $*$ **have** *x* $\in$ *underS a* **by** *simp*
hence *x* $\in$ *Field r*
using *underS-Field*[*of a*] **by** *auto*
}
ultimately show *x* $\in$ *UnderS A*
unfolding *UnderS-def* **by** *auto*
qed
qed

lemma *Refl-above-Above*:
assumes *REFL*: *Refl r* **and** *NE*: *A* $\neq$ {}
shows *Above A* = ($\bigcap$ *a* $\in$ *A*. *above a*)

proof
  show *Above A* $\subseteq$ ($\bigcap$ $a \in A$. *above a*)
  by(*unfold Above-def above-def*, *auto*)
next
  show ($\bigcap$ $a \in A$. *above a*) $\subseteq$ *Above A*
  proof(*auto*)
    fix x
    assume $*$: $\forall$ $xa \in A$. $x \in$ *above xa*
    hence $\forall$ $xa \in A$. $(xa,x) \in r$
    by (*simp add*: *above-def*)
    moreover
    {from *NE* **obtain** a **where** $a \in A$ **by** *blast*
     with $*$ **have** $x \in$ *above a* **by** *simp*
     hence $x \in$ *Field r*
     using *above-Field*[*of a*] **by** *auto*
    }
    ultimately show $x \in$ *Above A*
    unfolding *Above-def* **by** *auto*
  qed
qed

lemma *Refl-aboveS-AboveS*:
assumes *REFL*: *Refl r* **and** *NE*: $A \neq \{\}$
shows *AboveS A* = ($\bigcap$ $a \in A$. *aboveS a*)
proof
  show *AboveS A* $\subseteq$ ($\bigcap$ $a \in A$. *aboveS a*)
  by(*unfold AboveS-def aboveS-def*, *auto*)
next
  show ($\bigcap$ $a \in A$. *aboveS a*) $\subseteq$ *AboveS A*
  proof(*auto*)
    fix x
    assume $*$: $\forall$ $xa \in A$. $x \in$ *aboveS xa*
    hence $\forall$ $xa \in A$. $xa \neq x \wedge (xa,x) \in r$
    by (*auto simp add*: *aboveS-def*)
    moreover
    {from *NE* **obtain** a **where** $a \in A$ **by** *blast*
     with $*$ **have** $x \in$ *aboveS a* **by** *simp*
     hence $x \in$ *Field r*
     using *aboveS-Field*[*of a*] **by** *auto*
    }
    ultimately show $x \in$ *AboveS A*
    unfolding *AboveS-def* **by** *auto*
  qed
qed

lemma *under-Under-singl*: *under a* = *Under* $\{a\}$
by(*unfold Under-def under-def*, *auto simp add*: *Field-def*)

lemma *underS-UnderS-singl*: *underS a* = *UnderS* $\{a\}$
by(*unfold UnderS-def underS-def*, *auto simp add*: *Field-def*)

lemma *above-Above-singl*: *above a* = *Above* $\{a\}$
by(*unfold Above-def above-def*, *auto simp add*: *Field-def*)

lemma *aboveS-AboveS-singl*: *aboveS a* = *AboveS* $\{a\}$
by(*unfold AboveS-def aboveS-def*, *auto simp add*: *Field-def*)

lemma *Under-decr*: $A \leq B \Longrightarrow$ *Under B* $\leq$ *Under A*
by(*unfold Under-def*, *auto*)

lemma *UnderS-decr*: $A \leq B \Longrightarrow$ *UnderS B* $\leq$ *UnderS A*
by(*unfold UnderS-def*, *auto*)

lemma *Above-decr*: $A \leq B \Longrightarrow$ *Above B* $\leq$ *Above A*
by(*unfold Above-def*, *auto*)

lemma *AboveS-decr*: $A \leq B \Longrightarrow$ *AboveS B* $\leq$ *AboveS A*
by(*unfold AboveS-def*, *auto*)

lemma *under-incr*:
assumes *TRANS*: *trans r* **and** *REL*: $(a,b) \in r$
shows *under a* $\leq$ *under b*
proof(*unfold under-def*, *auto*)
  fix *x* **assume** $(x,a) \in r$
  with *REL TRANS trans-def*[*of r*]
  show $(x,b) \in r$ **by** *blast*
qed

lemma *under-incl-iff*:
assumes *TRANS*: *trans r* **and** *REFL*: *Refl r* **and** *IN*: $a \in$ *Field r*
shows (*under a* $\leq$ *under b*) = $((a,b) \in r)$
proof
  assume $(a,b) \in r$
  thus *under a* $\leq$ *under b* **using** *TRANS*
  by (*auto simp add*: *under-incr*)
next
  assume *under a* $\leq$ *under b*

```
  moreover
  have a ∈ under a using REFL IN
  by (auto simp add: Refl-under-in)
  ultimately show (a,b) ∈ r
  by (auto simp add: under-def)
qed


lemma underS-incr:
assumes TRANS: trans r and ANTISYM: antisym r and
        REL: (a,b) ∈ r
shows underS a ≤ underS b
proof(unfold underS-def, auto)
  assume *: b ≠ a and **: (b,a) ∈ r
  with ANTISYM antisym-def[of r] REL
  show False by auto
next
  fix x assume x ≠ a (x,a) ∈ r
  with REL TRANS trans-def[of r]
  show (x,b) ∈ r by blast
qed


lemma underS-incl-iff:
assumes LO: Linear-order r and
        INa: a ∈ Field r and INb: b ∈ Field r
shows (underS a ≤ underS b) = ((a,b) ∈ r)
proof
  assume (a,b) ∈ r
  thus underS a ≤ underS b using LO
  by (auto simp add: order-on-defs underS-incr)
next
  assume *: underS a ≤ underS b
  {assume a = b
   hence (a,b) ∈ r using assms
   by (auto simp add: order-on-defs refl-on-def)
  }
  moreover
  {assume a ≠ b ∧ (b,a) ∈ r
   hence b ∈ underS a unfolding underS-def by auto
   hence b ∈ underS b using * by auto
   hence False by (auto simp add: underS-notIn)
  }
  ultimately
  show (a,b) ∈ r using assms
  order-on-defs[of Field r r] total-on-def[of Field r r] by blast
qed
```

```
lemma above-decr:
assumes TRANS: trans r and REL: (a,b) ∈ r
shows above b ≤ above a
proof(unfold above-def, auto)
  fix x assume (b,x) ∈ r
  with REL TRANS trans-def[of r]
  show (a,x) ∈ r by blast
qed


lemma aboveS-decr:
assumes TRANS: trans r and ANTISYM: antisym r and
        REL: (a,b) ∈ r
shows aboveS b ≤ aboveS a
proof(unfold aboveS-def, auto)
  assume *: a ≠ b and **: (b,a) ∈ r
  with ANTISYM antisym-def[of r] REL
  show False by auto
next
  fix x assume x ≠ b (b,x) ∈ r
  with REL TRANS trans-def[of r]
  show (a,x) ∈ r by blast
qed


lemma under-trans:
assumes TRANS: trans r and
        IN1: a ∈ under b and IN2: b ∈ under c
shows a ∈ under c
proof−
  have (a,b) ∈ r ∧ (b,c) ∈ r
  using IN1 IN2 under-def by auto
  hence (a,c) ∈ r
  using TRANS trans-def[of r] by blast
  thus ?thesis unfolding under-def by simp
qed


lemma under-underS-trans:
assumes TRANS: trans r and ANTISYM: antisym r and
        IN1: a ∈ under b and IN2: b ∈ underS c
shows a ∈ underS c
proof−
  have 0: (a,b) ∈ r ∧ (b,c) ∈ r
  using IN1 IN2 under-def underS-def by auto
  hence 1: (a,c) ∈ r
  using TRANS trans-def[of r] by blast
  have 2: b ≠ c using IN2 underS-def by auto
  have 3: a ≠ c
```

proof
  assume $a = c$ **with** *0 2 ANTISYM antisym-def*[*of r*]
  show *False* **by** *auto*
qed
from *1 3* **show** *?thesis* **unfolding** *underS-def* **by** *simp*
qed

lemma *underS-under-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
  IN1: $a \in$ *underS b* **and** *IN2*: $b \in$ *under c*
shows $a \in$ *underS c*
proof−
  have *0*: $(a,b) \in r \wedge (b,c) \in r$
  using *IN1 IN2 under-def underS-def* **by** *auto*
  hence *1*: $(a,c) \in r$
  using *TRANS trans-def*[*of r*] **by** *blast*
  have *2*: $a \neq b$ **using** *IN1 underS-def* **by** *auto*
  have *3*: $a \neq c$
  proof
    assume $a = c$ **with** *0 2 ANTISYM antisym-def*[*of r*]
    show *False* **by** *auto*
  qed
  from *1 3* **show** *?thesis* **unfolding** *underS-def* **by** *simp*
qed

lemma *underS-underS-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
  IN1: $a \in$ *underS b* **and** *IN2*: $b \in$ *underS c*
shows $a \in$ *underS c*
proof−
  have $a \in$ *under b*
  using *IN1 underS-subset-under* **by** *auto*
  with *assms under-underS-trans* **show** *?thesis* **by** *auto*
qed

lemma *above-trans*:
assumes *TRANS*: *trans r* **and**
  IN1: $b \in$ *above a* **and** *IN2*: $c \in$ *above b*
shows $c \in$ *above a*
proof−
  have $(a,b) \in r \wedge (b,c) \in r$
  using *IN1 IN2 above-def* **by** *auto*
  hence $(a,c) \in r$
  using *TRANS trans-def*[*of r*] **by** *blast*
  thus *?thesis* **unfolding** *above-def* **by** *simp*

qed

lemma *above-aboveS-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
IN1: $b \in$ *above a* **and** *IN2*: $c \in$ *aboveS b*
shows $c \in$ *aboveS a*
proof –
have *0*: $(a,b) \in r \wedge (b,c) \in r$
using *IN1 IN2 above-def aboveS-def* **by** *auto*
hence *1*: $(a,c) \in r$
using *TRANS trans-def*[*of r*] **by** *blast*
have *2*: $b \neq c$ **using** *IN2 aboveS-def* **by** *auto*
have *3*: $a \neq c$
proof
assume $a = c$ **with** *0 2 ANTISYM antisym-def*[*of r*]
show *False* **by** *auto*
qed
from *1 3* **show** *?thesis* **unfolding** *aboveS-def* **by** *simp*
qed

lemma *aboveS-above-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
IN1: $b \in$ *aboveS a* **and** *IN2*: $c \in$ *above b*
shows $c \in$ *aboveS a*
proof –
have *0*: $(a,b) \in r \wedge (b,c) \in r$
using *IN1 IN2 above-def aboveS-def* **by** *auto*
hence *1*: $(a,c) \in r$
using *TRANS trans-def*[*of r*] **by** *blast*
have *2*: $a \neq b$ **using** *IN1 aboveS-def* **by** *auto*
have *3*: $a \neq c$
proof
assume $a = c$ **with** *0 2 ANTISYM antisym-def*[*of r*]
show *False* **by** *auto*
qed
from *1 3* **show** *?thesis* **unfolding** *aboveS-def* **by** *simp*
qed

lemma *aboveS-aboveS-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
IN1: $b \in$ *aboveS a* **and** *IN2*: $c \in$ *aboveS b*
shows $c \in$ *aboveS a*
proof –
have $b \in$ *above a*
using *IN1 aboveS-subset-above* **by** *auto*
with *assms above-aboveS-trans* **show** *?thesis* **by** *auto*

qed

lemma *under-Under-trans*:
assumes *TRANS*: *trans r* **and**
    IN1: $a \in$ *under b* **and** *IN2*: $b \in$ *Under C*
shows $a \in$ *Under C*
proof−
  have $(a,b) \in r \wedge (\forall c \in C. (b,c) \in r)$
  using *IN1 IN2 under-def Under-def* **by** *auto*
  hence $\forall c \in C. (a,c) \in r$
  using *TRANS trans-def*[*of r*] **by** *blast*
  moreover
  have $a \in$ *Field r* **using** *IN1 Field-def under-def* **by** *force*
  ultimately
  show *?thesis* **unfolding** *Under-def* **by** *auto*
qed

lemma *underS-Under-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
    IN1: $a \in$ *underS b* **and** *IN2*: $b \in$ *Under C*
shows $a \in$ *UnderS C*
proof−
  from *IN1* **have** $a \in$ *under b*
  using *underS-subset-under*[*of b*] **by** *blast*
  with *assms under-Under-trans*
  have $a \in$ *Under C* **by** *auto*

  moreover
  have $a \notin C$
  proof
    assume ∗: $a \in C$
    have *1*: $b \neq a \wedge (a,b) \in r$
    using *IN1 underS-def*[*of b*] **by** *auto*
    have $\forall c \in C. (b,c) \in r$
    using *IN2 Under-def*[*of C*] **by** *auto*
    with ∗ **have** $(b,a) \in r$ **by** *simp*
    with *1 ANTISYM antisym-def*[*of r*]
    show *False* **by** *blast*
  qed

  ultimately
  show *?thesis* **unfolding** *UnderS-def*
  using *Under-def* **by** *auto*
qed

lemma *under-UnderS-trans*:

assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
IN1: $a \in$ *under b* **and** *IN2*: $b \in$ *UnderS C*
shows $a \in$ *UnderS C*
proof−
from *IN2* **have** $b \in$ *Under C*
using *UnderS-subset-Under*[*of C*] **by** *blast*
with *assms under-Under-trans*
have $a \in$ *Under C* **by** *auto*

moreover
have $a \notin C$
proof
assume ∗: $a \in C$
have *1*: $(a,b) \in r$
using *IN1 under-def*[*of b*] **by** *auto*
have $\forall\, c \in C.\ b \neq c \wedge (b,c) \in r$
using *IN2 UnderS-def*[*of C*] **by** *auto*
with ∗ **have** $b \neq a \wedge (b,a) \in r$ **by** *simp*
with *1 ANTISYM antisym-def*[*of r*]
show *False* **by** *blast*
qed

ultimately
show *?thesis* **unfolding** *UnderS-def*
using *Under-def* **by** *auto*
qed

lemma *underS-UnderS-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
IN1: $a \in$ *underS b* **and** *IN2*: $b \in$ *UnderS C*
shows $a \in$ *UnderS C*
proof−
from *IN2* **have** $b \in$ *Under C*
using *UnderS-subset-Under*[*of C*] **by** *blast*
with *underS-Under-trans assms*
show *?thesis* **by** *auto*
qed

lemma *above-Above-trans*:
assumes *TRANS*: *trans r* **and**
IN1: $a \in$ *above b* **and** *IN2*: $b \in$ *Above C*
shows $a \in$ *Above C*
proof−
have $(b,a) \in r \wedge (\forall\, c \in C.\ (c,b) \in r)$
using *IN1 IN2 above-def Above-def* **by** *auto*
hence $\forall\, c \in C.\ (c,a) \in r$
using *TRANS trans-def*[*of r*] **by** *blast*

```
  moreover
  have a ∈ Field r using IN1 Field-def above-def by force
  ultimately
  show ?thesis unfolding Above-def by auto
qed


lemma aboveS-Above-trans:
assumes TRANS: trans r and ANTISYM: antisym r and
      IN1: a ∈ aboveS b and IN2: b ∈ Above C
shows a ∈ AboveS C
proof−
  from IN1 have a ∈ above b
  using aboveS-subset-above[of b] by blast
  with assms above-Above-trans
  have a ∈ Above C by auto

  moreover
  have a ∉ C
  proof
    assume ∗: a ∈ C
    have 1: b ≠ a ∧ (b,a) ∈ r
    using IN1 aboveS-def[of b] by auto
    have ∀ c ∈ C. (c,b) ∈ r
    using IN2 Above-def[of C] by auto
    with ∗ have (a,b) ∈ r by simp
    with 1 ANTISYM antisym-def[of r]
    show False by blast
  qed

  ultimately
  show ?thesis unfolding AboveS-def
  using Above-def by auto
qed


lemma above-AboveS-trans:
assumes TRANS: trans r and ANTISYM: antisym r and
      IN1: a ∈ above b and IN2: b ∈ AboveS C
shows a ∈ AboveS C
proof−
  from IN2 have b ∈ Above C
  using AboveS-subset-Above[of C] by blast
  with assms above-Above-trans
  have a ∈ Above C by auto

  moreover
  have a ∉ C
  proof
```

assume $*$: $a \in C$
have *1*: $(b,a) \in r$
using *IN1 above-def*[*of b*] **by** *auto*
have $\forall\, c \in C.\ b \neq c \wedge (c,b) \in r$
using *IN2 AboveS-def*[*of C*] **by** *auto*
with $*$ **have** $b \neq a \wedge (a,b) \in r$ **by** *simp*
with *1 ANTISYM antisym-def*[*of r*]
show *False* **by** *blast*
qed

ultimately
show *?thesis* **unfolding** *AboveS-def*
using *Above-def* **by** *auto*
qed

lemma *aboveS-AboveS-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
IN1: $a \in$ *aboveS b* **and** *IN2*: $b \in$ *AboveS C*
shows $a \in$ *AboveS C*
proof−
from *IN2* **have** $b \in$ *Above C*
using *AboveS-subset-Above*[*of C*] **by** *blast*
with *aboveS-Above-trans assms*
show *?thesis* **by** *auto*
qed

end

3.3 Properties depending on more than one relation

abbreviation *under* $\equiv$ *rel.under*
abbreviation *underS* $\equiv$ *rel.underS*
abbreviation *Under* $\equiv$ *rel.Under*
abbreviation *UnderS* $\equiv$ *rel.UnderS*
abbreviation *above* $\equiv$ *rel.above*
abbreviation *aboveS* $\equiv$ *rel.aboveS*
abbreviation *Above* $\equiv$ *rel.Above*
abbreviation *AboveS* $\equiv$ *rel.AboveS*

lemma *under-incr2*:
$r \leq r' \Longrightarrow$ *under r a* $\leq$ *under r' a*
unfolding *rel.under-def* **by** *blast*

lemma *underS-incr2*:
$r \leq r' \Longrightarrow$ *underS r a* $\leq$ *underS r' a*

unfolding *rel.underS-def* **by** *blast*

lemma *Under-incr*:
$r \leq r' \Longrightarrow Under\ r\ A \leq Under\ r\ A$
unfolding *rel.Under-def* **by** *blast*

lemma *UnderS-incr*:
$r \leq r' \Longrightarrow UnderS\ r\ A \leq UnderS\ r\ A$
unfolding *rel.UnderS-def* **by** *blast*

lemma *Under-incr-decr*:
$[\![r \leq r';\ A' \leq A]\!] \Longrightarrow Under\ r\ A \leq Under\ r\ A'$
unfolding *rel.Under-def* **by** *blast*

lemma *UnderS-incr-decr*:
$[\![r \leq r';\ A' \leq A]\!] \Longrightarrow UnderS\ r\ A \leq UnderS\ r\ A'$
unfolding *rel.UnderS-def* **by** *blast*

lemma *above-incr2*:
$r \leq r' \Longrightarrow above\ r\ a \leq above\ r'\ a$
unfolding *rel.above-def* **by** *blast*

lemma *aboveS-incr2*:
$r \leq r' \Longrightarrow aboveS\ r\ a \leq aboveS\ r'\ a$
unfolding *rel.aboveS-def* **by** *blast*

lemma *Above-incr*:
$r \leq r' \Longrightarrow Above\ r\ A \leq Above\ r\ A$
unfolding *rel.Above-def* **by** *blast*

lemma *AboveS-incr*:
$r \leq r' \Longrightarrow AboveS\ r\ A \leq AboveS\ r\ A$
unfolding *rel.AboveS-def* **by** *blast*

lemma *Above-incr-decr*:
$[\![r \leq r';\ A' \leq A]\!] \Longrightarrow Above\ r\ A \leq Above\ r\ A'$
unfolding *rel.Above-def* **by** *blast*

lemma *AboveS-incr-decr*:

$[\![r \le r'; A' \le A]\!] \Longrightarrow$ *AboveS r A* $\le$ *AboveS r A'*
unfolding *rel.AboveS-def* **by** *blast*

end

4 More on well-founded relations

theory *Wellfounded2* **imports** *Wellfounded Order-Relation2* ~~/src/HOL/Library/Wfrec
begin

This section contains some variations of results in the theory *Wellfounded.thy*:

- means for slightly more direct definitions by well-founded recursion;
- variations of well-founded induction;
- means for proving a linear order to be a well-order.

4.1 Well-founded recursion via genuine fixpoints

lemma *wfrec-fixpoint*:
fixes r :: $('a * 'a)$ *set* **and**
  H :: $('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$
assumes *WF*: *wf r* **and** *ADM*: *adm-wf r H*
shows *wfrec r H* = *H* (*wfrec r H*)
proof(*rule ext*)
 fix *x*
 have *wfrec r H x* = *H* (*cut* (*wfrec r H*) *r x*) *x*
 using *wfrec*[*of r H*] *WF* **by** *simp*
 also
 {**have** $\bigwedge$ *y*. (*y*,*x*) : *r* $\Longrightarrow$ (*cut* (*wfrec r H*) *r x*) *y* = (*wfrec r H*) *y*
  by (*auto simp add*: *cut-apply*)
  hence *H* (*cut* (*wfrec r H*) *r x*) *x* = *H* (*wfrec r H*) *x*
  using *ADM adm-wf-def*[*of r H*] **by** *auto*
 }
 finally show *wfrec r H x* = *H* (*wfrec r H*) *x* **.**
qed

lemma *adm-wf-unique-fixpoint*:
fixes r :: $('a * 'a)$ *set* **and**
  H :: $('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$ **and**
  f :: $'a \Rightarrow 'b$ **and** g :: $'a \Rightarrow 'b$
assumes *WF*: *wf r* **and** *ADM*: *adm-wf r H* **and** *fFP*: *f* = *H f* **and** *gFP*: *g* = *H*
g
shows *f* = *g*

proof−
 {**fix** x
  have $f\ x = g\ x$
  proof(*rule wf-induct*[*of r* ($\lambda x.\ f\ x = g\ x$)],
     auto simp add: *WF*)
   fix x **assume** $\forall y.\ (y, x) \in r \longrightarrow f\ y = g\ y$
   hence $H\ f\ x = H\ g\ x$ **using** *ADM adm-wf-def*[*of r H*] **by** *auto*
   thus $f\ x = g\ x$ **using** *fFP* **and** *gFP* **by** *simp*
  qed
 }
 thus *?thesis* **by** (*simp add*: *ext*)
qed

lemma *wfrec-unique-fixpoint*:
fixes r :: $('a * 'a)$ *set* **and**
  H :: $('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$ **and**
  f :: $'a \Rightarrow 'b$
assumes *WF*: *wf r* **and** *ADM*: *adm-wf r H* **and**
  fp: $f = H\ f$
shows f = *wfrec r H*
proof−
 have H (*wfrec r H*) = *wfrec r H*
 using *assms wfrec-fixpoint*[*of r H*] **by** *simp*
 thus *?thesis*
 using *assms adm-wf-unique-fixpoint*[*of r H wfrec r H*] **by** *simp*
qed

4.2 Characterizations of well-founded-ness

A transitive relation is well-founded iff it is "locally" well-founded, i.e., iff its restriction to the lower bounds of of any element is well-founded.

lemma *trans-wf-iff*:
assumes *trans r*
shows *wf r* = ($\forall a.$ *wf*(*r Int* ($r\hat{\ }{-1}``\{a\} \times r\hat{\ }{-1}``\{a\}$)))
proof−
 obtain R **where** *R-def*: R = ($\lambda\ a.$ *r Int* ($r\hat{\ }{-1}``\{a\} \times r\hat{\ }{-1}``\{a\}$)) **by** *blast*
 {**assume** $*$: *wf r*
  {**fix** a
   have *wf*($R\ a$)
   using $*$ *R-def wf-subset*[*of r R a*] **by** *auto*
  }
 }

 moreover
 {**assume** $*$: $\forall a.$ *wf*($R\ a$)
  have *wf r*
  proof(*unfold wf-def*, *clarify*)
   fix *phi a*

```
    assume **: ∀ a. (∀ b. (b,a) ∈ r ⟶ phi b) ⟶ phi a
    obtain chi where chi-def: chi = (λb. (b,a) ∈ r ⟶ phi b) by blast
    with * have wf (R a) by auto
    hence (∀ b. (∀ c. (c,b) ∈ R a ⟶ chi c) ⟶ chi b) ⟶ (∀ b. chi b)
    unfolding wf-def by blast
    moreover
    have ∀ b. (∀ c. (c,b) ∈ R a ⟶ chi c) ⟶ chi b
    proof(auto simp add: chi-def R-def)
      fix b
      assume 1: (b,a) ∈ r and 2: ∀ c. (c, b) ∈ r ∧ (c, a) ∈ r ⟶ phi c
      hence ∀ c. (c, b) ∈ r ⟶ phi c
      using assms trans-def[of r] by blast
      thus phi b using ** by blast
    qed
    ultimately have  ∀ b. chi b by (rule mp)
    with ** chi-def show phi a by blast
  qed
 }
 ultimately show ?thesis using R-def by blast
qed
```

The next lemma is a variation of *wf-eq-minimal* from Wellfounded, allowing one to assume the set included in the field.

```
lemma wf-eq-minimal2:
wf r = (∀ A. A <= Field r ∧ A ≠ {} ⟶ (∃ a ∈ A. ∀ a' ∈ A. ¬ (a',a) ∈ r))
proof−
 let ?phi = λ A. A ≠ {} ⟶ (∃ a ∈ A. ∀ a' ∈ A. ¬ (a',a) ∈ r)
 have wf r = (∀ A. ?phi A)
 proof(unfold wf-eq-minimal, auto)
   fix A c assume *: ∀ A. (∃ c. c ∈ A) ⟶ (∃ a∈A. ∀ a'. (a', a) ∈ r ⟶ a' ∉ A)
and
                  **: ∀ a∈A. ∃ a'∈A. (a', a) ∈ r and
                 ***: c ∈ A
   obtain a where a ∈ A ∧ (∀ a'. (a', a) ∈ r ⟶ a' ∉ A)
   using * *** by auto
   with ** show False by blast
 next
   fix A::'a set and c
   assume *: ∀ A. A ≠ {} ⟶ (∃ a∈A. ∀ a'∈A. (a', a) ∉ r) and
          **: c ∈ A
   obtain a where a ∈ A ∧ (∀ a'∈A. (a', a) ∉ r) using * ** by blast
   thus ∃ a∈A. ∀ a'. (a', a) ∈ r ⟶ a' ∉ A by blast
 qed

 also have (∀ A. ?phi A) = (∀ B ≤ Field r. ?phi B)
 proof
   assume ∀ A. ?phi A
   thus ∀ B ≤ Field r. ?phi B by simp
 next
```

```
    assume ∗: ∀ B ≤ Field r. ?phi B
    show ∀ A. ?phi A
    proof(clarify)
      fix A::'a set assume ∗∗: A ≠ {}
      obtain B where B-def: B = A Int (Field r) by blast
      show ∃ a ∈ A. ∀ a' ∈ A. (a',a) ∉ r
      proof(cases B = {})
        assume Case1: B = {}
        obtain a where 1: a ∈ A ∧ a ∉ Field r using ∗∗ Case1 B-def by auto
        hence ∀ a' ∈ A. (a',a) ∉ r using 1 unfolding Field-def by blast
        thus ?thesis using 1 by auto
      next
        assume Case2: B ≠ {} have 1: B ≤ Field r using B-def by auto
        obtain a where 2: a ∈ B ∧ (∀ a' ∈ B. (a',a) ∉ r)
        using Case2 1 ∗ by blast
        have ∀ a' ∈ A. (a',a) ∉ r
        proof(clarify)
          fix a' assume a' ∈ A and ∗∗: (a',a) ∈ r
          hence a' ∈ B using B-def Field-def by fastsimp
          thus False using 2 ∗∗ by auto
        qed
        thus ?thesis using 2 B-def by auto
      qed
    qed
  qed
  finally show ?thesis by blast
qed
```

The next lemma and its corollary enable one to prove that a linear order is a well-order in a way which is more standard than via well-founded-ness of the strict version of the relation.

```
lemma Linear-order-wf-diff-Id:
assumes LI: Linear-order r
shows wf(r − Id) = (∀ A ≤ Field r. A ≠ {} ⟶ (∃ a ∈ A. ∀ a' ∈ A. (a,a') ∈ r))
proof(cases r ≤ Id)
  assume Case1: r ≤ Id
  hence temp: r − Id = {} by blast
  hence wf(r − Id) by (auto simp add: temp)
  moreover
  {fix A assume ∗: A ≤ Field r and ∗∗: A ≠ {}
   obtain a where 1: r = {} ∨ r = {(a,a)} using LI
   unfolding order-on-defs using Case1 rel.Total-subset-Id by blast
   hence A = {a} ∧ r = {(a,a)} using ∗ ∗∗ unfolding Field-def by blast
   hence ∃ a ∈ A. ∀ a' ∈ A. (a,a') ∈ r using 1 by auto
  }
  ultimately show ?thesis by blast
next
  assume Case2: ¬ r ≤ Id
  hence 1: Field r = Field(r − Id) using rel.Total-Id-Field LI
```

unfolding *order-on-defs* **by** *blast*
show *?thesis*
proof
assume $*$: $wf(r - Id)$
show $\forall A \leq Field\ r.\ A \neq \{\} \longrightarrow (\exists a \in A.\ \forall a' \in A.\ (a,a') \in r)$
proof(*clarify*)
fix A **assume** $**$: $A \leq Field\ r$ **and** $***$: $A \neq \{\}$
hence $\exists a \in A.\ \forall a' \in A.\ (a',a) \notin r - Id$
using *1* $*$ **unfolding** *wf-eq-minimal2* **by** *auto*
moreover have $\forall a \in A.\ \forall a' \in A.\ ((a,a') \in r) = ((a',a) \notin r - Id)$
using *rel.Linear-order-in-diff-Id*[*of r*] $**$ *LI* **by** *blast*
ultimately show $\exists a \in A.\ \forall a' \in A.\ (a,a') \in r$ **by** *blast*
qed
next
assume $*$: $\forall A \leq Field\ r.\ A \neq \{\} \longrightarrow (\exists a \in A.\ \forall a' \in A.\ (a,a') \in r)$
show $wf(r - Id)$
proof(*unfold wf-eq-minimal2*, *clarify*)
fix A **assume** $**$: $A \leq Field(r - Id)$ **and** $***$: $A \neq \{\}$
hence $\exists a \in A.\ \forall a' \in A.\ (a,a') \in r$
using *1* $*$ **by** *auto*
moreover have $\forall a \in A.\ \forall a' \in A.\ ((a,a') \in r) = ((a',a) \notin r - Id)$
using *rel.Linear-order-in-diff-Id*[*of r*] $**$ *LI mono-Field*[*of* $r - Id$ *r*] **by** *blast*
ultimately show $\exists a \in A.\ \forall a' \in A.\ (a',a) \notin r - Id$ **by** *blast*
qed
qed
qed

corollary *Linear-order-Well-order-iff*:
assumes *LI*: *Linear-order r*
shows *Well-order r* $= (\forall A \leq Field\ r.\ A \neq \{\} \longrightarrow (\exists a \in A.\ \forall a' \in A.\ (a,a') \in r))$
using *assms* **unfolding** *well-order-on-def* **using** *Linear-order-wf-diff-Id*[*of r*] **by** *auto*

end

5 Well-order relations

theory *Wellorder-Relation* **imports** *Wellfounded2*
begin

In this section, we develop basic concepts and results pertaining to well-order relations. Note that we consider well-order relations as *non-strict relations*, i.e., as containing the diagonals of their fields.

locale *wo-rel* = *rel* + **assumes** *WELL*: *Well-order r*

begin

The following context encompasses all this section. In other words, for the whole section, we consider a fixed well-order relation r.

5.1 Auxiliaries

lemma *REFL*: *Refl r*
using *WELL order-on-defs*[*of* - *r*] **by** *auto*

lemma *TRANS*: *trans r*
using *WELL order-on-defs*[*of* - *r*] **by** *auto*

lemma *ANTISYM*: *antisym r*
using *WELL order-on-defs*[*of* - *r*] **by** *auto*

lemma *PREORD*: *Preorder r*
using *WELL order-on-defs*[*of* - *r*] **by** *auto*

lemma *PARORD*: *Partial-order r*
using *WELL order-on-defs*[*of* - *r*] **by** *auto*

lemma *TOTAL*: *Total r*
using *WELL order-on-defs*[*of* - *r*] **by** *auto*

lemma *TOTALS*: $\forall a \in \mathit{Field}\ r.\ \forall b \in \mathit{Field}\ r.\ (a,b) \in r \vee (b,a) \in r$
using *REFL TOTAL refl-on-def*[*of* - *r*] *total-on-def*[*of* - *r*] **by** *force*

lemma *LIN*: *Linear-order r*
using *WELL well-order-on-def*[*of* - *r*] **by** *auto*

lemma *WF*: $\mathit{wf}\ (r - \mathit{Id})$
using *WELL well-order-on-def*[*of* - *r*] **by** *auto*

lemma *cases-Total*:
$\bigwedge \mathit{phi}\ a\ b.\ [\![\{a,b\} <= \mathit{Field}\ r;\ ((a,b) \in r \Longrightarrow \mathit{phi}\ a\ b);\ ((b,a) \in r \Longrightarrow \mathit{phi}\ a\ b)]\!]$
$\Longrightarrow \mathit{phi}\ a\ b$
using *TOTALS* **by** *auto*

lemma *cases-Total2*:
$\bigwedge$ *phi a b*. $[\![\{a,b\} \leq Field\ r;\ ((a,b) \in r - Id \Longrightarrow phi\ a\ b);$
$((b,a) \in r - Id \Longrightarrow phi\ a\ b);\ (a = b \Longrightarrow phi\ a\ b)]\!]$
$\Longrightarrow phi\ a\ b$
using *TOTALS* **by** *auto*

lemma *cases-Total3*:
$\bigwedge$ *phi a b*. $[\![\{a,b\} \leq Field\ r;\ ((a,b) \in r - Id \vee (b,a) \in r - Id \Longrightarrow phi\ a\ b);$
$(a = b \Longrightarrow phi\ a\ b)]\!] \Longrightarrow phi\ a\ b$
using *TOTALS* **by** *auto*

5.2 Well-founded induction and recursion adapted to non-strict well-order relations

Here we provide induction and recursion principles specific to *non-strict* well-order relations. Although minor variations of those for well-founded relations, they will be useful for doing away with the tediousness of having to take out the diagonal each time in order to switch to a well-founded relation.

lemma *well-order-induct*:
assumes *IND*: $\bigwedge x.\ \forall y.\ y \neq x \wedge (y, x) \in r \longrightarrow P\ y \Longrightarrow P\ x$
shows $P\ a$
proof−
  have $\bigwedge x.\ \forall y.\ (y, x) \in r - Id \longrightarrow P\ y \Longrightarrow P\ x$
  using *IND* **by** *blast*
  thus $P\ a$ **using** *WF wf-induct*[*of* $r - Id\ P\ a$] **by** *blast*
qed

definition
worec :: $(('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$
where
worec $F \equiv$ *wfrec* $(r - Id)\ F$

definition
adm-wo :: $(('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow bool$
where
adm-wo $H \equiv \forall f\ g\ x.\ (\forall y \in underS\ x.\ f\ y = g\ y) \longrightarrow H\ f\ x = H\ g\ x$

lemma *worec-fixpoint*:
assumes *ADM*: *adm-wo H*
shows *worec* $H = H$ (*worec* H)
proof−
  let $?rS = r - Id$
  have *adm-wf* $(r - Id)\ H$

unfolding *adm-wf-def*
using *ADM adm-wo-def*[*of H*] *underS-def* **by** *auto*
hence *wfrec ?rS H* = *H* (*wfrec ?rS H*)
using *WF wfrec-fixpoint*[*of ?rS H*] **by** *simp*
thus *?thesis* **unfolding** *worec-def* **.**
qed

lemma *worec-unique-fixpoint*:
assumes *ADM*: *adm-wo H* **and** *fp*: *f* = *H f*
shows *f* = *worec H*
proof−
have *adm-wf* (*r* − *Id*) *H*
unfolding *adm-wf-def*
using *ADM adm-wo-def*[*of H*] *underS-def* **by** *auto*
hence *f* = *wfrec* (*r* − *Id*) *H*
using *fp WF wfrec-unique-fixpoint*[*of r* − *Id H*] **by** *simp*
thus *?thesis* **unfolding** *worec-def* **.**
qed

5.3 The notions of maximum, minimum, supremum, successor and order filter

We define the successor *of a set*, and not of an element (the latter is of course a particular case). Also, we define the maximum *of two elements*, *max2*, and the minimum *of a set*, *minim* – we chose these variants since we consider them the most useful for well-orders. The minimum is defined in terms of the auxiliary relational operator *isMinim*. Then, supremum and successor are defined in terms of minimum as expected. The minimum is only meaningful for non-empty sets, and the successor is only meaningful for sets for which strict upper bounds exist. Order filters for well-orders are also known as "initial segments".

definition *max2* :: $'a \Rightarrow 'a \Rightarrow 'a$
where *max2 a b* ≡ *if* (*a*,*b*) ∈ *r then b else a*

definition *isMinim* :: $'a$ *set* $\Rightarrow 'a \Rightarrow$ *bool*
where *isMinim A b* ≡ *b* ∈ *A* ∧ (∀ *a* ∈ *A*. (*b*,*a*) ∈ *r*)

definition *minim* :: $'a$ *set* $\Rightarrow 'a$
where *minim A* ≡ *THE b*. *isMinim A b*

definition *supr* :: $'a$ *set* $\Rightarrow 'a$
where *supr A* ≡ *minim* (*Above A*)

definition *suc* :: $'a$ *set* $\Rightarrow 'a$
where *suc A* ≡ *minim* (*AboveS A*)

definition *ofilter* :: *'a set* ⇒ *bool*
where
ofilter A ≡ (*A* ≤ *Field r*) ∧ (∀ *a* ∈ *A*. *under a* ≤ *A*)

5.3.1 Properties of max2

lemma *max2-greater-among*:
assumes *a* ∈ *Field r* **and** *b* ∈ *Field r*
shows (*a*, *max2 a b*) ∈ *r* ∧ (*b*, *max2 a b*) ∈ *r* ∧ *max2 a b* ∈ {*a*,*b*}
proof−
  {**assume** (*a*,*b*) ∈ *r*
   hence *?thesis* **using** *max2-def assms REFL refl-on-def*
   by (*auto simp add*: *refl-on-def*)
  }
  moreover
  {**assume** *a* = *b*
   hence (*a*,*b*) ∈ *r* **using** *REFL assms*
   by (*auto simp add*: *refl-on-def*)
  }
  moreover
  {**assume** ∗: *a* ≠ *b* ∧ (*b*,*a*) ∈ *r*
   hence (*a*,*b*) ∉ *r* **using** *ANTISYM*
   by (*auto simp add*: *antisym-def*)
   hence *?thesis* **using** ∗ *max2-def assms REFL refl-on-def*
   by (*auto simp add*: *refl-on-def*)
  }
  ultimately show *?thesis* **using** *assms TOTAL*
  total-on-def[*of Field r r*] **by** *blast*
qed

lemma *max2-greater*:
assumes *a* ∈ *Field r* **and** *b* ∈ *Field r*
shows (*a*, *max2 a b*) ∈ *r* ∧ (*b*, *max2 a b*) ∈ *r*
using *assms* **by** (*auto simp add*: *max2-greater-among*)

lemma *max2-among*:
assumes *a* ∈ *Field r* **and** *b* ∈ *Field r*
shows *max2 a b* ∈ {*a*, *b*}
using *assms max2-greater-among*[*of a b*] **by** *simp*

lemma *max2-equals1*:
assumes *a* ∈ *Field r* **and** *b* ∈ *Field r*
shows (*max2 a b* = *a*) = ((*b*,*a*) ∈ *r*)
using *assms ANTISYM* **unfolding** *antisym-def* **using** *TOTALS*
by(*auto simp add*: *max2-def max2-among*)

lemma *max2-equals2*:
assumes $a \in$ *Field* r **and** $b \in$ *Field* r
shows (*max2* a b $= b) = ((a,b) \in r)$
using *assms ANTISYM* **unfolding** *antisym-def* **using** *TOTALS*
unfolding *max2-def* **by** *auto*

lemma *max2-iff*:
assumes $a \in$ *Field* r **and** $b \in$ *Field* r
shows ((*max2* a b, $c) \in r) = ((a,c) \in r \wedge (b,c) \in r)$
proof
  assume (*max2* a b, $c) \in r$
  thus $(a,c) \in r \wedge (b,c) \in r$
  using *assms max2-greater*[*of a b*] *TRANS trans-def*[*of r*] **by** *blast*
next
  assume $(a,c) \in r \wedge (b,c) \in r$
  thus (*max2* a b, $c) \in r$
  using *assms max2-among*[*of a b*] **by** *auto*
qed

5.3.2 Existence and uniqueness for isMinim and well-definedness of minim

lemma *isMinim-unique*:
assumes *MINIM*: *isMinim B a* **and** *MINIM'*: *isMinim B a'*
shows $a = a'$
proof−
  {**have** $a \in B$
   using *MINIM isMinim-def* **by** *simp*
   hence $(a',a) \in r$
   using *MINIM' isMinim-def* **by** *simp*
  }
  moreover
  {**have** $a' \in B$
   using *MINIM' isMinim-def* **by** *simp*
   hence $(a,a') \in r$
   using *MINIM isMinim-def* **by** *simp*
  }
  ultimately
  show *?thesis* **using** *ANTISYM antisym-def*[*of r*] **by** *blast*
qed

lemma *Well-order-isMinim-exists*:
assumes *SUB*: $B \leq$ *Field* r **and** *NE*: $B \neq \{\}$
shows $\exists\, b$. *isMinim B b*
proof−

from *WF wf-eq-minimal*[*of r* $-$ *Id*] *NE Id-def* **obtain** b **where**
$*$: $b \in B \wedge (\forall b'. \ b' \neq b \wedge (b',b) \in r \longrightarrow b' \notin B)$ **by** *force*
show *?thesis*
proof(*simp add*: *isMinim-def*, *rule exI*[*of - b*], *auto*)
  show $b \in B$ **using** $*$ **by** *simp*
next
  fix b' **assume** *As*: $b' \in B$
  hence $**$: $b \in$ *Field* $r \wedge b' \in$ *Field* r **using** *As SUB* $*$ **by** *auto*

  from *As* $*$ **have** $b' = b \vee (b',b) \notin r$ **by** *auto*
  moreover
  {**assume** $b' = b$
   hence $(b,b') \in r$
   using $**$ *REFL* **by** (*auto simp add*: *refl-on-def*)
  }
  moreover
  {**assume** $b' \neq b \wedge (b',b) \notin r$
   hence $(b,b') \in r$
   using $**$ *TOTAL* **by** (*auto simp add*: *total-on-def*)
  }
  ultimately show $(b,b') \in r$ **by** *blast*
qed
qed

lemma *minim-isMinim*:
assumes *SUB*: $B \leq$ *Field* r **and** *NE*: $B \neq \{\}$
shows *isMinim B* (*minim B*)
proof$-$
  let *?phi* = (λ *b*. *isMinim B b*)
  from *assms Well-order-isMinim-exists*
  obtain b **where** $*$: *?phi b* **by** *blast*
  moreover
  have $\bigwedge b'$. *?phi* $b' \Longrightarrow b' = b$
  using *isMinim-unique* $*$ **by** *auto*
  ultimately show *?thesis*
  unfolding *minim-def* **using** *theI*[*of ?phi b*] **by** *blast*
qed

5.3.3 Properties of minim

lemma *minim-in*[*simp*]:
assumes $B \leq$ *Field* r **and** $B \neq \{\}$
shows *minim* $B \in B$
proof$-$
  from *minim-isMinim*[*of B*] *assms*
  have *isMinim B* (*minim B*) **by** *simp*
  thus *?thesis* **by** (*simp add*: *isMinim-def*)
qed

lemma *minim-inField*[*simp*]:
assumes $B \leq$ *Field r* **and** $B \neq \{\}$
shows *minim B* $\in$ *Field r*
proof−
  have *minim B* $\in B$ **using** *assms* **by** *simp*
  thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *minim-least*[*simp*]:
assumes *SUB*: $B \leq$ *Field r* **and** *IN*: $b \in B$
shows (*minim B*, *b*) $\in r$
proof−
  from *minim-isMinim*[*of B*] *assms*
  have *isMinim B* (*minim B*) **by** *auto*
  thus *?thesis* **by** (*auto simp add*: *isMinim-def IN*)
qed

lemma *minim-Under*:
$[\![B \leq$ *Field r*; $B \neq \{\}]\!] \Longrightarrow$ *minim B* $\in$ *Under B*
by(*auto simp add*: *Under-def*)

lemma *equals-minim*:
assumes *SUB*: $B \leq$ *Field r* **and** *IN*: $a \in B$ **and**
    LEAST: $\bigwedge b.\ b \in B \Longrightarrow (a,b) \in r$
shows $a =$ *minim B*
proof−
  from *minim-isMinim*[*of B*] *assms*
  have *isMinim B* (*minim B*) **by** *auto*
  moreover have *isMinim B a* **using** *IN LEAST isMinim-def* **by** *auto*
  ultimately show *?thesis*
  using *isMinim-unique* **by** *auto*
qed

lemma *equals-minim-Under*:
$[\![B \leq$ *Field r*; $a \in B$; $a \in$ *Under B*$]\!]$
 $\Longrightarrow a =$ *minim B*
by(*auto simp add*: *Under-def equals-minim*)

lemma *minim-iff-In-Under*:
assumes *SUB*: $B \leq$ *Field r* **and** *NE*: $B \neq \{\}$
shows ($a =$ *minim B*) $=$ ($a \in B \wedge a \in$ *Under B*)
proof

```
  assume a = minim B
  thus a ∈ B ∧ a ∈ Under B
  using assms minim-in minim-Under by simp
next
  assume a ∈ B ∧ a ∈ Under B
  thus a = minim B
  using assms equals-minim-Under by simp
qed


lemma minim-Under-under:
assumes NE: A ≠ {} and SUB: A ≤ Field r
shows Under A = under (minim A)
proof−

  have 1: minim A ∈ A
  using assms minim-in by auto
  have 2: ∀ x ∈ A. (minim A, x) ∈ r
  using assms minim-least by auto

  have Under A ≤ under (minim A)
  proof
    fix x assume x ∈ Under A
    with 1 Under-def have (x,minim A) ∈ r by auto
    thus x ∈ under(minim A) unfolding under-def by simp
  qed

  moreover

  have under (minim A) ≤ Under A
  proof
    fix x assume x ∈ under(minim A)
    hence 11: (x,minim A) ∈ r unfolding under-def by simp
    hence x ∈ Field r unfolding Field-def by auto
    moreover
    {fix a assume a ∈ A
     with 2 have (minim A, a) ∈ r by simp
     with 11 have (x,a) ∈ r
     using TRANS trans-def[of r] by blast
    }
    ultimately show x ∈ Under A by (unfold Under-def, auto)
  qed

  ultimately show ?thesis by blast
qed


lemma minim-UnderS-underS:
assumes NE: A ≠ {} and SUB: A ≤ Field r
```

shows *UnderS A* = *underS* (*minim A*)
proof−

have *1*: *minim A* ∈ *A*
using *assms minim-in* **by** *auto*
have *2*: ∀ *x* ∈ *A*. (*minim A*, *x*) ∈ *r*
using *assms minim-least* **by** *auto*

have *UnderS A* ≤ *underS* (*minim A*)
proof
fix *x* **assume** *x* ∈ *UnderS A*
with *1 UnderS-def* **have** *x* ≠ *minim A* ∧ (*x*,*minim A*) ∈ *r* **by** *auto*
thus *x* ∈ *underS*(*minim A*) **unfolding** *underS-def* **by** *simp*
qed

moreover

have *underS* (*minim A*) ≤ *UnderS A*
proof
fix *x* **assume** *x* ∈ *underS*(*minim A*)
hence *11*: *x* ≠ *minim A* ∧ (*x*,*minim A*) ∈ *r* **unfolding** *underS-def* **by** *simp*
hence *x* ∈ *Field r* **unfolding** *Field-def* **by** *auto*
moreover
{**fix** *a* **assume** *a* ∈ *A*
with *2* **have** *3*: (*minim A*, *a*) ∈ *r* **by** *simp*
with *11* **have** (*x*,*a*) ∈ *r*
using *TRANS trans-def*[*of r*] **by** *blast*
moreover
have *x* ≠ *a*
proof
assume *x* = *a*
with *11 3 ANTISYM antisym-def*[*of r*]
show *False* **by** *auto*
qed
ultimately
have *x* ≠ *a* ∧ (*x*,*a*) ∈ *r* **by** *simp*
}
ultimately show *x* ∈ *UnderS A* **by** (*unfold UnderS-def*, *auto*)
qed

ultimately show *?thesis* **by** *blast*
qed

5.3.4 Properties of supr

lemma *supr-Above*:
assumes *SUB*: *B* ≤ *Field r* **and** *ABOVE*: *Above B* ≠ {}
shows *supr B* ∈ *Above B*
proof(*unfold supr-def*)

have *Above B* $\leq$ *Field r*
using *Above-Field* **by** *auto*
thus *minim* (*Above B*) $\in$ *Above B*
using *assms* **by** *simp*
qed

lemma *supr-greater*:
assumes *SUB*: $B \leq$ *Field r* **and** *ABOVE*: *Above B* $\neq$ {} **and**
IN: $b \in B$
shows (b, *supr B*) $\in r$
proof−
from *assms supr-Above*
have *supr B* $\in$ *Above B* **by** *simp*
with *IN Above-def* **show** *?thesis* **by** *simp*
qed

lemma *supr-least-Above*:
assumes *SUB*: $B \leq$ *Field r* **and**
ABOVE: $a \in$ *Above B*
shows (*supr B*, a) $\in r$
proof(*unfold supr-def*)
have *Above B* $\leq$ *Field r*
using *Above-Field* **by** *auto*
thus (*minim* (*Above B*), a) $\in r$
using *assms minim-least*
by *simp*
qed

lemma *supr-least*:
$[\![B \leq$ *Field r*; $a \in$ *Field r*; $(\bigwedge b.\ b \in B \Longrightarrow (b,a) \in r)]\!]$
$\Longrightarrow$ (*supr B*, a) $\in r$
by(*auto simp add*: *supr-least-Above Above-def*)

lemma *equals-supr-Above*:
assumes *SUB*: $B \leq$ *Field r* **and** *ABV*: $a \in$ *Above B* **and**
MINIM: $\bigwedge a'.\ a' \in$ *Above B* $\Longrightarrow (a,a') \in r$
shows $a =$ *supr B*
proof(*unfold supr-def*)
have *Above B* $\leq$ *Field r*
using *Above-Field* **by** *auto*
thus $a =$ *minim* (*Above B*)
using *assms equals-minim* **by** *simp*
qed

lemma *equals-supr*:
assumes *SUB*: $B \leq$ *Field r* **and** *IN*: $a \in$ *Field r* **and**
ABV: $\bigwedge b.\ b \in B \Longrightarrow (b,a) \in r$ **and**
MINIM: $\bigwedge a'.\ [\![\ a' \in$ *Field r*; $\bigwedge b.\ b \in B \Longrightarrow (b,a') \in r]\!] \Longrightarrow (a,a') \in r$
shows $a =$ *supr B*
proof−
have $a \in$ *Above B*
unfolding *Above-def* **using** *ABV IN* **by** *simp*
moreover
have $\bigwedge a'.\ a' \in$ *Above B* $\Longrightarrow (a,a') \in r$
unfolding *Above-def* **using** *MINIM* **by** *simp*
ultimately show *?thesis*
using *equals-supr-Above SUB* **by** *auto*
qed

lemma *supr-inField*:
assumes $B \leq$ *Field r* **and** *Above B* $\neq \{\}$
shows *supr B* $\in$ *Field r*
proof−
have *supr B* $\in$ *Above B* **using** *supr-Above assms* **by** *simp*
thus *?thesis* **using** *assms Above-Field* **by** *auto*
qed

lemma *supr-above-Above*:
assumes *SUB*: $B \leq$ *Field r* **and** *ABOVE*: *Above B* $\neq \{\}$
shows *Above B* $=$ *above* (*supr B*)
proof(*unfold Above-def above-def*, *auto*)
fix *a* **assume** $a \in$ *Field r* $\forall b \in B.\ (b,a) \in r$
with *supr-least assms*
show (*supr B*, *a*) $\in r$ **by** *auto*
next
fix *b* **assume** (*supr B*, *b*) $\in r$
thus $b \in$ *Field r*
using *REFL refl-on-def*[*of - r*] **by** *auto*
next
fix *a b*
assume *1*: (*supr B*, *b*) $\in r$ **and** *2*: $a \in B$
with *assms supr-greater*
have (*a*,*supr B*) $\in r$ **by** *auto*
thus $(a,b) \in r$
using *1 TRANS trans-def*[*of r*] **by** *blast*
qed

lemma *supr-under*:
assumes *IN*: $a \in$ *Field r*

shows $a = supr\ (under\ a)$
proof−
  have $under\ a \leq Field\ r$
  using *under-Field* **by** *auto*
  moreover
  have $under\ a \neq \{\}$
  using *IN Refl-under-in REFL* **by** *auto*
  moreover
  have $a \in Above\ (under\ a)$
  using *in-Above-under IN* **by** *auto*
  moreover
  have $\forall\, a' \in Above\ (under\ a).\ (a,a') \in r$
  proof(*unfold Above-def under-def*, *auto*)
    fix a'
    assume $\forall\, aa.\ (aa,\ a) \in r \longrightarrow (aa,\ a') \in r$
    hence $(a,a) \in r \longrightarrow (a,a') \in r$ **by** *blast*
    moreover have $(a,a) \in r$
    using *REFL IN* **by** (*auto simp add*: *refl-on-def*)
    ultimately
    show $(a,\ a') \in r$ **by** (*rule mp*)
  qed
  ultimately show *?thesis*
  using *equals-supr-Above* **by** *auto*
qed

5.3.5 Properties of successor

lemma *suc-AboveS*:
assumes *SUB*: $B \leq Field\ r$ **and** *ABOVES*: $AboveS\ B \neq \{\}$
shows $suc\ B \in AboveS\ B$
proof(*unfold suc-def*)
  have $AboveS\ B \leq Field\ r$
  using *AboveS-Field* **by** *auto*
  thus $minim\ (AboveS\ B) \in AboveS\ B$
  using *assms* **by** *simp*
qed

lemma *suc-greater*:
assumes *SUB*: $B \leq Field\ r$ **and** *ABOVES*: $AboveS\ B \neq \{\}$ **and**
    IN: $b \in B$
shows $suc\ B \neq b \wedge (b,suc\ B) \in r$
proof−
  from *assms suc-AboveS*
  have $suc\ B \in AboveS\ B$ **by** *simp*
  with *IN AboveS-def* **show** *?thesis* **by** *simp*
qed

lemma *suc-least-AboveS*:
assumes *ABOVES*: $a \in$ *AboveS B*
shows (*suc B*,*a*) $\in r$
proof(*unfold suc-def*)
  have *AboveS B* $\leq$ *Field r*
  using *AboveS-Field* **by** *auto*
  thus (*minim* (*AboveS B*),*a*) $\in r$
  using *assms minim-least* **by** *simp*
qed

lemma *suc-least*:
$[\![B \leq \mathit{Field}\ r;\ a \in \mathit{Field}\ r;\ (\bigwedge\ b.\ b \in B \Longrightarrow a \neq b \wedge (b,a) \in r)]\!]$
$\Longrightarrow (\mathit{suc}\ B,\ a) \in r$
by(*auto simp add*: *suc-least-AboveS AboveS-def*)

lemma *suc-inField*:
assumes $B \leq$ *Field r* **and** *AboveS B* $\neq \{\}$
shows *suc B* $\in$ *Field r*
proof−
  have *suc B* $\in$ *AboveS B* **using** *suc-AboveS assms* **by** *simp*
  thus *?thesis*
  using *assms AboveS-Field* **by** *auto*
qed

lemma *equals-suc-AboveS*:
assumes *SUB*: $B \leq$ *Field r* **and** *ABV*: $a \in$ *AboveS B* **and**
    MINIM: $\bigwedge\ a'.\ a' \in \mathit{AboveS}\ B \Longrightarrow (a,a') \in r$
shows $a =$ *suc B*
proof(*unfold suc-def*)
  have *AboveS B* $\leq$ *Field r*
  using *AboveS-Field*[*of B*] **by** *auto*
  thus $a =$ *minim* (*AboveS B*)
  using *assms equals-minim*
  by *simp*
qed

lemma *equals-suc*:
assumes *SUB*: $B \leq$ *Field r* **and** *IN*: $a \in$ *Field r* **and**
 ABVS: $\bigwedge\ b.\ b \in B \Longrightarrow a \neq b \wedge (b,a) \in r$ **and**
 MINIM: $\bigwedge\ a'.\ [\![a' \in \mathit{Field}\ r;\ \bigwedge\ b.\ b \in B \Longrightarrow a' \neq b \wedge (b,a') \in r]\!] \Longrightarrow (a,a') \in r$
shows $a =$ *suc B*
proof−
  have $a \in$ *AboveS B*
  unfolding *AboveS-def* **using** *ABVS IN* **by** *simp*
  moreover

have $\bigwedge$ *a'*. *a'* ∈ *AboveS B* ⟹ (*a*,*a'*) ∈ *r*
unfolding *AboveS-def* **using** *MINIM* **by** *simp*
ultimately show *?thesis*
using *equals-suc-AboveS SUB* **by** *auto*
qed

lemma *suc-above-AboveS*:
assumes *SUB*: *B* ≤ *Field r* **and**
ABOVE: *AboveS B* ≠ {}
shows *AboveS B* = *above* (*suc B*)
proof(*unfold AboveS-def above-def*, *auto*)
fix *a* **assume** *a* ∈ *Field r* ∀ *b* ∈ *B*. *a* ≠ *b* ∧ (*b*,*a*) ∈ *r*
with *suc-least assms*
show (*suc B*,*a*) ∈ *r* **by** *auto*
next
fix *b* **assume** (*suc B*, *b*) ∈ *r*
thus *b* ∈ *Field r*
using *REFL refl-on-def*[*of* - *r*] **by** *auto*
next
fix *a b*
assume *1*: (*suc B*, *b*) ∈ *r* **and** *2*: *a* ∈ *B*
with *assms suc-greater*[*of B a*]
have (*a*,*suc B*) ∈ *r* **by** *auto*
thus (*a*,*b*) ∈ *r*
using *1 TRANS trans-def*[*of r*] **by** *blast*
next
fix *a*
assume *1*: (*suc B*, *a*) ∈ *r* **and** *2*: *a* ∈ *B*
with *assms suc-greater*[*of B a*]
have (*a*,*suc B*) ∈ *r* **by** *auto*
moreover have *suc B* ∈ *Field r*
using *assms suc-inField* **by** *simp*
ultimately have *a* = *suc B*
using *1 2 SUB ANTISYM antisym-def*[*of r*] **by** *auto*
thus *False*
using *assms suc-greater*[*of B a*] *2* **by** *auto*
qed

lemma *suc-underS*:
assumes *IN*: *a* ∈ *Field r*
shows *a* = *suc* (*underS a*)
proof−
have *underS a* ≤ *Field r*
using *underS-Field* **by** *auto*
moreover
have *a* ∈ *AboveS* (*underS a*)
using *in-AboveS-underS IN* **by** *auto*

```
  moreover
  have ∀ a' ∈ AboveS (underS a). (a,a') ∈ r
  proof(clarify)
    fix a'
    assume *: a' ∈ AboveS (underS a)
    hence **: a' ∈ Field r
    using AboveS-Field by auto
    {assume (a,a') ∉ r
     hence a' = a ∨ (a',a) ∈ r
     using TOTAL IN ** by (auto simp add: total-on-def)
     moreover
     {assume a' = a
      hence (a,a') ∈ r
      using REFL IN ** by (auto simp add: refl-on-def)
     }
     moreover
     {assume a' ≠ a ∧ (a',a) ∈ r
      hence a' ∈ underS a
      unfolding underS-def by simp
      hence a' ∉ AboveS (underS a)
      using AboveS-disjoint by blast
      with * have False by simp
     }
     ultimately have (a,a') ∈ r by blast
    }
    thus  (a, a') ∈ r by blast
  qed
  ultimately show ?thesis
  using equals-suc-AboveS by auto
qed


lemma suc-singl-pred:
assumes IN: a ∈ Field r and ABOVE-NE: aboveS a ≠ {} and
        REL: (a',suc {a}) ∈ r and DIFF: a' ≠ suc {a}
shows a' = a ∨ (a',a) ∈ r
proof−
  have *: suc {a} ∈ Field r ∧ a' ∈ Field r
  using WELL REL well-order-on-domain by auto
  {assume **: a' ≠ a
   hence (a,a') ∈ r ∨ (a',a) ∈ r
   using TOTAL IN * by (auto simp add: total-on-def)
   moreover
   {assume (a,a') ∈ r
    with ** * assms WELL suc-least[of {a} a']
    have (suc {a},a') ∈ r by auto
    with REL DIFF * ANTISYM antisym-def[of r]
    have False by simp
   }
```

ultimately have $(a',a) \in r$
by *blast*
}
thus *?thesis* **by** *blast*
qed

lemma *under-underS-suc*:
assumes *IN*: $a \in$ *Field r* **and** *ABV*: *aboveS a* $\neq \{\}$
shows *underS* (*suc* $\{a\}$) = *under a*
proof−
have *1*: *AboveS* $\{a\} \neq \{\}$
using *ABV aboveS-AboveS-singl* **by** *auto*
have *2*: $a \neq$ *suc* $\{a\} \wedge (a,$*suc* $\{a\}) \in r$
using *suc-greater*[*of* $\{a\}$ *a*] *IN 1* **by** *auto*

have *underS* (*suc* $\{a\}$) $\leq$ *under a*
proof(*unfold underS-def under-def*, *auto*)
fix *x* **assume** $*$: $x \neq$ *suc* $\{a\}$ **and** $**$: $(x,$*suc* $\{a\}) \in r$
with *suc-singl-pred*[*of a x*] *IN ABV*
have $x = a \vee (x,a) \in r$ **by** *auto*
with *REFL refl-on-def*[*of - r*] *IN*
show $(x,a) \in r$ **by** *auto*
qed

moreover

have *under a* $\leq$ *underS* (*suc* $\{a\}$)
proof(*unfold underS-def under-def*, *auto*)
assume (*suc* $\{a\}$, $a) \in r$
with *2 ANTISYM antisym-def*[*of r*]
show *False* **by** *auto*
next
fix *x* **assume** $*$: $(x,a) \in r$
with *2 TRANS trans-def*[*of r*]
show $(x,$*suc* $\{a\}) \in r$ **by** *blast*

qed

ultimately show *?thesis* **by** *blast*
qed

5.3.6 Properties of order filters

lemma *under-ofilter*[*simp*]:
ofilter (*under a*)
proof(*unfold ofilter-def under-def*, *auto simp add*: *Field-def*)
fix *aa x*
assume $(aa,a) \in r$ $(x,aa) \in r$

thus $(x,a) \in r$
using *TRANS trans-def*[*of r*] **by** *blast*
qed

lemma *underS-ofilter*[*simp*]:
ofilter (*underS a*)
proof(*unfold ofilter-def underS-def under-def*, *auto simp add*: *Field-def*)
fix *aa* **assume** $(a,\ aa) \in r$ $(aa,\ a) \in r$ **and** *DIFF*: $aa \neq a$
thus *False*
using *ANTISYM antisym-def*[*of r*] **by** *blast*
next
fix *aa x*
assume $(aa,a) \in r$ $aa \neq a$ $(x,aa) \in r$
thus $(x,a) \in r$
using *TRANS trans-def*[*of r*] **by** *blast*
qed

lemma *Field-ofilter*[*simp*]:
ofilter (*Field r*)
by(*unfold ofilter-def under-def*, *auto simp add*: *Field-def*)

lemma *ofilter-underS-Field*:
ofilter $A = ((\exists\, a \in \mathit{Field}\ r.\ A = \mathit{underS}\ a) \vee (A = \mathit{Field}\ r))$
proof
assume $(\exists\, a{\in}\mathit{Field}\ r.\ A = \mathit{underS}\ a) \vee A = \mathit{Field}\ r$
thus *ofilter A*
by *auto*
next
assume $*$: *ofilter A*
let *?One* = $(\exists\, a{\in}\mathit{Field}\ r.\ A = \mathit{underS}\ a)$
let *?Two* = $(A = \mathit{Field}\ r)$
show *?One* $\vee$ *?Two*
proof(*cases ?Two*, *simp*)
let *?B* = $(\mathit{Field}\ r) - A$
let *?a* = *minim ?B*
assume $A \neq \mathit{Field}\ r$
moreover have $A \leq \mathit{Field}\ r$ **using** $*$ *ofilter-def* **by** *simp*
ultimately have *1*: *?B* $\neq \{\}$ **by** *blast*
hence *2*: *?a* $\in$ *Field r* **using** *minim-inField*[*of ?B*] **by** *blast*
have *3*: *?a* $\in$ *?B* **using** *minim-in*[*of ?B*] *1* **by** *blast*
hence *4*: *?a* $\notin$ *A* **by** *blast*
have *5*: $A \leq \mathit{Field}\ r$ **using** $*$ *ofilter-def*[*of A*] **by** *auto*

moreover
have *A* = *underS ?a*
proof

show $A \leq$ *underS ?a*
proof(*unfold underS-def*, *auto simp add*: *4*)
fix x **assume** $**$: $x \in A$
hence *11*: $x \in$ *Field r* **using** *5* **by** *auto*
have *12*: $x \neq$ *?a* **using** *4* $**$ **by** *auto*
have *13*: *under x* $\leq A$ **using** $*$ *ofilter-def* $**$ **by** *auto*
{**assume** $(x,?a) \notin r$
hence $(?a,x) \in r$
using *TOTAL total-on-def*[*of Field r r*]
2 4 11 12 **by** *auto*
hence *?a* $\in$ *under x* **using** *under-def* **by** *auto*
hence *?a* $\in A$ **using** $**$ *13* **by** *blast*
with *4* **have** *False* **by** *simp*
}
thus $(x,?a) \in r$ **by** *blast*
qed
next
show *underS ?a* $\leq A$
proof(*unfold underS-def*, *auto*)
fix x
assume $**$: $x \neq$ *?a* **and** $***$: $(x,?a) \in r$
hence *11*: $x \in$ *Field r* **using** *Field-def* **by** *fastforce*
{**assume** $x \notin A$
hence $x \in$ *?B* **using** *11* **by** *auto*
hence $(?a,x) \in r$ **using** *3 minim-least*[*of ?B x*] **by** *blast*
hence *False*
using *ANTISYM antisym-def*[*of r*] $**$ $***$ **by** *auto*
}
thus $x \in A$ **by** *blast*
qed
qed
ultimately have *?One* **using** *2* **by** *blast*
thus *?thesis* **by** *simp*
qed
qed

lemma *ofilter-Under*[*simp*]:
assumes $A \leq$ *Field r*
shows *ofilter*(*Under A*)
proof(*unfold ofilter-def*, *auto*)
fix x **assume** $x \in$ *Under A*
thus $x \in$ *Field r*
using *Under-Field assms* **by** *auto*
next
fix *a x*
assume $a \in$ *Under A* **and** $x \in$ *under a*
thus $x \in$ *Under A*
using *TRANS under-Under-trans* **by** *auto*

qed

lemma *ofilter-UnderS*[*simp*]:
assumes $A \leq$ *Field r*
shows *ofilter*(*UnderS A*)
proof(*unfold ofilter-def*, *auto*)
  fix x **assume** $x \in$ *UnderS A*
  thus $x \in$ *Field r*
  using *UnderS-Field assms* **by** *auto*
next
  fix a x
  assume $a \in$ *UnderS A* **and** $x \in$ *under a*
  thus $x \in$ *UnderS A*
  using *TRANS ANTISYM under-UnderS-trans* **by** *auto*
qed

lemma *ofilter-Int*[*simp*]: $[\![$*ofilter A*; *ofilter B*$]\!] \Longrightarrow$ *ofilter*(*A Int B*)
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-INTER*:
$[\![I \neq \{\}; \bigwedge i.\ i \in I \Longrightarrow$ *ofilter*$(A\ i)]\!] \Longrightarrow$ *ofilter* $(\bigcap\ i \in I.\ A\ i)$
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-Inter*:
$[\![S \neq \{\}; \bigwedge A.\ A \in S \Longrightarrow$ *ofilter A*$]\!] \Longrightarrow$ *ofilter* (*Inter S*)
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-Un*[*simp*]: $[\![$*ofilter A*; *ofilter B*$]\!] \Longrightarrow$ *ofilter*$(A \cup B)$
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-UNION*:
$(\bigwedge i.\ i \in I \Longrightarrow$ *ofilter*$(A\ i)) \Longrightarrow$ *ofilter* $(\bigcup\ i \in I.\ A\ i)$
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-Union*:
$(\bigwedge A.\ A \in S \Longrightarrow$ *ofilter A*$) \Longrightarrow$ *ofilter* (*Union S*)
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-under-UNION*:
assumes *ofilter A*
shows $A = (\bigcup\ a \in A.$ *under a*$)$

proof
  have $\forall a \in A.$ *under a* $\leq A$
  using *assms ofilter-def* **by** *auto*
  thus $(\bigcup\ a \in A.\ under\ a) \leq A$ **by** *blast*
next
  have $\forall a \in A.\ a \in under\ a$
  using *REFL Refl-under-in assms ofilter-def* **by** *blast*
  thus $A \leq (\bigcup\ a \in A.\ under\ a)$ **by** *blast*
qed

lemma *ofilter-under-Union*:
$ofilter\ A \Longrightarrow A = Union\ \{under\ a|\ a.\ a \in A\}$
using *ofilter-under-UNION*[*of A*]
by(*unfold Union-eq*, *auto*)

5.3.7 Other properties

lemma *Trans-Under-regressive*:
assumes *NE*: $A \neq \{\}$ **and** *SUB*: $A \leq Field\ r$
shows $Under(Under\ A) \leq Under\ A$
proof
  let $?a = minim\ A$

  have *1*: $minim\ A \in Under\ A$
  using *assms minim-Under* **by** *auto*
  have *2*: $\forall y \in A.\ (minim\ A,\ y) \in r$
  using *assms minim-least* **by** *auto*

  fix x **assume** $x \in Under(Under\ A)$
  with *1* **have** *1*: $(x,minim\ A) \in r$
  using *Under-def* **by** *auto*
  with *Field-def* **have** $x \in Field\ r$ **by** *fastforce*
  moreover
  {fix y **assume** $*$: $y \in A$
   hence $(x,y) \in r$
   using *1 2 TRANS trans-def*[*of r*] **by** *blast*
   with *Field-def* **have** $(x,y) \in r$ **by** *auto*
  }
  ultimately
  show $x \in Under\ A$ **unfolding** *Under-def* **by** *auto*
qed

lemma *ofilter-linord*:
assumes *OF1*: *ofilter A* **and** *OF2*: *ofilter B*
shows $A \leq B \vee B \leq A$
proof(*cases* $A = Field\ r$)
  assume *Case1*: $A = Field\ r$

hence $B \leq A$ **using** *OF2 ofilter-def* **by** *auto*
thus *?thesis* **by** *simp*
next
assume *Case2*: $A \neq$ *Field r*
with *ofilter-underS-Field OF1* **obtain** *a* **where**
1: $a \in$ *Field r* $\wedge$ $A =$ *underS a* **by** *auto*
show *?thesis*
proof(*cases* $B =$ *Field r*)
assume *Case21*: $B =$ *Field r*
hence $A \leq B$ **using** *OF1 ofilter-def* **by** *auto*
thus *?thesis* **by** *simp*
next
assume *Case22*: $B \neq$ *Field r*
with *ofilter-underS-Field OF2* **obtain** *b* **where**
2: $b \in$ *Field r* $\wedge$ $B =$ *underS b* **by** *auto*
have $a = b \vee (a,b) \in r \vee (b,a) \in r$
using *1 2 TOTAL total-on-def*[*of - r*] **by** *auto*
moreover
{assume $a = b$ **with** *1 2* **have** *?thesis* **by** *auto*
}
moreover
{assume $(a,b) \in r$
with *underS-incr TRANS ANTISYM 1 2*
have $A \leq B$ **by** *auto*
hence *?thesis* **by** *auto*
}
moreover
{assume $(b,a) \in r$
with *underS-incr TRANS ANTISYM 1 2*
have $B \leq A$ **by** *auto*
hence *?thesis* **by** *auto*
}
ultimately show *?thesis* **by** *blast*
qed
qed

lemma *ofilter-AboveS-Field*:
assumes *ofilter A*
shows $A \cup ($*AboveS A*$) =$ *Field r*
proof
show $A \cup ($*AboveS A*$) \leq$ *Field r*
using *assms ofilter-def AboveS-Field* **by** *auto*
next
{fix *x* **assume** $*$: $x \in$ *Field r* **and** $**$: $x \notin A$
{fix *y* **assume** $***$: $y \in A$
with $**$ **have** *1*: $y \neq x$ **by** *auto*
{assume $(y,x) \notin r$
moreover

have $y \in$ *Field r* **using** *assms ofilter-def* $***$ **by** *auto*
ultimately have $(x,y) \in r$
using *1* $*$ *TOTAL total-on-def*[*of* - *r*] **by** *auto*
with $***$ *assms ofilter-def under-def* **have** $x \in A$ **by** *auto*
with $**$ **have** *False* **by** *contradiction*
}
hence $(y,x) \in r$ **by** *blast*
with *1* **have** $y \neq x \wedge (y,x) \in r$ **by** *auto*
}
with $*$ **have** $x \in$ *AboveS A* **unfolding** *AboveS-def* **by** *auto*
}
thus *Field r* $\leq A \cup$ (*AboveS A*) **by** *blast*
qed

lemma *ofilter-suc-Field*:
assumes *OF*: *ofilter A* **and** *NE*: $A \neq$ *Field r*
shows *ofilter* $(A \cup \{$*suc A*$\})$
proof−

have *1*: $A \leq$ *Field r* **using** *OF ofilter-def* **by** *auto*
hence *2*: *AboveS A* $\neq \{\}$
using *ofilter-AboveS-Field NE OF* **by** *blast*
from *1 2 suc-inField*
have *3*: *suc A* $\in$ *Field r* **by** *auto*

show *?thesis*
proof(*unfold ofilter-def*, *auto simp add*: *1 3*)
fix *a x*
assume $a \in A$ $x \in$ *under a* $x \notin A$
with *OF ofilter-def* **have** *False* **by** *auto*
thus $x =$ *suc A* **by** *simp*
next
fix *x* **assume** $*$: $x \in$ *under* (*suc A*) **and** $**$: $x \notin A$
hence $x \in$ *Field r* **using** *under-def Field-def* **by** *fastforce*
with $**$ **have** $x \in$ *AboveS A*
using *ofilter-AboveS-Field*[*of A*] *OF* **by** *auto*
hence (*suc A*,x) $\in r$
using *suc-least-AboveS* **by** *auto*
moreover
have (x,*suc A*) $\in r$ **using** $*$ *under-def* **by** *auto*
ultimately show $x =$ *suc A*
using *ANTISYM antisym-def*[*of r*] **by** *auto*
qed
qed

lemma *suc-ofilter-in*:
assumes *OF*: *ofilter A* **and** *ABOVE-NE*: *AboveS A* $\neq \{\}$ **and**

```
    REL: (b,suc A) ∈ r and DIFF: b ≠ suc A
shows b ∈ A
proof−
  have *: suc A ∈ Field r ∧ b ∈ Field r
  using WELL REL well-order-on-domain by auto
  {assume **: b ∉ A
   hence b ∈ AboveS A
   using OF * ofilter-AboveS-Field by auto
   hence (suc A, b) ∈ r
   using suc-least-AboveS by auto
   hence False using REL DIFF ANTISYM *
   by (auto simp add: antisym-def)
  }
  thus ?thesis by blast
qed

end

abbreviation worec ≡ wo-rel.worec
abbreviation adm-wo ≡ wo-rel.adm-wo
abbreviation isMinim ≡ wo-rel.isMinim
abbreviation minim ≡ wo-rel.minim
abbreviation max2 ≡ wo-rel.max2
abbreviation supr ≡ wo-rel.supr
abbreviation suc ≡ wo-rel.suc
abbreviation ofilter ≡ wo-rel.ofilter

end
```

6 Well-order embeddings

```
theory Wellorder-Embedding imports ~~/src/HOL/Library/Zorn Fun2 Wellorder-Relation
begin
```

In this section, we introduce well-order *embeddings* and *isomorphisms* and prove their basic properties. The notion of embedding is considered from the point of view of the theory of ordinals, and therefore requires the source to be injected as an *initial segment* (i.e., *order filter*) of the target. A main result of this section is the existence of embeddings (in one direction or another) between any two well-orders, having as a consequence the fact that, given any two sets on any two types, one is smaller than (i.e., can be injected into) the other.

6.1 Auxiliaries

lemma *UNION-inj-on-ofilter*:
assumes *WELL*: *Well-order r* **and**
OF: $\bigwedge$ *i*. $i \in I \Longrightarrow$ *ofilter r* $(A\ i)$ **and**
INJ: $\bigwedge$ *i*. $i \in I \Longrightarrow$ *inj-on f* $(A\ i)$
shows *inj-on f* $(\bigcup\ i \in I.\ A\ i)$
proof−
have *wo-rel r* **using** *WELL* **by** (*simp add*: *wo-rel-def*)
hence $\bigwedge$ *i j*. $[\![i \in I;\ j \in I]\!] \Longrightarrow A\ i <= A\ j \vee A\ j <= A\ i$
using *wo-rel.ofilter-linord*[*of r*] *OF* **by** *blast*
with *WELL INJ* **show** *?thesis*
by (*auto simp add*: *UNION-inj-on*)
qed

lemma *UNION-bij-betw-ofilter*:
assumes *WELL*: *Well-order r* **and**
OF: $\bigwedge$ *i*. $i \in I \Longrightarrow$ *ofilter r* $(A\ i)$ **and**
BIJ: $\bigwedge$ *i*. $i \in I \Longrightarrow$ *bij-betw f* $(A\ i)\ (A'\ i)$
shows *bij-betw f* $(\bigcup\ i \in I.\ A\ i)\ (\bigcup\ i \in I.\ A'\ i)$
proof−
have *wo-rel r* **using** *WELL* **by** (*simp add*: *wo-rel-def*)
hence $\bigwedge$ *i j*. $[\![i \in I;\ j \in I]\!] \Longrightarrow A\ i \le A\ j \vee A\ j \le A\ i$
using *wo-rel.ofilter-linord*[*of r*] *OF* **by** *blast*
with *WELL BIJ* **show** *?thesis*
by (*auto simp add*: *UNION-bij-betw*)
qed

lemma *under-underS-bij-betw*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
IN: $a \in$ *Field r* **and** *IN'*: $f\ a \in$ *Field r'* **and**
BIJ: *bij-betw f* (*underS r a*) (*underS r'* $(f\ a)$)
shows *bij-betw f* (*under r a*) (*under r'* $(f\ a)$)
proof−
have $a \notin$ *underS r a* $\wedge\ f\ a \notin$ *underS r'* $(f\ a)$
unfolding *rel.underS-def* **by** *auto*
moreover
{**have** *Refl r* $\wedge$ *Refl r'* **using** *WELL WELL'*
by (*auto simp add*: *order-on-defs*)
hence *under r a* = *underS r a* $\cup \{a\}$ $\wedge$
under r' $(f\ a)$ = *underS r'* $(f\ a) \cup \{f\ a\}$

using *IN IN′* **by**(*auto simp add*: *rel.Refl-under-underS*)
}
ultimately show *?thesis*
using *BIJ notIn-Un-bij-betw*[*of a underS r a f underS r′* (*f a*)] **by** *auto*
qed

6.2 (Well-order) embeddings, strict embeddings, isomorphisms and order-compatible functions

Standardly, a function is an embedding of a well-order in another if it injectively and order-compatibly maps the former into an order filter of the latter. Here we opt for a more succinct definition (operator *embed*), asking that, for any element in the source, the function should be a bijection between the set of strict lower bounds of that element and the set of strict lower bounds of its image. (Later we prove equivalence with the standard definition – lemma *embed-iff-compat-inj-on-ofilter*.) A *strict embedding* (operator *embedS*) is a non-bijective embedding and an isomorphism (operator *iso*) is a bijective embedding.

definition *embed* :: $'a\ rel \Rightarrow 'a'\ rel \Rightarrow ('a \Rightarrow 'a') \Rightarrow bool$
where
$embed\ r\ r'\ f \equiv \forall\, a \in Field\ r.\ bij\text{-}betw\ f\ (under\ r\ a)\ (under\ r'\ (f\ a))$

lemmas *embed-defs* = *embed-def embed-def-raw*

Strict embeddings:

definition *embedS* :: $'a\ rel \Rightarrow 'a'\ rel \Rightarrow ('a \Rightarrow 'a') \Rightarrow bool$
where
$embedS\ r\ r'\ f \equiv embed\ r\ r'\ f \wedge \neg\ bij\text{-}betw\ f\ (Field\ r)\ (Field\ r')$

lemmas *embedS-defs* = *embedS-def embedS-def-raw*

definition *iso* :: $'a\ rel \Rightarrow 'a'\ rel \Rightarrow ('a \Rightarrow 'a') \Rightarrow bool$
where
$iso\ r\ r'\ f \equiv embed\ r\ r'\ f \wedge bij\text{-}betw\ f\ (Field\ r)\ (Field\ r')$

lemmas *iso-defs* = *iso-def iso-def-raw*

definition *compat* :: $'a\ rel \Rightarrow 'a'\ rel \Rightarrow ('a \Rightarrow 'a') \Rightarrow bool$
where
$compat\ r\ r'\ f \equiv \forall\, a\ b.\ (a,b) \in r \longrightarrow (f\ a,\ f\ b) \in r'$

lemma *embed-halfcong*:
assumes *EQ*: $\bigwedge$ *a*. $a \in$ *Field* $r \Longrightarrow f\ a = g\ a$ **and**
EMB: *embed* $r\ r'\ f$
shows *embed* $r\ r'\ g$
proof(*unfold embed-def*, *auto*)
fix *a* **assume** $*$: $a \in$ *Field* r
hence *bij-betw* f (*under* $r\ a$) (*under* r' ($f\ a$))
using *EMB* **unfolding** *embed-def* **by** *simp*
moreover
{**have** *under* $r\ a \leq$ *Field* r
by (*auto simp add*: *rel.under-Field*)
hence $\bigwedge$ *b*. $b \in$ *under* $r\ a \Longrightarrow f\ b = g\ b$
using *EQ* **by** *blast*
}
moreover have $f\ a = g\ a$ **using** $*$ *EQ* **by** *auto*
ultimately show *bij-betw* g (*under* $r\ a$) (*under* r' ($g\ a$))
using *bij-betw-cong*[*of under* $r\ a\ f\ g$ *under* r' ($f\ a$)] **by** *auto*
qed

lemma *embed-cong*[*fundef-cong*]:
assumes $\bigwedge$ *a*. $a \in$ *Field* $r \Longrightarrow f\ a = g\ a$
shows *embed* $r\ r'\ f$ = *embed* $r\ r'\ g$
using *assms embed-halfcong*[*of* $r\ f\ g\ r'$]
embed-halfcong[*of* $r\ g\ f\ r'$] **by** *auto*

lemma *embedS-cong*[*fundef-cong*]:
assumes $\bigwedge$ *a*. $a \in$ *Field* $r \Longrightarrow f\ a = g\ a$
shows *embedS* $r\ r'\ f$ = *embedS* $r\ r'\ g$
unfolding *embedS-def* **using** *assms*
embed-cong[*of* $r\ f\ g\ r'$] *bij-betw-cong*[*of Field* $r\ f\ g$ *Field* r'] **by** *blast*

lemma *iso-cong*[*fundef-cong*]:
assumes $\bigwedge$ *a*. $a \in$ *Field* $r \Longrightarrow f\ a = g\ a$
shows *iso* $r\ r'\ f$ = *iso* $r\ r'\ g$
unfolding *iso-def* **using** *assms*
embed-cong[*of* $r\ f\ g\ r'$] *bij-betw-cong*[*of Field* $r\ f\ g$ *Field* r'] **by** *blast*

lemma *id-compat*: *compat* $r\ r$ *id*
by(*auto simp add*: *id-def compat-def*)

lemma *comp-compat*:
$[\![$*compat* $r\ r'\ f$; *compat* $r'\ r''\ f'$$]\!] \Longrightarrow$ *compat* $r\ r''$ (f' *o* f)
by(*auto simp add*: *comp-def compat-def*)

lemma *compat-wf*:
assumes *CMP*: *compat* r r' f **and** *WF*: *wf* r'
shows *wf* r
proof−
  have $r \le$ *inv-image* r' f
  unfolding *inv-image-def* **using** *CMP*
  by (*auto simp add*: *compat-def*)
  with *WF* **show** *?thesis*
  using *wf-inv-image*[*of* r' f] *wf-subset*[*of inv-image* r' f] **by** *auto*
qed

lemma *id-embed*: *embed r r id*
by(*auto simp add*: *id-def embed-def bij-betw-def*)

lemma *id-iso*: *iso r r id*
by(*auto simp add*: *id-def embed-def iso-def bij-betw-def*)

lemma *embed-in-Field*:
assumes *WELL*: *Well-order* r **and**
    EMB: *embed* r r' f **and** *IN*: $a \in$ *Field* r
shows $f\ a \in$ *Field* r'
proof−
  have *Well*: *wo-rel r*
  using *WELL* **by** (*auto simp add*: *wo-rel-def*)
  hence *1*: *Refl r*
  by (*auto simp add*: *wo-rel.REFL*)
  hence $a \in$ *under* r a **using** *IN rel.Refl-under-in* **by** *fastforce*
  hence $f\ a \in$ *under* r' $(f\ a)$
  using *EMB IN* **by** (*auto simp add*: *embed-def bij-betw-def*)
  thus *?thesis* **unfolding** *Field-def*
  by (*auto simp*: *rel.under-def*)
qed

lemma *comp-embed*:
assumes *WELL*: *Well-order* r **and**
    EMB: *embed* r r' f **and** *EMB*$'$: *embed* r' r'' f'
shows *embed* r r'' $(f'\ o\ f)$
proof(*unfold embed-def*, *auto*)
  fix a **assume** $*$: $a \in$ *Field* r
  hence *bij-betw* f (*under* r a) (*under* r' $(f\ a)$)
  using *embed-def*[*of* r] *EMB* **by** *auto*
  moreover
  {**have** $f\ a \in$ *Field* r'
   using *EMB WELL* $*$ **by** (*auto simp add*: *embed-in-Field*)

hence *bij-betw* f' (*under* r' (f a)) (*under* r'' (f' (f a)))
using *embed-def*[*of* r'] *EMB'* **by** *auto*
}
ultimately
show *bij-betw* ($f' \circ f$) (*under* r a) (*under* r'' (f'(f a)))
by(*auto simp add*: *bij-betw-comp*)
qed

lemma *comp-iso*:
assumes *WELL*: *Well-order* r **and**
EMB: *iso* r r' f **and** *EMB'*: *iso* r' r'' f'
shows *iso* r r'' (f' *o* f)
using *assms* **unfolding** *iso-def*
by (*auto simp add*: *comp-embed bij-betw-comp*)

That *embedS* is also preserved by function composition shall be proved only later.

lemma *embed-Field*:
$[\![$*Well-order* r; *embed* r r' $f]\!] \Longrightarrow f`(Field\ r) \leq Field\ r'$
by (*auto simp add*: *embed-in-Field*)

lemma *embed-preserves-ofilter*:
assumes *WELL*: *Well-order* r **and** *WELL'*: *Well-order* r' **and**
EMB: *embed* r r' f **and** *OF*: *ofilter* r A
shows *ofilter* r' ($f`A$)
proof−

from *WELL* **have** *Well*: *wo-rel* r **unfolding** *wo-rel-def* **.**
from *WELL'* **have** *Well'*: *wo-rel* r' **unfolding** *wo-rel-def* **.**
from *OF* **have** *0*: $A \leq Field\ r$ **by**(*auto simp add*: *Well wo-rel.ofilter-def*)

show *?thesis* **using** *Well' WELL EMB 0 embed-Field*[*of* r r' f]
proof(*unfold wo-rel.ofilter-def*, *auto simp add*: *image-def*)
fix a b'
assume $*$: $a \in A$ **and** $**$: $b' \in under\ r'\ (f\ a)$
hence $a \in Field\ r$ **using** *0* **by** *auto*
hence *bij-betw* f (*under* r a) (*under* r' (f a))
using $*$ *EMB* **by** (*auto simp add*: *embed-def*)
hence $f`(under\ r\ a) = under\ r'\ (f\ a)$
by (*simp add*: *bij-betw-def*)
with $**$ *image-def*[*of* f *under* r a] **obtain** b **where**
1: $b \in under\ r\ a \wedge b' = f\ b$ **by** *blast*
hence $b \in A$ **using** *Well* $*$ *OF*
by (*auto simp add*: *wo-rel.ofilter-def*)
with *1* **show** $\exists\, b \in A.\ b' = f\ b$ **by** *blast*
qed
qed

lemma *embed-Field-ofilter*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
EMB: *embed r r' f*
shows *ofilter r'* (*f'*(*Field r*))
proof−
have *ofilter r* (*Field r*)
using *WELL* **by** (*auto simp add*: *wo-rel-def wo-rel.Field-ofilter*)
with *WELL WELL' EMB*
show *?thesis* **by** (*auto simp add*: *embed-preserves-ofilter*)
qed

lemma *embed-compat*:
assumes *EMB*: *embed r r' f*
shows *compat r r' f*
proof(*unfold compat-def*, *clarify*)
fix *a b*
assume $*$: (*a*,*b*) ∈ *r*
hence *1*: *b* ∈ *Field r* **using** *Field-def*[*of r*] **by** *blast*
have *a* ∈ *under r b*
using $*$ *rel.under-def*[*of r*] **by** *simp*
hence *f a* ∈ *under r'* (*f b*)
using *EMB embed-def*[*of r r' f*]
bij-betw-def[*of f under r b under r'* (*f b*)]
image-def[*of f under r b*] *1* **by** *auto*
thus (*f a*, *f b*) ∈ *r'*
by (*auto simp add*: *rel.under-def*)
qed

lemma *embed-inj-on*:
assumes *WELL*: *Well-order r* **and** *EMB*: *embed r r' f*
shows *inj-on f* (*Field r*)
proof(*unfold inj-on-def*, *clarify*)

from *WELL* **have** *Well*: *wo-rel r* **unfolding** *wo-rel-def* **.**
with *wo-rel.TOTAL*[*of r*]
have *Total*: *Total r* **by** *simp*
from *Well wo-rel.REFL*[*of r*]
have *Refl*: *Refl r* **by** *simp*

fix *a b*
assume $*$: *a* ∈ *Field r* **and** $**$: *b* ∈ *Field r* **and**
$***$: *f a* = *f b*
hence *1*: *a* ∈ *Field r* ∧ *b* ∈ *Field r*
unfolding *Field-def* **by** *auto*
{assume (*a*,*b*) ∈ *r*

hence $a \in$ *under r b* $\wedge$ $b \in$ *under r b*
using *Refl* **by**(*auto simp add*: *rel.under-def refl-on-def*)
hence $a = b$
using *EMB 1* $***$
by (*auto simp add*: *embed-def bij-betw-def inj-on-def*)
}
moreover
{assume $(b,a) \in r$
hence $a \in$ *under r a* $\wedge$ $b \in$ *under r a*
using *Refl* **by**(*auto simp add*: *rel.under-def refl-on-def*)
hence $a = b$
using *EMB 1* $***$
by (*auto simp add*: *embed-def bij-betw-def inj-on-def*)
}
ultimately
show $a = b$ **using** *Total 1*
by (*auto simp add*: *total-on-def*)
qed

lemma *embed-underS*:
assumes *WELL*: *Well-order r* **and** *WELL*: *Well-order r'* **and**
EMB: *embed r r' f* **and** *IN*: $a \in$ *Field r*
shows *bij-betw f* (*underS r a*) (*underS r'* (*f a*))
proof−
have *bij-betw f* (*under r a*) (*under r'* (*f a*))
using *assms* **by** (*auto simp add*: *embed-def*)
moreover
{have *f a* $\in$ *Field r'* **using** *assms* *embed-Field*[*of r r' f*] **by** *auto*
hence *under r a* = *underS r a* $\cup$ $\{a\}$ $\wedge$
under r' (*f a*) = *underS r'* (*f a*) $\cup$ {*f a*}
using *assms* **by** (*auto simp add*: *order-on-defs rel.Refl-under-underS*)
}
moreover
{have $a \notin$ *underS r a* $\wedge$ *f a* $\notin$ *underS r'* (*f a*)
unfolding *rel.underS-def* **by** *blast*
}
ultimately show *?thesis*
by (*auto simp add*: *notIn-Un-bij-betw3*)
qed

lemma *embed-iff-compat-inj-on-ofilter*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
shows *embed r r' f* = (*compat r r' f* $\wedge$ *inj-on f* (*Field r*) $\wedge$ *ofilter r'* (*f'*(*Field r*)))
using *assms*
proof(*auto simp add*: *embed-compat embed-inj-on embed-Field-ofilter*,
unfold embed-def, *auto*)
fix *a*

assume $*$: *inj-on f* (*Field r*) **and**
$**$: *compat r r′ f* **and**
$***$: *ofilter r′* (f‘(*Field r*)) **and**
$****$: $a \in$ *Field r*

have *Well*: *wo-rel r*
using *WELL wo-rel-def*[*of r*] **by** *simp*
hence *Refl*: *Refl r*
using *wo-rel.REFL*[*of r*] **by** *simp*
have *Total*: *Total r*
using *Well wo-rel.TOTAL*[*of r*] **by** *simp*
have *Well′*: *wo-rel r′*
using *WELL′ wo-rel-def*[*of r′*] **by** *simp*
hence *Antisym′*: *antisym r′*
using *wo-rel.ANTISYM*[*of r′*] **by** *simp*
have $(a,a) \in r$
using $****$ *Well wo-rel.REFL*[*of r*]
refl-on-def[*of - r*] **by** *auto*
hence $(f\ a,\ f\ a) \in r'$
using $**$ **by**(*auto simp add*: *compat-def*)
hence *0*: $f\ a \in$ *Field r′*
unfolding *Field-def* **by** *auto*
have $f\ a \in f$‘(*Field r*)
using $****$ **by** *auto*
hence *2*: *under r′* $(f\ a) \leq f$‘(*Field r*)
using *Well′* $***$ *wo-rel.ofilter-def*[*of r′ f*‘(*Field r*)] **by** *fastforce*

show *bij-betw f* (*under r a*) (*under r′* $(f\ a)$)
proof(*unfold bij-betw-def*, *auto*)
  show *inj-on f* (*under r a*)
  using $*$
  by (*auto simp add*: *rel.under-Field subset-inj-on*)
next
  fix *b* **assume** $b \in$ *under r a*
  thus $f\ b \in$ *under r′* $(f\ a)$
  unfolding *rel.under-def* **using** $**$
  by (*auto simp add*: *compat-def*)
next
  fix *b′* **assume** $*****$: $b' \in$ *under r′* $(f\ a)$
  hence $b' \in f$‘(*Field r*)
  using *2* **by** *auto*
  with *Field-def*[*of r*] **obtain** *b* **where**
  3: $b \in$ *Field r* **and** *4*: $b' = f\ b$ **by** *auto*
  have (b,a): *r*
  proof−
    {**assume** $(a,b) \in r$
    with $**$ *4* **have** $(f\ a,\ b')$: *r′*
    by (*auto simp add*: *compat-def*)
    with $*****$ *Antisym′* **have** $f\ a = b'$

by(*auto simp add*: *rel.under-def antisym-def*)
with *3* $****$ *4* $*$ **have** $a = b$
by(*auto simp add*: *inj-on-def*)
}
moreover
{assume $a = b$
hence $(b,a) \in r$ **using** *Refl* $****$ *3*
by (*auto simp add*: *refl-on-def*)
}
ultimately
show *?thesis* **using** *Total* $****$ *3* **by** (*fastforce simp add*: *total-on-def*)
qed
with *4* **show** $b' \in f`(under\ r\ a)$
unfolding *rel.under-def* **by** *auto*
qed
qed

lemma *inv-into-ofilter-embed*:
assumes *WELL*: *Well-order r* **and** *OF*: *ofilter r A* **and**
BIJ: $\forall b \in A$. *bij-betw f* (*under r b*) (*under r′* (*f b*)) **and**
IMAGE: *f ‘ A* = *Field r′*
shows *embed r′ r* (*inv-into A f*)
proof−

have *Well*: *wo-rel r*
using *WELL wo-rel-def*[*of r*] **by** *simp*
have *Refl*: *Refl r*
using *Well wo-rel.REFL*[*of r*] **by** *simp*
have *Total*: *Total r*
using *Well wo-rel.TOTAL*[*of r*] **by** *simp*

have *1*: *bij-betw f A* (*Field r′*)
proof(*unfold bij-betw-def inj-on-def*, *auto simp add*: *IMAGE*)
fix *b1 b2*
assume $*$: $b1 \in A$ **and** $**$: $b2 \in A$ **and**
$***$: $f\,b1 = f\,b2$
have *11*: $b1 \in Field\ r \wedge b2 \in Field\ r$
using $*$ $**$ *Well OF* **by** (*auto simp add*: *wo-rel.ofilter-def*)
moreover
{assume $(b1,b2) \in r$
hence $b1 \in under\ r\ b2 \wedge b2 \in under\ r\ b2$
unfolding *rel.under-def* **using** *11 Refl*
by (*auto simp add*: *refl-on-def*)
hence $b1 = b2$ **using** *BIJ* $*$ $**$ $***$
by (*auto simp add*: *bij-betw-def inj-on-def*)
}
moreover
{assume $(b2,b1) \in r$

hence $b1 \in$ *under r b1* $\wedge$ $b2 \in$ *under r b1*
unfolding *rel.under-def* **using** *11 Refl*
by (*auto simp add*: *refl-on-def*)
hence $b1 = b2$ **using** *BIJ* $*$ $**$ $***$
by (*auto simp add*: *bij-betw-def inj-on-def*)
}
ultimately
show $b1 = b2$
using *Total* **by** (*auto simp add*: *total-on-def*)
qed

let $?f' = $ (*inv-into A f*)

have *2*: $\forall\, b \in A.$ *bij-betw* $?f'$ (*under* r' (f b)) (*under r b*)
proof(*clarify*)
fix *b* **assume** $*$: $b \in A$
hence *under r b* $\leq A$
using *Well OF* **by**(*auto simp add*: *wo-rel.ofilter-def*)
moreover
have f ‘ (*under r b*) = *under* r' (f b)
using $*$ *BIJ* **by** (*auto simp add*: *bij-betw-def*)
ultimately
show *bij-betw* $?f'$ (*under* r' (f b)) (*under r b*)
using *1* **by** (*auto simp add*: *bij-betw-inv-into-subset*)
qed

have *3*: $\forall\, b' \in$ *Field* r'. *bij-betw* $?f'$ (*under* r' b') (*under r* ($?f'$ b'))
proof(*clarify*)
fix b' **assume** $*$: $b' \in$ *Field* r'
have $b' = f$ ($?f'$ b') **using** $*$ *1*
by (*auto simp add*: *bij-betw-inv-into-right*)
moreover
{**obtain** *b* **where** *31*: $b \in A$ **and** $f\ b = b'$ **using** *IMAGE* $*$ **by** *force*
hence $?f'$ $b' = b$ **using** *1* **by** (*auto simp add*: *bij-betw-inv-into-left*)
with *31* **have** $?f'$ $b' \in A$ **by** *auto*
}
ultimately
show *bij-betw* $?f'$ (*under* r' b') (*under r* ($?f'$ b'))
using *2* **by** *auto*
qed

thus *?thesis* **unfolding** *embed-def* **.**
qed

lemma *inv-into-underS-embed*:
assumes *WELL*: *Well-order r* **and**
BIJ: $\forall\, b \in$ *underS r a*. *bij-betw f* (*under r b*) (*under* r' (f b)) **and**
IN: $a \in$ *Field r* **and**

IMAGE: f ‘ (*underS* r a) = *Field* r'
shows *embed* r' r (*inv-into* (*underS* r a) f)
using *assms*
by(*auto simp add*: *wo-rel-def wo-rel.underS-ofilter inv-into-ofilter-embed*)

lemma *inv-into-Field-embed*:
assumes *WELL*: *Well-order* r **and** *EMB*: *embed* r r' f **and**
IMAGE: *Field* $r' \leq f$ ‘ (*Field* r)
shows *embed* r' r (*inv-into* (*Field* r) f)
proof−
have ($\forall b \in$ *Field* r. *bij-betw* f (*under* r b) (*under* r' (f b)))
using *EMB* **by** (*auto simp add*: *embed-def*)
moreover
have f ‘ (*Field* r) $\leq$ *Field* r'
using *EMB WELL* **by** (*auto simp add*: *embed-Field*)
ultimately
show *?thesis* **using** *assms*
by(*auto simp add*: *wo-rel-def wo-rel.Field-ofilter inv-into-ofilter-embed*)
qed

lemma *inv-into-Field-embed-bij-betw*:
assumes *WELL*: *Well-order* r **and**
EMB: *embed* r r' f **and** *BIJ*: *bij-betw* f (*Field* r) (*Field* r')
shows *embed* r' r (*inv-into* (*Field* r) f)
proof−
have *Field* $r' \leq f$ ‘ (*Field* r)
using *BIJ* **by** (*auto simp add*: *bij-betw-def*)
thus *?thesis* **using** *assms*
by(*auto simp add*: *inv-into-Field-embed*)
qed

6.3 Given any two well-orders, one can be embedded in the other

Here is an overview of the proof of of this fact, stated in theorem *wellorders-totally-ordered*:

Fix the well-orders $r::'a$ *rel* and $r'::'a'$ *rel*. Attempt to define an embedding $f::'a \Rightarrow 'a'$ from r to r' in the natural way by well-order recursion ("hoping" that *Field* r turns out to be smaller than *Field* r'), but also record, at the recursive step, in a function $g::'a \Rightarrow$ *bool*, the extra information of whether *Field* r' gets exhausted or not.

If *Field* r' does not get exhausted, then *Field* r is indeed smaller and f is the desired embedding from r to r' (lemma *wellorders-totally-ordered-aux*).

Otherwise, it means that *Field* r' is the smaller one, and the inverse of (the "good" segment of) f is the desired embedding from r' to r (lemma *wellorders-totally-ordered-aux2*).

```
lemma wellorders-totally-ordered-aux:
fixes r ::'a rel  and r'::'a' rel and
      f :: 'a ⇒ 'a' and a::'a
assumes WELL: Well-order r and WELL': Well-order r' and IN: a ∈ Field r
and
       IH: ∀ b ∈ underS r a. bij-betw f (under r b) (under r' (f b)) and
       NOT: f ` (underS r a) ≠ Field r' and SUC: f a = suc r' (f`(underS r a))
shows bij-betw f (under r a) (under r' (f a))
proof−

  have Well: wo-rel r using WELL unfolding wo-rel-def .
  hence Refl: Refl r using wo-rel.REFL[of r] by auto
  have Trans: trans r using Well wo-rel.TRANS[of r] by auto
  have Well': wo-rel r' using WELL' unfolding wo-rel-def .
  have OF: ofilter r (underS r a)
  by (auto simp add: Well wo-rel.underS-ofilter)
  hence UN: underS r a = (⋃  b ∈ underS r a. under r b)
  using Well wo-rel.ofilter-under-UNION[of r underS r a] by blast

  {fix b assume *: b ∈ underS r a
   hence t0: (b,a) ∈ r ∧ b ≠ a unfolding rel.underS-def by auto
   have t1: b ∈ Field r
   using * rel.underS-Field[of r a] by auto
   have t2: f`(under r b) = under r' (f b)
   using IH * by (auto simp add: bij-betw-def)
   hence t3: ofilter r' (f`(under r b))
   using Well' by (auto simp add: wo-rel.under-ofilter)
   have f`(under r b) ≤ Field r'
   using t2 by (auto simp add: rel.under-Field)
   moreover
   have b ∈ under r b
   using t1 by(auto simp add: Refl rel.Refl-under-in)
   ultimately
   have t4:  f b ∈ Field r' by auto
   have f`(under r b) = under r' (f b) ∧
         ofilter r' (f`(under r b)) ∧
         f b ∈ Field r'
   using t2 t3 t4 by auto
  }
  hence bFact:
  ∀ b ∈ underS r a. f`(under r b) = under r' (f b) ∧
                       ofilter r' (f`(under r b)) ∧
                       f b ∈ Field r' by blast

  have subField: f`(underS r a) ≤ Field r'
  using bFact by blast

  have OF': ofilter r' (f`(underS r a))
  proof−
```

have $f`(underS\ r\ a) = f`(\bigcup\ \ b \in underS\ r\ a.\ under\ r\ b)$
using *UN* **by** *auto*
also have $\ldots = (\bigcup\ \ b \in underS\ r\ a.\ f`(under\ r\ b))$ **by** *blast*
also have $\ldots = (\bigcup\ \ b \in underS\ r\ a.\ (under\ r'\ (f\ b)))$
using *bFact* **by** *auto*
finally
have $f`(underS\ r\ a) = (\bigcup\ \ b \in underS\ r\ a.\ (under\ r'\ (f\ b)))$ **.**
thus *?thesis*
using *Well′ bFact*
wo-rel.ofilter-UNION[*of* r' *underS* $r\ a\ \lambda\ b.\ under\ r'\ (f\ b)$] **by** *fastforce*
qed

have $f`(underS\ r\ a) \cup AboveS\ r'\ (f`(underS\ r\ a)) = Field\ r'$
using *Well′ OF′* **by** (*auto simp add*: *wo-rel.ofilter-AboveS-Field*)
hence *NE*: $AboveS\ r'\ (f`(underS\ r\ a)) \neq \{\}$
using *subField NOT* **by** *blast*

have *INCL1*: $f`(underS\ r\ a) \leq underS\ r'\ (f\ a)$
proof(*auto*)
fix b **assume** $*$: $b \in underS\ r\ a$
have $f\ b \neq f\ a \wedge (f\ b, f\ a) \in r'$
using *subField Well′ SUC NE* $*$
wo-rel.suc-greater[*of* $r'\ f`(underS\ r\ a)\ f\ b$] **by** *auto*
thus $f\ b \in underS\ r'\ (f\ a)$
unfolding *rel.underS-def* **by** *simp*
qed

have *INCL2*: $underS\ r'\ (f\ a) \leq f`(underS\ r\ a)$
proof
fix b' **assume** $b' \in underS\ r'\ (f\ a)$
hence $b' \neq f\ a \wedge (b', f\ a) \in r'$
unfolding *rel.underS-def* **by** *simp*
thus $b' \in f`(underS\ r\ a)$
using *Well′ SUC NE OF′*
wo-rel.suc-ofilter-in[*of* $r'\ f\ `\ underS\ r\ a\ b'$] **by** *auto*
qed

have *INJ*: *inj-on* f $(underS\ r\ a)$
proof−
have $\forall\, b \in underS\ r\ a.$ *inj-on* $f\ (under\ r\ b)$
using *IH* **by** (*auto simp add*: *bij-betw-def*)
moreover
have $\forall\, b.$ *ofilter* $r\ (under\ r\ b)$
using *Well* **by** (*auto simp add*: *wo-rel.under-ofilter*)
ultimately show *?thesis*
using *WELL bFact UN*
UNION-inj-on-ofilter[*of* r *underS* $r\ a\ \lambda b.\ under\ r\ b\ f$]
by *auto*
qed

```
  have BIJ: bij-betw f (underS r a) (underS r' (f a))
  unfolding bij-betw-def
  using INJ INCL1 INCL2 by auto

  have f a ∈ Field r'
  using Well' subField NE SUC
  by (auto simp add: wo-rel.suc-inField)
  thus ?thesis
  using WELL WELL' IN BIJ under-underS-bij-betw[of r r' a f] by auto
qed


lemma wellorders-totally-ordered-aux2:
fixes r ::'a rel  and r'::'a' rel and
      f :: 'a ⇒ 'a' and g :: 'a ⇒ bool  and a::'a
assumes WELL: Well-order r and WELL': Well-order r' and
MAIN1:
  ⋀ a. (False ∉ g‘(underS r a) ∧ f‘(underS r a) ≠ Field r'
          ⟶ f a = suc r' (f‘(underS r a)) ∧ g a = True)
         ∧
         (¬(False ∉ (g‘(underS r a)) ∧ f‘(underS r a) ≠ Field r')
          ⟶ g a = False) and
MAIN2: ⋀ a. a ∈ Field r ∧ False ∉ g‘(under r a) ⟶
             bij-betw f (under r a) (under r' (f a)) and
Case: a ∈ Field r ∧ False ∈ g‘(under r a)
shows ∃f'. embed r' r f'
proof−
  have Well: wo-rel r using WELL unfolding wo-rel-def .
  hence Refl: Refl r using wo-rel.REFL[of r] by auto
  have Trans: trans r using Well wo-rel.TRANS[of r] by auto
  have Antisym: antisym r using Well wo-rel.ANTISYM[of r] by auto
  have Well': wo-rel r' using WELL' unfolding wo-rel-def .

  have 0: under r a = underS r a ∪ {a}
  using Refl Case by(auto simp add: rel.Refl-under-underS)

  have 1: g a = False
  proof−
    {assume g a ≠ False
     with 0 Case have False ∈ g‘(underS r a) by blast
     with MAIN1 have g a = False by blast}
    thus ?thesis by blast
  qed
  let ?A = {a ∈ Field r. g a = False}
  let ?a = (minim r ?A)

  have 2: ?A ≠ {} ∧ ?A ≤ Field r using Case 1 by blast
```

  have *3*: $False \notin g`(underS\ r\ ?a)$
  proof
    assume $False \in g`(underS\ r\ ?a)$
    then obtain *b* **where** $b \in underS\ r\ ?a$ **and** *31*: $g\ b = False$ **by** *auto*
    hence *32*: $(b,?a) \in r \wedge b \neq ?a$
    by (*auto simp add*: *rel.underS-def*)
    hence $b \in Field\ r$ **unfolding** *Field-def* **by** *auto*
    with *31* **have** $b \in ?A$ **by** *auto*
    hence $(?a,b) \in r$ **using** *wo-rel.minim-least 2 Well* **by** *fastforce*

    with *32 Antisym* **show** *False*
    by (*auto simp add*: *antisym-def*)
  qed
  have *temp*: $?a \in ?A$
  using *Well 2 wo-rel.minim-in*[*of r ?A*] **by** *auto*
  hence *4*: $?a \in Field\ r$ **by** *auto*

  have *5*: $g\ ?a = False$ **using** *temp* **by** *blast*

  have *6*: $f`(underS\ r\ ?a) = Field\ r'$
  using *MAIN1*[*of ?a*] *3 5* **by** *blast*

  have *7*: $\forall\, b \in underS\ r\ ?a.\ bij\text{-}betw\ f\ (under\ r\ b)\ (under\ r'\ (f\ b))$
  proof
    fix *b* **assume** *as*: $b \in underS\ r\ ?a$
    moreover
    have *ofilter r* (*underS r ?a*)
    using *Well* **by** (*auto simp add*: *wo-rel.underS-ofilter*)
    ultimately
   have $False \notin g`(under\ r\ b)$ **using** *3 Well* **by** (*auto simp add*: *wo-rel.ofilter-def*)
    moreover have $b \in Field\ r$
    unfolding *Field-def* **using** *as* **by** (*auto simp add*: *rel.underS-def*)
    ultimately
    show $bij\text{-}betw\ f\ (under\ r\ b)\ (under\ r'\ (f\ b))$
    using *MAIN2* **by** *auto*
  qed

  have $embed\ r'\ r\ (inv\text{-}into\ (underS\ r\ ?a)\ f)$
  using *WELL WELL′ 7 4 6 inv-into-underS-embed*[*of r ?a f r′*] **by** *auto*
  thus *?thesis*
  unfolding *embed-def* **by** *blast*
qed

theorem *wellorders-totally-ordered*:
fixes $r :: 'a\ rel$ **and** $r' :: 'a'\ rel$
assumes *WELL*: *Well-order r* **and** *WELL′*: *Well-order r′*
shows $(\exists f.\ embed\ r\ r'\ f) \vee (\exists f'.\ embed\ r'\ r\ f')$
proof−

```
have Well: wo-rel r using WELL unfolding wo-rel-def .
hence Refl: Refl r using wo-rel.REFL[of r] by auto
have Trans: trans r using Well wo-rel.TRANS[of r] by auto
have Well': wo-rel r' using WELL' unfolding wo-rel-def .

obtain H where H-def: H =
(λh a. if False ∉ (snd o h)`(underS r a) ∧ (fst o h)`(underS r a) ≠ Field r'
            then (suc r' ((fst o h)`(underS r a)), True)
            else (undefined, False)) by blast
have Adm: adm-wo r H
using Well
proof(unfold wo-rel.adm-wo-def, clarify)
  fix h1::'a ⇒ 'a' * bool and h2::'a ⇒ 'a' * bool and x
  assume ∀ y∈underS r x. h1 y = h2 y
  hence ∀ y∈underS r x. (fst o h1) y = (fst o h2) y ∧
                        (snd o h1) y = (snd o h2) y by auto
  hence (fst o h1)`(underS r x) = (fst o h2)`(underS r x) ∧
        (snd o h1)`(underS r x) = (snd o h2)`(underS r x)
  by (auto simp add: image-def)
  thus H h1 x = H h2 x by (simp add: H-def)
qed

obtain h::'a ⇒ 'a' * bool and f::'a ⇒ 'a' and g::'a ⇒ bool
where h-def: h = worec r H and
      f-def: f = fst o h and g-def: g = snd o h by blast
obtain test where test-def:
test = (λ a. False ∉ (g`(underS r a)) ∧ f`(underS r a) ≠ Field r') by blast

have *: ⋀ a. h a  = H h a
using Adm Well wo-rel.worec-fixpoint[of r H] by (simp add: h-def)
have Main1:
⋀ a. (test a ⟶ f a = suc r' (f`(underS r a)) ∧ g a = True) ∧
      (¬(test a) ⟶ g a = False)
proof−
  fix a show (test a ⟶ f a = suc r' (f`(underS r a)) ∧ g a = True) ∧
            (¬(test a) ⟶ g a = False)
  using *[of a] test-def f-def g-def H-def by auto
qed

let ?phi = λ a. a ∈ Field r ∧ False ∉ g`(under r a) ⟶
                bij-betw f (under r a) (under r' (f a))
have Main2: ⋀ a. ?phi a
proof−
  fix a show ?phi a
  proof(rule wo-rel.well-order-induct[of r ?phi],
        simp only: Well, clarify)
    fix a
    assume IH: ∀ b. b ≠ a ∧ (b,a) ∈ r ⟶ ?phi b and
```

$*$: $a \in$ *Field r* **and**
$**$: *False* $\notin$ *g*‘(*under r a*)
have *1*: $\forall\, b \in$ *underS r a*. *bij-betw f* (*under r b*) (*under r′* (*f b*))
proof(*clarify*)
fix *b* **assume** $***$: $b \in$ *underS r a*
hence *0*: $(b,a) \in r \wedge b \neq a$ **unfolding** *rel.underS-def* **by** *auto*
moreover have $b \in$ *Field r*
using $***$ *rel.underS-Field*[*of r a*] **by** *auto*
moreover have *False* $\notin$ *g*‘(*under r b*)
using *0* $**$ *Trans rel.under-incr*[*of r b a*] **by** *auto*
ultimately show *bij-betw f* (*under r b*) (*under r′* (*f b*))
using *IH* **by** *auto*
qed

have *21*: *False* $\notin$ *g*‘(*underS r a*)
using $**$ *rel.underS-subset-under*[*of r a*] **by** *auto*
have *22*: *g*‘(*under r a*) $\leq$ {*True*} **using** $**$ **by** *auto*
moreover have *23*: $a \in$ *under r a*
using *Refl* $*$ **by** (*auto simp add*: *rel.Refl-under-in*)
ultimately have *24*: *g a* = *True* **by** *blast*
have *2*: *f*‘(*underS r a*) $\neq$ *Field r′*
proof
assume *f*‘(*underS r a*) = *Field r′*
hence *g a* = *False* **using** *Main1 test-def* **by** *blast*
with *24* **show** *False* **using** $**$ **by** *blast*
qed

have *3*: *f a* = *suc r′* (*f*‘(*underS r a*))
using *21 2 Main1 test-def* **by** *blast*

show *bij-betw f* (*under r a*) (*under r′* (*f a*))
using *WELL WELL′ 1 2 3* $*$
wellorders-totally-ordered-aux[*of r r′ a f*] **by** *auto*
qed
qed

let *?chi* = (λ *a*. $a \in$ *Field r* $\wedge$ *False* $\in$ *g*‘(*under r a*))
show *?thesis*
proof(*cases* $\exists$ *a*. *?chi a*)
assume $\neg$ ($\exists$ *a*. *?chi a*)
hence $\forall\, a \in$ *Field r*. *bij-betw f* (*under r a*) (*under r′* (*f a*))
using *Main2* **by** *blast*
thus *?thesis* **unfolding** *embed-def* **by** *blast*
next
assume $\exists$ *a*. *?chi a*
then obtain *a* **where** *?chi a* **by** *blast*
hence $\exists$ *f′*. *embed r′ r f′*
using *wellorders-totally-ordered-aux2*[*of r r′ g f a*]
WELL WELL′ Main1 Main2 test-def **by** *blast*

thus *?thesis* **by** *blast*
qed
qed

corollary *one-set-greater*:
$(\exists f{::}'a \Rightarrow 'a'.\ f\ `\ A \leq A' \wedge \mathit{inj\text{-}on}\ f\ A) \vee (\exists g{::}'a' \Rightarrow 'a.\ g\ `\ A' \leq A \wedge \mathit{inj\text{-}on}\ g\ A')$
proof−
obtain *r* **where** *well-order-on A r* **by** (*fastforce simp add*: *well-order-on*)
hence *1*: $A = \mathit{Field}\ r \wedge \mathit{Well\text{-}order}\ r$
using *rel.well-order-on-Well-order* **by** *auto*
obtain r' **where** *2*: *well-order-on* A' r' **by** (*fastforce simp add*: *well-order-on*)
hence *2*: $A' = \mathit{Field}\ r' \wedge \mathit{Well\text{-}order}\ r'$
using *rel.well-order-on-Well-order* **by** *auto*
hence $(\exists f.\ \mathit{embed}\ r\ r'\ f) \vee (\exists g.\ \mathit{embed}\ r'\ r\ g)$
using *1 2* **by** (*auto simp add*: *wellorders-totally-ordered*)
moreover
{**fix** *f* **assume** *embed* $r\ r'\ f$
hence $f`A \leq A' \wedge \mathit{inj\text{-}on}\ f\ A$
using *1 2* **by** (*auto simp add*: *embed-Field embed-inj-on*)
}
moreover
{**fix** *g* **assume** *embed* $r'\ r\ g$
hence $g`A' \leq A \wedge \mathit{inj\text{-}on}\ g\ A'$
using *1 2* **by** (*auto simp add*: *embed-Field embed-inj-on*)
}
ultimately show *?thesis* **by** *blast*
qed

corollary *one-type-greater*:
$(\exists f{::}'a \Rightarrow 'a'.\ \mathit{inj}\ f) \vee (\exists g{::}'a' \Rightarrow 'a.\ \mathit{inj}\ g)$
using *one-set-greater*[*of UNIV UNIV*] **by** *auto*

6.4 Uniqueness of embeddings

Here we show a fact complementary to the one from the previous subsection – namely, that between any two well-orders there is *at most* one embedding, and is the one definable by the expected well-order recursive equation. As a consequence, any two embeddings of opposite directions are mutually inverse.

lemma *embed-determined*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order* r' **and**
EMB: *embed* $r\ r'\ f$ **and** *IN*: $a \in \mathit{Field}\ r$
shows $f\ a = \mathit{suc}\ r'\ (f`(\mathit{underS}\ r\ a))$
proof−
have *bij-betw f* (*underS r a*) (*underS* r' (*f a*))
using *assms* **by** (*auto simp add*: *embed-underS*)

hence $f`(underS\ r\ a) = underS\ r'\ (f\ a)$
by (*auto simp add*: *bij-betw-def*)
moreover
{**have** $f\ a \in Field\ r'$ **using** *IN*
using *EMB WELL embed-Field*[*of r r' f*] **by** *auto*
hence $f\ a = suc\ r'\ (underS\ r'\ (f\ a))$
using *WELL'* **by** (*auto simp add*: *wo-rel-def wo-rel.suc-underS*)
}
ultimately show *?thesis* **by** *simp*
qed

lemma *embed-unique*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
EMBf: *embed r r' f* **and** *EMBg*: *embed r r' g*
shows $a \in Field\ r \longrightarrow f\ a = g\ a$
proof(*rule wo-rel.well-order-induct*[*of r*], *auto simp add*: *WELL wo-rel-def*)
fix *a*
assume *IH*: $\forall b.\ b \neq a \wedge (b,a): r \longrightarrow b \in Field\ r \longrightarrow f\ b = g\ b$ **and**
$*$: $a \in Field\ r$
hence $\forall b \in underS\ r\ a.\ f\ b = g\ b$
unfolding *rel.underS-def* **by** (*auto simp add*: *Field-def*)
hence $f`(underS\ r\ a) = g`(underS\ r\ a)$ **by** *force*
thus $f\ a = g\ a$
using *assms* $*$ *embed-determined*[*of r r' f a*] *embed-determined*[*of r r' g a*] **by** *auto*
qed

lemma *embed-bothWays-inverse*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
EMB: *embed r r' f* **and** *EMB'*: *embed r' r f'*
shows $(\forall a \in Field\ r.\ f'(f\ a) = a) \wedge (\forall a' \in Field\ r'.\ f(f'\ a') = a')$
proof−
have *embed r r* $(f'\ o\ f)$ **using** *assms*
by(*auto simp add*: *comp-embed*)
moreover have *embed r r id* **using** *assms*
by (*auto simp add*: *id-embed*)
ultimately have $\forall a \in Field\ r.\ f'(f\ a) = a$
using *assms embed-unique*[*of r r f' o f id*] *id-def* **by** *auto*
moreover
{**have** *embed* $r'\ r'\ (f\ o\ f')$ **using** *assms*
by(*auto simp add*: *comp-embed*)
moreover have *embed r' r' id* **using** *assms*
by (*auto simp add*: *id-embed*)
ultimately have $\forall a' \in Field\ r'.\ f(f'\ a') = a'$
using *assms embed-unique*[*of r' r' f o f' id*] *id-def* **by** *auto*
}
ultimately show *?thesis* **by** *blast*

qed

6.5 More properties of embeddings, strict embeddings and isomorphisms

lemma *embed-bothWays-Field-bij-betw*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
EMB: *embed r r' f* **and** *EMB'*: *embed r' r f'*
shows *bij-betw f (Field r) (Field r')*
proof−
have $(\forall a \in \mathit{Field}\ r.\ f'(f\ a) = a) \wedge (\forall a' \in \mathit{Field}\ r'.\ f(f'\ a') = a')$
using *assms* **by** (*auto simp add*: *embed-bothWays-inverse*)
moreover
have $f`(\mathit{Field}\ r) \leq \mathit{Field}\ r' \wedge f'\ `\ (\mathit{Field}\ r') \leq \mathit{Field}\ r$
using *assms* **by** (*auto simp add*: *embed-Field*)
ultimately
show *?thesis* **using** *bij-betw-byWitness*[*of Field r f' f Field r'*] **by** *auto*
qed

lemma *embedS-comp-embed*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and** *WELL''*: *Well-order r''*
and *EMB*: *embedS r r' f* **and** *EMB'*: *embed r' r'' f'*
shows *embedS r r'' (f' o f)*
proof−
let *?g* = (*f' o f*) **let** *?h* = *inv-into* (*Field r*) *?g*
have *1*: $\mathit{embed}\ r\ r'\ f \wedge \neg\ (\mathit{bij\text{-}betw}\ f\ (\mathit{Field}\ r)\ (\mathit{Field}\ r'))$
using *EMB* **by** (*auto simp add*: *embedS-def*)
hence *2*: *embed r r'' ?g*
using *WELL EMB' comp-embed*[*of r r' f r'' f'*] **by** *auto*
moreover
{**assume** *bij-betw ?g (Field r) (Field r'')*
hence *embed r'' r ?h* **using** *2 WELL*
by (*auto simp add*: *inv-into-Field-embed-bij-betw*)
hence *embed r' r (?h o f')* **using** *WELL' EMB'*
by (*auto simp add*: *comp-embed*)
hence *bij-betw f (Field r) (Field r')* **using** *WELL WELL' 1*
by (*auto simp add*: *embed-bothWays-Field-bij-betw*)
with *1* **have** *False* **by** *blast*
}
ultimately show *?thesis* **unfolding** *embedS-def* **by** *auto*
qed

lemma *embed-comp-embedS*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and** *WELL''*: *Well-order r''*
and *EMB*: *embed r r' f* **and** *EMB'*: *embedS r' r'' f'*

shows *embedS r r''* $(f'$ *o* $f)$
proof−
let *?g* = $(f'$ *o* $f)$ **let** *?h* = *inv-into* (*Field r*) *?g*
have *1*: *embed r' r'' f'* ∧ ¬ (*bij-betw f'* (*Field r'*) (*Field r''*))
using *EMB'* **by** (*auto simp add*: *embedS-def*)
hence *2*: *embed r r'' ?g*
using *WELL EMB comp-embed*[*of r r' f r'' f'*] **by** *auto*
moreover
{**assume** *bij-betw ?g* (*Field r*) (*Field r''*)
hence *embed r'' r ?h* **using** *2 WELL*
by (*auto simp add*: *inv-into-Field-embed-bij-betw*)
hence *embed r'' r'* (*f o ?h*) **using** *WELL'' EMB*
by (*auto simp add*: *comp-embed*)
hence *bij-betw f'* (*Field r'*) (*Field r''*) **using** *WELL' WELL'' 1*
by (*auto simp add*: *embed-bothWays-Field-bij-betw*)
with *1* **have** *False* **by** *blast*
}
ultimately show *?thesis* **unfolding** *embedS-def* **by** *auto*
qed

lemma *comp-embedS*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and** *WELL''*: *Well-order r''*
and *EMB*: *embedS r r' f* **and** *EMB'*: *embedS r' r'' f'*
shows *embedS r r''* (*f' o f*)
proof−
have *embed r' r'' f'* **using** *EMB'* **unfolding** *embedS-def* **by** *simp*
thus *?thesis* **using** *assms* **by** (*auto simp add*: *embedS-comp-embed*)
qed

lemma *embed-comp-iso*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and** *WELL''*: *Well-order r''*
and *EMB*: *embed r r' f* **and** *EMB'*: *iso r' r'' f'*
shows *embed r r''* (*f' o f*)
using *assms* **unfolding** *iso-def*
by (*auto simp add*: *comp-embed*)

lemma *iso-comp-embed*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and** *WELL''*: *Well-order r''*
and *EMB*: *iso r r' f* **and** *EMB'*: *embed r' r'' f'*
shows *embed r r''* (*f' o f*)
using *assms* **unfolding** *iso-def*
by (*auto simp add*: *comp-embed*)

lemma *embedS-comp-iso*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and** *WELL''*: *Well-order r''*
and *EMB*: *embedS r r' f* **and** *EMB'*: *iso r' r'' f'*
shows *embedS r r''* $(f' \ o \ f)$
using *assms* **unfolding** *iso-def*
by (*auto simp add*: *embedS-comp-embed*)

lemma *iso-comp-embedS*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and** *WELL''*: *Well-order r''*
and *EMB*: *iso r r' f* **and** *EMB'*: *embedS r' r'' f'*
shows *embedS r r''* $(f' \ o \ f)$
using *assms* **unfolding** *iso-def* **using** *embed-comp-embedS*
by (*auto simp add*: *embed-comp-embedS*)

lemma *embedS-Field*:
assumes *WELL*: *Well-order r* **and** *EMB*: *embedS r r' f*
shows *f* ‘ (*Field r*) < *Field r'*
proof−
have *f*‘(*Field r*) ≤ *Field r'* **using** *assms*
by (*auto simp add*: *embed-Field embedS-def*)
moreover
{**have** *inj-on f* (*Field r*) **using** *assms*
by (*auto simp add*: *embedS-def embed-inj-on*)
hence *f*‘(*Field r*) ≠ *Field r'* **using** *EMB*
by (*auto simp add*: *embedS-def bij-betw-def*)
}
ultimately show *?thesis* **by** *blast*
qed

lemma *embedS-iff*:
assumes *WELL*: *Well-order r* **and** *ISO*: *embed r r' f*
shows *embedS r r' f* = (*f* ‘ (*Field r*) < *Field r'*)
proof
assume *embedS r r' f*
thus *f* ‘ *Field r* ⊂ *Field r'*
using *WELL* **by** (*auto simp add*: *embedS-Field*)
next
assume *f* ‘ *Field r* ⊂ *Field r'*
hence ¬ *bij-betw f* (*Field r*) (*Field r'*)
unfolding *bij-betw-def* **by** *blast*
thus *embedS r r' f* **unfolding** *embedS-def*
using *ISO* **by** *auto*
qed

lemma *iso-Field*:
iso r r' f $\Longrightarrow$ f ‘ (*Field* r) = *Field* r'
using *assms* **by** (*auto simp add*: *iso-def bij-betw-def*)

lemma *iso-iff*:
assumes *Well-order* r
shows *iso* r r' f = (*embed* r r' f $\wedge$ f ‘ (*Field* r) = *Field* r')
proof
  assume *iso* r r' f
  thus *embed* r r' f $\wedge$ f ‘ (*Field* r) = *Field* r'
  by (*auto simp add*: *iso-Field iso-def*)
next
  assume $*$: *embed* r r' f $\wedge$ f ‘ *Field* r = *Field* r'
  hence *inj-on* f (*Field* r) **using** *assms* **by** (*auto simp add*: *embed-inj-on*)
  with $*$ **have** *bij-betw* f (*Field* r) (*Field* r')
  unfolding *bij-betw-def* **by** *simp*
  with $*$ **show** *iso* r r' f **unfolding** *iso-def* **by** *auto*
qed

lemma *iso-iff2*:
assumes *Well-order* r
shows *iso* r r' f = (*bij-betw* f (*Field* r) (*Field* r') $\wedge$
          ($\forall$ a $\in$ *Field* r. $\forall$ b $\in$ *Field* r.
            (((a,b) $\in$ r) = ((f a, f b) $\in$ r'))))
using *assms*
proof(*auto simp add*: *iso-def*)
  fix a b
  assume *embed* r r' f
  hence *compat* r r' f **using** *embed-compat*[*of* r] **by** *auto*
  moreover assume (a,b) $\in$ r
  ultimately show (f a, f b) $\in$ r' **using** *compat-def*[*of* r] **by** *auto*
next
  let *?f'* = *inv-into* (*Field* r) f
  assume *embed* r r' f **and** *1*: *bij-betw* f (*Field* r) (*Field* r')
  hence *embed* r' r *?f'* **using** *assms*
  by (*auto simp add*: *inv-into-Field-embed-bij-betw*)
  hence *2*: *compat* r' r *?f'* **using** *embed-compat*[*of* r'] **by** *auto*
  fix a b **assume** $*$: a $\in$ *Field* r b $\in$ *Field* r **and** $**$: (f a,f b) $\in$ r'
  hence *?f'*(f a) = a $\wedge$ *?f'*(f b) = b **using** *1*
  by (*auto simp add*: *bij-betw-inv-into-left*)
  thus (a,b) $\in$ r **using** $**$ *2* *compat-def*[*of* r' r *?f'*] **by** *fastforce*
next
  assume $*$: *bij-betw* f (*Field* r) (*Field* r') **and**
      $**$: $\forall$ $a$$\in$*Field* r. $\forall$ $b$$\in$*Field* r. ((a, b) $\in$ r) = ((f a, f b) $\in$ r')
  have *1*: $\bigwedge$ a. *under* r a $\le$ *Field* r $\wedge$ *under* r' (f a) $\le$ *Field* r'

```
  by (auto simp add: rel.under-Field)
  have 2: inj-on f (Field r) using ∗ by (auto simp add: bij-betw-def)
  {fix a assume ∗∗∗: a ∈ Field r
   have bij-betw f (under r a) (under r′ (f a))
   proof(unfold bij-betw-def, auto)
     show inj-on f (under r a)
     using 1 2 by (auto simp add: subset-inj-on)
   next
     fix b assume b ∈ under r a
     hence a ∈ Field r ∧ b ∈ Field r ∧ (b,a) ∈ r
     unfolding rel.under-def by (auto simp add: Field-def Range-def Domain-def)
     with 1 ∗∗ show f b ∈ under r′ (f a)
     unfolding rel.under-def by auto
   next
     fix b′ assume b′ ∈ under r′ (f a)
     hence 3: (b′,f a) ∈ r′ unfolding rel.under-def by simp
     hence b′ ∈ Field r′ unfolding Field-def by auto
     with ∗ obtain b where b ∈ Field r ∧ f b = b′
     unfolding bij-betw-def by force
     with 3 ∗∗ ∗∗∗
     show b′ ∈ f ‘ (under r a) unfolding rel.under-def by blast
   qed
  }
  thus embed r r′ f unfolding embed-def using ∗ by auto
qed


lemma iso-iff3:
assumes WELL: Well-order r and WELL′: Well-order r′
shows iso r r′ f = (bij-betw f (Field r) (Field r′) ∧ compat r r′ f)
proof
  assume iso r r′ f
  thus bij-betw f (Field r) (Field r′) ∧ compat r r′ f
  unfolding compat-def using WELL by (auto simp add: iso-iff2 Field-def)
next
  have Well: wo-rel r ∧ wo-rel r′ using WELL WELL′
  by (auto simp add: wo-rel-def)
  assume ∗: bij-betw f (Field r) (Field r′) ∧ compat r r′ f
  thus iso r r′ f
  unfolding compat-def using assms
  proof(auto simp add: iso-iff2)
    fix a b assume ∗∗: a ∈ Field r b ∈ Field r and
                   ∗∗∗: (f a, f b) ∈ r′
    {assume (b,a) ∈ r ∨ b = a
     hence (b,a): rusing Well ∗∗ wo-rel.REFL[of r] refl-on-def[of - r] by blast
     hence (f b, f a) ∈ r′ using ∗ unfolding compat-def by auto
     hence f a = f b
     using Well ∗∗∗ wo-rel.ANTISYM[of r′] antisym-def[of r′] by blast
     hence a = b using ∗ ∗∗ unfolding bij-betw-def inj-on-def by auto
```

```
    hence (a,b) ∈ r using Well ∗∗ wo-rel.REFL[of r] refl-on-def[of - r] by blast
   }
   thus (a,b) ∈ r
   using Well ∗∗ wo-rel.TOTAL[of r] total-on-def[of - r] by blast
 qed
qed
```

```
lemma embed-bothWays-bij-betw:
assumes WELL: Well-order r and WELL′: Well-order r′ and
        EMB: embed r r′ f and EMB′: embed r′ r g
shows bij-betw f (Field r) (Field r′)
proof−
  let ?A = Field r let ?A′ = Field r′
  have embed r r (g o f) ∧ embed r′ r′ (f o g)
  using assms by (auto simp add: comp-embed)
  hence 1: (∀ a ∈ ?A. g(f a) = a) ∧ (∀ a′ ∈ ?A′. f(g a′) = a′)
  using WELL id-embed[of r] embed-unique[of r r g o f id]
        WELL′ id-embed[of r′] embed-unique[of r′ r′ f o g id]
        id-def by auto
  have 2: (∀ a ∈ ?A. f a ∈ ?A′) ∧ (∀ a′ ∈ ?A′. g a′ ∈ ?A)
  using assms embed-Field[of r r′ f] embed-Field[of r′ r g] by blast

  show ?thesis
  proof(unfold bij-betw-def inj-on-def, auto simp add: 2)
    fix a b assume ∗: a ∈ ?A b ∈ ?A and ∗∗: f a = f b
    have a = g(f a) ∧ b = g(f b) using ∗ 1 by auto
    with ∗∗ show a = b by auto
  next
    fix a′ assume ∗: a′ ∈ ?A′
    hence g a′ ∈ ?A ∧ f(g a′) = a′ using 1 2 by auto
    thus a′ ∈ f ‘ ?A by force
  qed
qed
```

```
lemma embed-bothWays-iso:
assumes WELL: Well-order r and WELL′: Well-order r′ and
        EMB: embed r r′ f and EMB′: embed r′ r g
shows iso r r′ f
unfolding iso-def using assms by (auto simp add: embed-bothWays-bij-betw)
```

```
lemma iso-iff4:
assumes WELL: Well-order r and WELL′: Well-order r′
shows iso r r′ f = (embed r r′ f ∧ embed r′ r (inv-into (Field r) f))
using assms embed-bothWays-iso
by(unfold iso-def, auto simp add: inv-into-Field-embed-bij-betw)
```

lemma *embed-embedS-iso*:
embed r r' f = (*embedS* r r' f ∨ *iso* r r' f)
unfolding *embedS-def iso-def* **by** *blast*

lemma *not-embedS-iso*:
¬ (*embedS* r r' f ∧ *iso* r r' f)
unfolding *embedS-def iso-def* **by** *blast*

lemma *embed-embedS-iff-not-iso*:
assumes *embed* r r' f
shows *embedS* r r' f = (¬ *iso* r r' f)
using *assms* **unfolding** *embedS-def iso-def* **by** *blast*

lemma *iso-inv-into*:
assumes *WELL*: *Well-order* r **and** *ISO*: *iso* r r' f
shows *iso* r' r (*inv-into* (*Field* r) f)
using *assms* **unfolding** *iso-def*
using *bij-betw-inv-into inv-into-Field-embed-bij-betw* **by** *blast*

lemma *embedS-or-iso*:
assumes *WELL*: *Well-order* r **and** *WELL'*: *Well-order* r'
shows (∃ g. *embedS* r r' g) ∨ (∃ h. *embedS* r' r h) ∨ (∃ f. *iso* r r' f)
proof−
  {**fix** f **assume** ∗: *embed* r r' f
   {**assume** *bij-betw* f (*Field* r) (*Field* r')
    hence *?thesis* **using** ∗ **by** (*auto simp add*: *iso-def*)
   }
   moreover
   {**assume** ¬ *bij-betw* f (*Field* r) (*Field* r')
    hence *?thesis* **using** ∗ **by** (*auto simp add*: *embedS-def*)
   }
   ultimately have *?thesis* **by** *auto*
  }
  moreover
  {**fix** f **assume** ∗: *embed* r' r f
   {**assume** *bij-betw* f (*Field* r') (*Field* r)
    hence *iso* r' r f **using** ∗ **by** (*auto simp add*: *iso-def*)
    hence *iso* r r' (*inv-into* (*Field* r') f)
    using *WELL'* **by** (*auto simp add*: *iso-inv-into*)
    hence *?thesis* **by** *blast*
   }
   moreover
   {**assume** ¬ *bij-betw* f (*Field* r') (*Field* r)
    hence *?thesis* **using** ∗ **by** (*auto simp add*: *embedS-def*)

}
ultimately have *?thesis* **by** *auto*
}
ultimately show *?thesis* **using** *WELL WELL'*
wellorders-totally-ordered[*of r r'*] **by** *blast*
qed

end

7 Constructions on wellorders

theory *Constructions-on-Wellorders* **imports** *Wellorder-Embedding*
begin

In this section, we study basic constructions on well-orders, such as restriction to a set/order filter, copy via direct images, ordinal-like sum of disjoint well-orders, and bounded square. We also define between well-orders the relations *ordLeq*, of being embedded (abbreviated $\leq o$), *ordLess*, of being strictly embedded (abbreviated $<o$), and *ordIso*, of being isomorphic (abbreviated $=o$). We study the connections between these relations, order filters, and the aforementioned constructions. A main result of this section is that $<o$ is well-founded.

7.1 Restriction to a set

abbreviation *Restr* :: *'a rel* ⇒ *'a set* ⇒ *'a rel*
where *Restr r A* ≡ *r Int* (*A* × *A*)

lemma *Restr-incr2*:
r <= r' ⟹ *Restr r A <= Restr r' A*
by *blast*

lemma *Restr-incr*:
⟦*r* ≤ *r'*; *A* ≤ *A'*⟧ ⟹ *Restr r A* ≤ *Restr r' A'*
by *blast*

lemma *Restr-Int*:
Restr (*Restr r A*) *B* = *Restr r* (*A Int B*)
by *blast*

lemma *Restr-subset*:
$A \leq B \Longrightarrow Restr\ (Restr\ r\ B)\ A = Restr\ r\ A$
by *blast*

lemma *Restr-iff*: $(a,b) : Restr\ r\ A = (a : A \wedge b : A \wedge (a,b) : r)$
by (*auto simp add*: *Field-def*)

lemma *Restr-subset1*: $Restr\ r\ A \leq r$
by *auto*

lemma *Restr-subset2*: $Restr\ r\ A \leq A \times A$
by *auto*

lemma *Restr-Field*: $Restr\ r\ (Field\ r) = r$
unfolding *Field-def* **by** *auto*

lemma *Refl-Restr*: $Refl\ r \Longrightarrow Refl(Restr\ r\ A)$
unfolding *refl-on-def Field-def* **by** *auto*

lemma *antisym-Restr*:
$antisym\ r \Longrightarrow antisym(Restr\ r\ A)$
unfolding *antisym-def Field-def* **by** *auto*

lemma *Total-Restr*:
$Total\ r \Longrightarrow Total(Restr\ r\ A)$
unfolding *total-on-def Field-def* **by** *auto*

lemma *trans-Restr*:
$trans\ r \Longrightarrow trans(Restr\ r\ A)$
unfolding *trans-def Field-def* **by** *blast*

lemma *Preorder-Restr*:
$Preorder\ r \Longrightarrow Preorder(Restr\ r\ A)$
unfolding *preorder-on-def* **by**(*auto simp add*: *Refl-Restr trans-Restr*)

lemma *Partial-order-Restr*:
$Partial\text{-}order\ r \Longrightarrow Partial\text{-}order(Restr\ r\ A)$
unfolding *partial-order-on-def* **by** (*auto simp add*: *Preorder-Restr antisym-Restr*)

lemma *Linear-order-Restr*:
Linear-order r $\Longrightarrow$ *Linear-order*(*Restr r A*)
unfolding *linear-order-on-def* **by** (*auto simp add*: *Partial-order-Restr Total-Restr*)

lemma *wf-Restr*:
wf r $\Longrightarrow$ *wf*(*Restr r A*)
using *wf-subset Restr-subset* **by** *blast*

lemma *Well-order-Restr*:
assumes *Well-order r*
shows *Well-order*(*Restr r A*)
proof−
  have *Restr r A* − *Id* $\leq$ *r* − *Id* **using** *Restr-subset* **by** *blast*
  hence *wf*(*Restr r A* − *Id*) **using** *assms*
  using *well-order-on-def wf-subset* **by** *blast*
  thus *?thesis* **using** *assms* **unfolding** *well-order-on-def*
  by (*auto simp add*: *Linear-order-Restr*)
qed

lemma *Field-Restr-subset*: *Field*(*Restr r A*) $\leq$ *A*
by(*auto simp add*: *Field-def*)

lemma *Refl-Field-Restr*:
Refl r $\Longrightarrow$ *Field*(*Restr r A*) = (*Field r*) *Int A*
by(*auto simp add*: *refl-on-def Field-def*)

lemma *Refl-Field-Restr2*:
$[\![$*Refl r*; *A* $\leq$ *Field r*$]\!]$ $\Longrightarrow$ *Field*(*Restr r A*) = *A*
by (*auto simp add*: *Refl-Field-Restr*)

lemma *well-order-on-Restr*:
assumes *WELL*: *Well-order r* **and** *SUB*: *A* $\leq$ *Field r*
shows *well-order-on A* (*Restr r A*)
using *assms*
using *Well-order-Restr*[*of r A*] *Refl-Field-Restr2*[*of r A*]
    order-on-defs[*of Field r r*] **by** *auto*

lemma *Restr-incr1*:
A $\leq$ *B* $\Longrightarrow$ *Restr r A* $\leq$ *Restr r B*
by *blast*

7.2 Order filters versus restrictions and embeddings

lemma *Field-Restr-ofilter*:
$[\![$*Well-order r*; *ofilter r A*$]\!] \Longrightarrow$ *Field*(*Restr r A*) = *A*
by (*auto simp add*: *wo-rel-def wo-rel.ofilter-def wo-rel.REFL Refl-Field-Restr2*)

lemma *ofilter-Restr-under*:
assumes *WELL*: *Well-order r* **and** *OF*: *ofilter r A* **and** *IN*: $a \in A$
shows *under* (*Restr r A*) *a* = *under r a*
using *assms wo-rel-def*
proof(*auto simp add*: *wo-rel.ofilter-def rel.under-def*)
  fix *b* **assume** $*$: $a \in A$ **and** $(b,a) \in r$
  hence $b \in$ *under r a* $\wedge$ $a \in$ *Field r*
  unfolding *rel.under-def* **using** *Field-def* **by** *fastforce*
  thus $b \in A$ **using** $*$ *assms* **by** (*auto simp add*: *wo-rel-def wo-rel.ofilter-def*)
qed

lemma *ofilter-Restr*:
assumes *WELL*: *Well-order r* **and**
    OFA: *ofilter r A* **and** *OFB*: *ofilter r B* **and** *SUB*: $A \leq B$
shows *ofilter* (*Restr r B*) *A*
proof−
  let *?rB* = *Restr r B*
  have *Well*: *wo-rel r* **unfolding** *wo-rel-def* **using** *WELL* **.**
  hence *Refl*: *Refl r* **by** (*auto simp add*: *wo-rel.REFL*)
  hence *Field*: *Field ?rB* = *Field r Int B*
  using *Refl-Field-Restr* **by** *blast*
  have *WellB*: *wo-rel ?rB* $\wedge$ *Well-order ?rB* **using** *WELL*
  by (*auto simp add*: *Well-order-Restr wo-rel-def*)

  show *?thesis*
  proof(*auto simp add*: *WellB wo-rel.ofilter-def*)
    fix *a* **assume** $a \in A$
    hence $a \in$ *Field r* $\wedge$ $a \in B$ **using** *assms Well*
    by (*auto simp add*: *wo-rel.ofilter-def*)
    with *Field* **show** $a \in$ *Field*(*Restr r B*) **by** *auto*
  next
    fix *a b* **assume** $*$: $a \in A$ **and** $b \in$ *under* (*Restr r B*) *a*
    hence $b \in$ *under r a*
    using *WELL OFB SUB ofilter-Restr-under*[*of r B a*] **by** *auto*
    thus $b \in A$ **using** $*$ *Well OFA* **by**(*auto simp add*: *wo-rel.ofilter-def*)
  qed
qed

lemma *ofilter-embed*:
assumes *Well-order r*
shows *ofilter r A* = ($A \leq$ *Field r* $\wedge$ *embed* (*Restr r A*) *r id*)

proof
  assume $*$: *ofilter r A*
  show $A \leq$ *Field r* $\wedge$ *embed* (*Restr r A*) *r id*
  proof(*unfold embed-def*, *auto*)
    fix *a* **assume** $a \in A$ **thus** $a \in$ *Field r* **using** *assms* $*$
    by (*auto simp add*: *wo-rel-def wo-rel.ofilter-def*)
  next
    fix *a* **assume** $a \in$ *Field* (*Restr r A*)
    thus *bij-betw id* (*under* (*Restr r A*) *a*) (*under r a*) **using** *assms* $*$
    by (*auto simp add*: *ofilter-Restr-under Field-Restr-ofilter*)
  qed
next
  assume $*$: $A \leq$ *Field r* $\wedge$ *embed* (*Restr r A*) *r id*
  hence *Field*(*Restr r A*) $\leq$ *Field r*
  using *assms* *embed-Field*[*of Restr r A r id*] *id-def*
        Well-order-Restr[*of r*] **by** *auto*
  {**fix** *a* **assume** $a \in A$
   hence $a \in$ *Field*(*Restr r A*) **using** $*$ *assms*
   by (*auto simp add*: *order-on-defs Refl-Field-Restr2*)
   hence *bij-betw id* (*under* (*Restr r A*) *a*) (*under r a*)
   using $*$ **unfolding** *embed-def* **by** *auto*
   hence *under r a* $\leq$ *under* (*Restr r A*) *a*
   unfolding *bij-betw-def* **by** *auto*
   also have $\ldots \leq$ *Field*(*Restr r A*)
   by (*auto simp add*: *rel.under-Field*)
   also have $\ldots \leq A$ **by** (*auto simp add*: *Field-Restr-subset*)
   finally have *under r a* $\leq A$ **.**
  }
  thus *ofilter r A* **using** *assms* $*$
  by(*auto simp add*: *wo-rel-def wo-rel.ofilter-def*)
qed

lemma *ofilter-Restr-Int*:
assumes *WELL*: *Well-order r* **and** *OFA*: *ofilter r A*
shows *ofilter* (*Restr r B*) (*A Int B*)
proof−
  let *?rB* = *Restr r B*
  have *Well*: *wo-rel r* **unfolding** *wo-rel-def* **using** *WELL* **.**
  hence *Refl*: *Refl r* **by** (*auto simp add*: *wo-rel.REFL*)
  hence *Field*: *Field ?rB* = *Field r Int B*
  using *Refl-Field-Restr* **by** *blast*
  have *WellB*: *wo-rel ?rB* $\wedge$ *Well-order ?rB* **using** *WELL*
  by (*auto simp add*: *Well-order-Restr wo-rel-def*)

  show *?thesis* **using** *WellB assms*
  proof(*auto simp add*: *wo-rel.ofilter-def rel.under-def*)
    fix *a* **assume** $a \in A$ **and** $*$: $a \in B$
    hence $a \in$ *Field r* **using** *OFA Well* **by** (*auto simp add*: *wo-rel.ofilter-def*)

with ∗ **show** *a* ∈ *Field ?rB* **using** *Field* **by** *auto*
next
fix *a b* **assume** *a* ∈ *A* **and** (*b*,*a*) ∈ *r*
thus *b* ∈ *A* **using** *Well OFA* **by** (*auto simp add*: *wo-rel.ofilter-def rel.under-def*)
qed
qed

lemma *ofilter-Restr-subset*:
assumes *WELL*: *Well-order r* **and** *OFA*: *ofilter r A* **and** *SUB*: *A* ≤ *B*
shows *ofilter* (*Restr r B*) *A*
proof−
have *A Int B* = *A* **using** *SUB* **by** *blast*
thus *?thesis* **using** *assms ofilter-Restr-Int*[*of r A B*] **by** *auto*
qed

lemma *ofilter-subset-embed*:
assumes *WELL*: *Well-order r* **and**
OFA: *ofilter r A* **and** *OFB*: *ofilter r B*
shows (*A* ≤ *B*) = (*embed* (*Restr r A*) (*Restr r B*) *id*)
proof−
let *?rA* = *Restr r A* **let** *?rB* = *Restr r B*
have *Well*: *wo-rel r* **unfolding** *wo-rel-def* **using** *WELL* **.**
hence *Refl*: *Refl r* **by** (*auto simp add*: *wo-rel.REFL*)
hence *FieldA*: *Field ?rA* = *Field r Int A*
using *Refl-Field-Restr* **by** *blast*
have *FieldB*: *Field ?rB* = *Field r Int B*
using *Refl Refl-Field-Restr* **by** *blast*
have *WellA*: *wo-rel ?rA* ∧ *Well-order ?rA* **using** *WELL*
by (*auto simp add*: *Well-order-Restr wo-rel-def*)
have *WellB*: *wo-rel ?rB* ∧ *Well-order ?rB* **using** *WELL*
by (*auto simp add*: *Well-order-Restr wo-rel-def*)

show *?thesis*
proof
assume ∗: *A* ≤ *B*
hence *ofilter* (*Restr r B*) *A* **using** *assms*
by (*auto simp add*: *ofilter-Restr-subset*)
hence *embed* (*Restr ?rB A*) (*Restr r B*) *id*
using *WellB ofilter-embed*[*of ?rB A*] **by** *auto*
thus *embed* (*Restr r A*) (*Restr r B*) *id*
using ∗ **by** (*auto simp add*: *Restr-subset*)
next
assume ∗: *embed* (*Restr r A*) (*Restr r B*) *id*
{**fix** *a* **assume** ∗∗: *a* ∈ *A*
hence *a* ∈ *Field r* **using** *Well OFA* **by** (*auto simp add*: *wo-rel.ofilter-def*)
with ∗∗ *FieldA* **have** *a* ∈ *Field ?rA* **by** *auto*
hence *a* ∈ *Field ?rB* **using** ∗ *WellA embed-Field*[*of ?rA ?rB id*] **by** *auto*

hence $a \in B$ **using** *FieldB* **by** *auto*
}
thus $A \leq B$ **by** *blast*
qed
qed

lemma *ofilter-subset-embedS-iso*:
assumes *WELL*: *Well-order r* **and**
OFA: *ofilter r A* **and** *OFB*: *ofilter r B*
shows $((A < B) = (\mathit{embedS}\ (\mathit{Restr}\ r\ A)\ (\mathit{Restr}\ r\ B)\ \mathit{id})) \wedge$
$((A = B) = (\mathit{iso}\ (\mathit{Restr}\ r\ A)\ (\mathit{Restr}\ r\ B)\ \mathit{id}))$
proof−
let *?rA* = *Restr r A* **let** *?rB* = *Restr r B*
have *Well*: *wo-rel r* **unfolding** *wo-rel-def* **using** *WELL* **.**
hence *Refl*: *Refl r* **by** (*auto simp add*: *wo-rel.REFL*)
hence *Field ?rA* = *Field r Int A*
using *Refl-Field-Restr* **by** *blast*
hence *FieldA*: *Field ?rA* = *A* **using** *OFA Well*
by (*auto simp add*: *wo-rel.ofilter-def*)
have *Field ?rB* = *Field r Int B*
using *Refl Refl-Field-Restr* **by** *blast*
hence *FieldB*: *Field ?rB* = *B* **using** *OFB Well*
by (*auto simp add*: *wo-rel.ofilter-def*)

show *?thesis* **unfolding** *embedS-def iso-def*
using *assms ofilter-subset-embed*[*of r A B*]
FieldA FieldB bij-betw-id-iff[*of A B*] **by** *auto*
qed

lemma *ofilter-subset-embedS*:
assumes *WELL*: *Well-order r* **and**
OFA: *ofilter r A* **and** *OFB*: *ofilter r B*
shows $(A < B) = \mathit{embedS}\ (\mathit{Restr}\ r\ A)\ (\mathit{Restr}\ r\ B)\ \mathit{id}$
using *assms*
by (*auto simp add*: *ofilter-subset-embedS-iso*)

lemma *ofilter-subset-iso*:
assumes *WELL*: *Well-order r* **and**
OFA: *ofilter r A* **and** *OFB*: *ofilter r B*
shows $(A = B) = \mathit{iso}\ (\mathit{Restr}\ r\ A)\ (\mathit{Restr}\ r\ B)\ \mathit{id}$
using *assms*
by (*auto simp add*: *ofilter-subset-embedS-iso*)

lemma *embed-implies-iso-Restr*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**

EMB: *embed* r' *r f*
shows *iso* r' (*Restr r* (*f* ' (*Field* r'))) *f*
proof−
let $?A' = \textit{Field } r'$
let $?r'' = \textit{Restr r } (f \text{ ` } ?A')$
have *0*: *Well-order* $?r''$ **using** *WELL Well-order-Restr* **by** *blast*
have *1*: *ofilter r* (*f* ' $?A'$) **using** *assms embed-Field-ofilter* **by** *blast*
hence *Field* $?r''$ = *f* ' (*Field* r') **using** *WELL Field-Restr-ofilter* **by** *blast*
hence *bij-betw f* $?A'$ (*Field* $?r''$)
using *EMB embed-inj-on* $WELL'$ **unfolding** *bij-betw-def* **by** *blast*
moreover
{**have** $\forall a\ b.\ (a,b) \in r' \longrightarrow a \in \textit{Field } r' \wedge b \in \textit{Field } r'$
unfolding *Field-def* **by** *auto*
hence *compat* r' $?r''$ *f*
using *assms embed-iff-compat-inj-on-ofilter*
unfolding *compat-def* **by** *blast*
}
ultimately show *?thesis* **using** $WELL'$ *0 iso-iff3* **by** *blast*
qed

7.3 The strict inclusion on proper ofilters is well-founded

definition *ofilterIncl* :: $'a\ rel \Rightarrow 'a\ set\ rel$
where
ofilterIncl r $\equiv \{(A,B).\ \textit{ofilter } r\ A \wedge A \neq \textit{Field } r \wedge$
$\textit{ofilter } r\ B \wedge B \neq \textit{Field } r \wedge A < B\}$

lemma *wf-ofilterIncl*:
assumes *WELL*: *Well-order r*
shows *wf*(*ofilterIncl r*)
proof−
have *Well*: *wo-rel r* **using** *WELL* **by** (*auto simp add*: *wo-rel-def*)
hence *Lo*: *Linear-order r*
by (*auto simp add*: *wo-rel.LIN*)
let $?h = (\lambda\ A.\ \textit{suc } r\ A)$
let $?rS = r - \textit{Id}$
have *wf ?rS* **using** *WELL* **by** (*auto simp add*: *order-on-defs*)
moreover
have *compat* (*ofilterIncl r*) *?rS ?h*
proof(*unfold compat-def ofilterIncl-def*,
intro allI impI, *simp*, *elim conjE*)
fix *A B*
assume $*$: *ofilter r A* $A \neq \textit{Field } r$ **and**
$**$: *ofilter r B* $B \neq \textit{Field } r$ **and** $***$: $A < B$
then obtain *a* **and** *b* **where** *0*: $a \in \textit{Field } r \wedge b \in \textit{Field } r$ **and**
1: $A = \textit{underS } r\ a \wedge B = \textit{underS } r\ b$
using *Well* **by** (*auto simp add*: *wo-rel.ofilter-underS-Field*)
hence $a \neq b$ **using** $***$ **by** *auto*

moreover
have $(a,b) \in r$ **using** *0 1 Lo* $***$
by (*auto simp add*: *rel.underS-incl-iff*)
moreover
have $a = suc\ r\ A \wedge b = suc\ r\ B$
using *Well 0 1* **by** (*auto simp add*: *wo-rel.suc-underS*)
ultimately
show $(suc\ r\ A,\ suc\ r\ B) \in r \wedge suc\ r\ A \neq suc\ r\ B$ **by** *simp*
qed
ultimately show *wf* (*ofilterIncl r*) **by** (*auto simp add*: *compat-wf*)
qed

7.4 Ordering the well-orders by existence of embeddings

We define three relations between well-orders:

- *ordLeq*, of being embedded (abbreviated $\leq o$);
- *ordLess*, of being strictly embedded (abbreviated $<o$);
- *ordIso*, of being isomorphic (abbreviated $=o$).

The prefix "ord" and the index "o" in these names stand for "ordinal-like". These relations shall be proved to be inter-connected in a similar fashion as the trio $\leq$, $<$, $=$ associated to a total order on a set.

definition *ordLeq* :: $('a\ rel * 'a'\ rel)\ set$
where
$ordLeq = \{(r,r').\ Well\text{-}order\ r \wedge Well\text{-}order\ r' \wedge (\exists f.\ embed\ r\ r'\ f)\}$

abbreviation *ordLeq2* :: $'a\ rel \Rightarrow 'a'\ rel \Rightarrow bool$ (**infix** $<=o$ *50*)
where $r <=o\ r' \equiv (r,r') \in ordLeq$

abbreviation *ordLeq3* :: $'a\ rel \Rightarrow 'a'\ rel \Rightarrow bool$ (**infix** $\leq o$ *50*)
where $r \leq o\ r' \equiv r <=o\ r'$

definition *ordLess* :: $('a\ rel * 'a'\ rel)\ set$
where
$ordLess = \{(r,r').\ Well\text{-}order\ r \wedge Well\text{-}order\ r' \wedge (\exists f.\ embedS\ r\ r'\ f)\}$

abbreviation *ordLess2* :: $'a\ rel \Rightarrow 'a'\ rel \Rightarrow bool$ (**infix** $<o$ *50*)
where $r <o\ r' \equiv (r,r') \in ordLess$

definition *ordIso* :: $('a\ rel * 'a'\ rel)\ set$
where
$ordIso = \{(r,r').\ Well\text{-}order\ r \wedge Well\text{-}order\ r' \wedge (\exists f.\ iso\ r\ r'\ f)\}$

abbreviation *ordIso2* :: $'a$ *rel* $\Rightarrow$ $'a'$ *rel* $\Rightarrow$ *bool* (**infix** $=o$ *50*)
where $r =o\ r' \equiv (r,r') \in$ *ordIso*

lemmas *ordRels-def* = *ordLeq-def ordLess-def ordIso-def*

lemma *ordLeq-Well-order-simp*[*simp*]:
assumes $r \leq o\ r'$
shows *Well-order* $r \wedge$ *Well-order* r'
using *assms* **unfolding** *ordLeq-def* **by** *simp*

lemma *ordLess-Well-order-simp*[*simp*]:
assumes $r <o\ r'$
shows *Well-order* $r \wedge$ *Well-order* r'
using *assms* **unfolding** *ordLess-def* **by** *simp*

lemma *ordIso-Well-order-simp*[*simp*]:
assumes $r =o\ r'$
shows *Well-order* $r \wedge$ *Well-order* r'
using *assms* **unfolding** *ordIso-def* **by** *simp*

Notice that the relations $\leq o$, $<o$, $=o$ connect well-orders on potentially *distinct* types. However, some of the lemmas below, including the next one, restrict implicitly the type of these relations to $(('a\ rel) * ('a\ rel))$ *set* , i.e., to $'a$ *rel rel*.

lemma *ordLeq-reflexive*:
Well-order $r \Longrightarrow r \leq o\ r$
unfolding *ordLeq-def* **using** *id-embed*[*of r*] **by** *blast*

corollary *ordLeq-refl-on*: *refl-on* $\{r.$ *Well-order* $r\}$ *ordLeq*
using *ordLeq-reflexive* **unfolding** *ordLeq-def refl-on-def*
by *blast*

lemma *ordLeq-transitive*[*trans*]:
assumes $*$: $r \leq o\ r'$ **and** $**$: $r' \leq o\ r''$
shows $r \leq o\ r''$
proof−
  obtain f **and** f'
  where *1*: *Well-order* $r \wedge$ *Well-order* $r' \wedge$ *Well-order* r'' **and**
    embed $r\ r'\ f$ **and** *embed* $r'\ r''\ f'$
  using $*$ $**$ **unfolding** *ordLeq-def* **by** *blast*
  hence *embed* $r\ r''\ (f'\ o\ f)$
  using *comp-embed*[*of* $r\ r'\ f\ r''\ f'$] **by** *auto*
  thus $r \leq o\ r''$ **unfolding** *ordLeq-def* **using** *1* **by** *auto*

qed

corollary *ordLeq-trans*: *trans ordLeq*
using *trans-def*[*of ordLeq*] *ordLeq-transitive* **by** *blast*

corollary *ordLeq-preorder-on*: *preorder-on* $\{r.$ *Well-order r*$\}$ *ordLeq*
by(*auto simp add*: *preorder-on-def ordLeq-refl-on ordLeq-trans*)

lemma *ordLeq-total*:
$[\![$*Well-order r*; *Well-order r'*$]\!] \Longrightarrow r \leq o\ r' \vee r' \leq o\ r$
unfolding *ordLeq-def* **using** *wellorders-totally-ordered* **by** *blast*

lemma *ordIso-reflexive*:
Well-order r $\Longrightarrow r =o\ r$
unfolding *ordIso-def* **using** *id-iso*[*of r*] **by** *blast*

corollary *ordIso-refl-on*: *refl-on* $\{r.$ *Well-order r*$\}$ *ordIso*
using *ordIso-reflexive* **unfolding** *refl-on-def ordIso-def*
by *blast*

lemma *ordIso-transitive*[*trans*]:
assumes $*$: $r =o\ r'$ **and** $**$: $r' =o\ r''$
shows $r =o\ r''$
proof−
  obtain f **and** f'
  where *1*: *Well-order r* $\wedge$ *Well-order r'* $\wedge$ *Well-order r''* **and**
      iso r r' f **and** *3*: *iso r' r'' f'*
  using $*$ $**$ **unfolding** *ordIso-def* **by** *auto*
  hence *iso r r''* $(f'\ o\ f)$
  using *comp-iso*[*of r r' f r'' f'*] **by** *auto*
  thus $r =o\ r''$ **unfolding** *ordIso-def* **using** *1* **by** *auto*
qed

corollary *ordIso-trans*: *trans ordIso*
using *trans-def*[*of ordIso*] *ordIso-transitive* **by** *blast*

lemma *ordIso-symmetric*:
assumes $*$: $r =o\ r'$
shows $r' =o\ r$
proof−
  obtain f **where** *1*: *Well-order r* $\wedge$ *Well-order r'* **and**

2: *embed* r r' f $\wedge$ *bij-betw* f (*Field* r) (*Field* r')
using $*$ **unfolding** *ordIso-def* **by** (*auto simp add*: *iso-def*)
let *?f* $'$ = *inv-into* (*Field* r) f
have *embed* r' r *?f* $'$ $\wedge$ *bij-betw* *?f* $'$ (*Field* r') (*Field* r)
using *1 2* **by** (*auto simp add*: *bij-betw-inv-into inv-into-Field-embed-bij-betw*)
thus $r' =o\ r$ **unfolding** *ordIso-def* **using** *1* **by** (*auto simp add*: *iso-def*)
qed

corollary *ordIso-sym*: *sym ordIso*
by (*auto simp add*: *sym-def ordIso-symmetric*)

corollary *ordIso-equiv*: *equiv* $\{r.$ *Well-order* $r\}$ *ordIso*
by (*auto simp add*: *equiv-def ordIso-sym ordIso-refl-on ordIso-trans*)

lemma *ordLeq-ordLess-trans*[*trans*]:
assumes $r \leq o\ r'$ **and** $r' <o\ r''$
shows $r <o\ r''$
proof−
have *Well-order* r $\wedge$ *Well-order* r''
using *assms* **unfolding** *ordLeq-def ordLess-def* **by** *auto*
thus *?thesis* **using** *assms* **unfolding** *ordLeq-def ordLess-def*
using *embed-comp-embedS* **by** *blast*
qed

lemma *ordLess-ordLeq-trans*[*trans*]:
assumes $r <o\ r'$ **and** $r' \leq o\ r''$
shows $r <o\ r''$
proof−
have *Well-order* r $\wedge$ *Well-order* r''
using *assms* **unfolding** *ordLeq-def ordLess-def* **by** *auto*
thus *?thesis* **using** *assms* **unfolding** *ordLeq-def ordLess-def*
using *embedS-comp-embed* **by** *blast*
qed

lemma *ordLeq-ordIso-trans*[*trans*]:
assumes $r \leq o\ r'$ **and** $r' =o\ r''$
shows $r \leq o\ r''$
proof−
have *Well-order* r $\wedge$ *Well-order* r''
using *assms* **unfolding** *ordLeq-def ordIso-def* **by** *auto*
thus *?thesis* **using** *assms* **unfolding** *ordLeq-def ordIso-def*
using *embed-comp-iso* **by** *blast*
qed

lemma *ordIso-ordLeq-trans*[*trans*]:
assumes $r =o\ r'$ **and** $r' \leq o\ r''$
shows $r \leq o\ r''$
proof−
  have *Well-order* r ∧ *Well-order* r''
  using *assms* **unfolding** *ordLeq-def ordIso-def* **by** *auto*
  thus *?thesis* **using** *assms* **unfolding** *ordLeq-def ordIso-def*
  using *iso-comp-embed* **by** *blast*
qed

lemma *ordLess-ordIso-trans*[*trans*]:
assumes $r <o\ r'$ **and** $r' =o\ r''$
shows $r <o\ r''$
proof−
  have *Well-order* r ∧ *Well-order* r''
  using *assms* **unfolding** *ordLess-def ordIso-def* **by** *auto*
  thus *?thesis* **using** *assms* **unfolding** *ordLess-def ordIso-def*
  using *embedS-comp-iso* **by** *blast*
qed

lemma *ordIso-ordLess-trans*[*trans*]:
assumes $r =o\ r'$ **and** $r' <o\ r''$
shows $r <o\ r''$
proof−
  have *Well-order* r ∧ *Well-order* r''
  using *assms* **unfolding** *ordLess-def ordIso-def* **by** *auto*
  thus *?thesis* **using** *assms* **unfolding** *ordLess-def ordIso-def*
  using *iso-comp-embedS* **by** *blast*
qed

lemma *ordLess-not-embed*:
assumes $r <o\ r'$
shows $\neg(\exists f'.\ embed\ r'\ r\ f')$
proof−
  obtain f **where** *1*: *Well-order* r ∧ *Well-order* r' **and** *2*: *embed* $r\ r'\ f$ **and**
        3: ¬ *bij-betw* f (*Field* r) (*Field* r')
  using *assms* **unfolding** *ordLess-def* **by**(*auto simp add*: *embedS-def*)
  {**fix** f' **assume** ∗: *embed* $r'\ r\ f'$
   hence *bij-betw* f (*Field* r) (*Field* r') **using** *1 2*
   by (*auto simp add*: *embed-bothWays-Field-bij-betw*)
   with *3* **have** *False* **by** *contradiction*
  }
  thus *?thesis* **by** *blast*
qed

lemma *ordLess-Field*:
assumes *OL*: *r1* $<o$ *r2* **and** *EMB*: *embed r1 r2 f*
shows $\neg$ (*f*‘(*Field r1*) = *Field r2*)
proof−
let *?A1* = *Field r1* **let** *?A2* = *Field r2*
obtain *g* **where**
0: *Well-order r1* $\wedge$ *Well-order r2* **and**
1: *embed r1 r2 g* $\wedge$ $\neg$(*bij-betw g ?A1 ?A2*)
using *OL* **unfolding** *ordLess-def* **by** (*auto simp add*: *embedS-def*)
hence $\forall$ *a* $\in$ *?A1*. *f a* = *g a*
using *0 EMB embed-unique*[*of r1*] **by** *auto*
hence $\neg$(*bij-betw f ?A1 ?A2*)
using *1 bij-betw-cong*[*of ?A1*] **by** *blast*
moreover
have *inj-on f ?A1* **using** *EMB 0*
by (*auto simp add*: *embed-inj-on*)
ultimately show *?thesis*
by (*auto simp add*: *bij-betw-def*)
qed

lemma *ordLess-iff*:
$r <o r' = (\textit{Well-order } r \wedge \textit{Well-order } r' \wedge \neg(\exists f'.\ \textit{embed } r'\ r\ f'))$
proof
assume $*$: $r <o r'$
hence $\neg(\exists f'.\ \textit{embed } r'\ r\ f')$ **using** *ordLess-not-embed*[*of r r'*] **by** *simp*
with $*$ **show** $\textit{Well-order } r \wedge \textit{Well-order } r' \wedge \neg\ (\exists f'.\ \textit{embed } r'\ r\ f')$
unfolding *ordLess-def* **by** *auto*
next
assume $*$: $\textit{Well-order } r \wedge \textit{Well-order } r' \wedge \neg\ (\exists f'.\ \textit{embed } r'\ r\ f')$
then obtain *f* **where** *1*: *embed r r' f*
using *wellorders-totally-ordered*[*of r r'*] **by** *blast*
moreover
{**assume** *bij-betw f* (*Field r*) (*Field r'*)
with $*$ *1* **have** *embed r' r* (*inv-into* (*Field r*) *f*)
using *inv-into-Field-embed-bij-betw*[*of r r' f*] **by** *auto*
with $*$ **have** *False* **by** *blast*
}
ultimately show (*r*,*r'*) $\in$ *ordLess*
unfolding *ordLess-def* **using** $*$ **by** (*fastforce simp add*: *embedS-def*)
qed

lemma *ordLess-irreflexive*: $\neg$ $r <o r$
proof
assume $r <o r$
hence *Well-order r* $\wedge$ $\neg(\exists f.\ \textit{embed } r\ r\ f)$
unfolding *ordLess-iff* **..**

moreover have *embed r r id* **using** *id-embed*[*of r*] **.**
ultimately show *False* **by** *blast*
qed

lemma *ordLess-irrefl*: *irrefl ordLess*
by(*unfold irrefl-def*, *auto simp add*: *ordLess-irreflexive*)

lemma *ordLess-or-ordIso*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
shows $r <o\ r' \vee r' <o\ r \vee r =o\ r'$
unfolding *ordLess-def ordIso-def*
using *assms embedS-or-iso*[*of r r'*] **by** *auto*

lemma *ordLeq-iff-ordLess-or-ordIso*:
$r \leq o\ r' = (r <o\ r' \vee r =o\ r')$
unfolding *ordRels-def embedS-defs iso-defs* **by** *blast*

corollary *ordLeq-ordLess-Un-ordIso*:
ordLeq = *ordLess* ∪ *ordIso*
by (*auto simp add*: *ordLeq-iff-ordLess-or-ordIso*)

lemma *ordIso-iff-ordLeq*:
$(r =o\ r') = (r \leq o\ r' \wedge r' \leq o\ r)$
proof
assume $r =o\ r'$
then obtain *f* **where** *1*: *Well-order r* ∧ *Well-order r'* ∧
embed r r' f ∧ *bij-betw f* (*Field r*) (*Field r'*)
unfolding *ordIso-def iso-defs* **by** *auto*
hence *embed r r' f* ∧ *embed r' r* (*inv-into* (*Field r*) *f*)
by (*auto simp add*: *inv-into-Field-embed-bij-betw*)
thus $r \leq o\ r' \wedge r' \leq o\ r$
unfolding *ordLeq-def* **using** *1* **by** *auto*
next
assume $r \leq o\ r' \wedge r' \leq o\ r$
then obtain *f* **and** *g* **where** *1*: *Well-order r* ∧ *Well-order r'* ∧
embed r r' f ∧ *embed r' r g*
unfolding *ordLeq-def* **by** *auto*
hence *iso r r' f* **by** (*auto simp add*: *embed-bothWays-iso*)
thus $r =o\ r'$ **unfolding** *ordIso-def* **using** *1* **by** *auto*
qed

lemma *not-ordLess-ordLeq*:
$r <o\ r' \Longrightarrow \neg\ r' \leq o\ r$

using *ordLess-ordLeq-trans ordLess-irreflexive* **by** *blast*

lemma *not-ordLeq-ordLess*:
$r \leq o\ r' \Longrightarrow \neg\ r' <o\ r$
using *not-ordLess-ordLeq* **by** *blast*

lemma *ordLess-or-ordLeq*:
assumes *WELL*: *Well-order* r **and** *WELL'*: *Well-order* r'
shows $r <o\ r' \vee r' \leq o\ r$
proof−
  have $r \leq o\ r' \vee r' \leq o\ r$
  using *assms* **by** (*auto simp add*: *ordLeq-total*)
  moreover
  {**assume** $\neg\ r <o\ r' \wedge r \leq o\ r'$
   hence $r =o\ r'$ **using** *ordLeq-iff-ordLess-or-ordIso* **by** *blast*
   hence $r' \leq o\ r$ **using** *ordIso-symmetric ordIso-iff-ordLeq* **by** *blast*
  }
  ultimately show *?thesis* **by** *blast*
qed

lemma *ordIso-or-ordLess*:
assumes *WELL*: *Well-order* r **and** *WELL'*: *Well-order* r'
shows $r =o\ r' \vee r <o\ r' \vee r' <o\ r$
using *assms ordLess-or-ordLeq ordLeq-iff-ordLess-or-ordIso* **by** *blast*

lemma *not-ordLess-ordIso*:
$r <o\ r' \Longrightarrow \neg\ r =o\ r'$
using *assms ordLess-ordIso-trans ordIso-symmetric ordLess-irreflexive* **by** *blast*

lemma *not-ordLeq-iff-ordLess*[*simp*]:
assumes *WELL*: *Well-order* r **and** *WELL'*: *Well-order* r'
shows $(\neg\ r' \leq o\ r) = (r <o\ r')$
using *assms not-ordLess-ordLeq ordLess-or-ordLeq* **by** *blast*

lemma *not-ordLess-iff-ordLeq*[*simp*]:
assumes *WELL*: *Well-order* r **and** *WELL'*: *Well-order* r'
shows $(\neg\ r' <o\ r) = (r \leq o\ r')$
using *assms not-ordLess-ordLeq ordLess-or-ordLeq* **by** *blast*

lemma *ordLess-transitive*[*trans*]:
$[\![r <o\ r';\ r' <o\ r'']\!] \Longrightarrow r <o\ r''$
using *assms ordLess-ordLeq-trans ordLeq-iff-ordLess-or-ordIso* **by** *blast*

corollary *ordLess-trans*: *trans ordLess*
unfolding *trans-def* **using** *ordLess-transitive* **by** *blast*

lemmas *ordIso-equivalence* = *ordIso-transitive ordIso-reflexive ordIso-symmetric*

lemmas *ord-trans* = *ordIso-transitive ordLeq-transitive ordLess-transitive*
ordIso-ordLeq-trans ordLeq-ordIso-trans
ordIso-ordLess-trans ordLess-ordIso-trans
ordLess-ordLeq-trans ordLeq-ordLess-trans

lemma *ordIso-imp-ordLeq*:
$r =o\ r' \Longrightarrow r \leq o\ r'$
using *ordIso-iff-ordLeq* **by** *blast*

lemma *ordLess-imp-ordLeq*:
$r <o\ r' \Longrightarrow r \leq o\ r'$
using *ordLeq-iff-ordLess-or-ordIso* **by** *blast*

lemma *ofilter-subset-ordLeq*:
assumes *WELL*: *Well-order r* **and**
OFA: *ofilter r A* **and** *OFB*: *ofilter r B*
shows $(A \leq B) = (\mathit{Restr}\ r\ A \leq o\ \mathit{Restr}\ r\ B)$
proof
assume $A \leq B$
thus *Restr r A* $\leq o$ *Restr r B*
unfolding *ordLeq-def* **using** *assms*
Well-order-Restr Well-order-Restr ofilter-subset-embed **by** *blast*
next
assume $*$: *Restr r A* $\leq o$ *Restr r B*
then obtain *f* **where** *embed* (*Restr r A*) (*Restr r B*) *f*
unfolding *ordLeq-def* **by** *blast*
{**assume** $B < A$
hence *Restr r B* $<o$ *Restr r A*
unfolding *ordLess-def* **using** *assms*
Well-order-Restr Well-order-Restr ofilter-subset-embedS **by** *blast*
hence *False* **using** $*$ *not-ordLess-ordLeq* **by** *blast*
}
thus $A \leq B$ **using** *OFA OFB WELL*
wo-rel-def[*of r*] *wo-rel.ofilter-linord*[*of r A B*] **by** *blast*
qed

lemma *ofilter-subset-ordLess*:

assumes *WELL*: *Well-order r* **and**
    OFA: *ofilter r A* **and** *OFB*: *ofilter r B*
shows $(A < B) = (Restr\ r\ A <o\ Restr\ r\ B)$
proof−
  let *?rA* = *Restr r A* **let** *?rB* = *Restr r B*
  have *1*: *Well-order ?rA* $\wedge$ *Well-order ?rB*
  using *WELL Well-order-Restr* **by** *blast*
  have $(A < B) = (\neg\ B \leq A)$ **using** *assms*
  wo-rel-def wo-rel.ofilter-linord[*of r A B*] **by** *blast*
  also have ... = ($\neg$ *Restr r B* $\leq$*o Restr r A*)
  using *assms ofilter-subset-ordLeq* **by** *blast*
  also have ... = (*Restr r A* <*o Restr r B*)
  using *1 not-ordLeq-iff-ordLess* **by** *blast*
  finally show *?thesis* **.**
qed

lemma *ofilter-ordLeq*:
assumes *Well-order r* **and** *ofilter r A*
shows *Restr r A* $\leq$*o r*
proof−
  have $A \leq$ *Field r* **using** *assms* **by** (*auto simp add*: *wo-rel-def wo-rel.ofilter-def*)
  thus *?thesis* **using** *assms*
  by (*auto simp add*: *ofilter-subset-ordLeq wo-rel.Field-ofilter*
    wo-rel-def Restr-Field)
qed

corollary *under-Restr-ordLeq*:
Well-order r $\Longrightarrow$ *Restr r* (*under r a*) $\leq$*o r*
by (*auto simp add*: *ofilter-ordLeq wo-rel.under-ofilter wo-rel-def*)

lemma *ofilter-ordLess*:
$[\![$*Well-order r*; *ofilter r A*$]\!]$ $\Longrightarrow$ (*A* < *Field r*) = (*Restr r A* <*o r*)
by (*auto simp add*: *ofilter-subset-ordLess wo-rel.Field-ofilter*
  wo-rel-def Restr-Field)

corollary *underS-Restr-ordLess*:
assumes *Well-order r* **and** *Field r* $\neq \{\}$
shows *Restr r* (*underS r a*) <*o r*
proof−
  have *underS r a* < *Field r* **using** *assms*
  by (*auto simp add*: *rel.underS-Field3*)
  thus *?thesis* **using** *assms*
  by (*auto simp add*: *ofilter-ordLess wo-rel.underS-ofilter wo-rel-def*)
qed

lemma *embed-ordLess-ofilterIncl*:
assumes
OL12: *r1* <*o r2* **and** *OL23*: *r2* <*o r3* **and**
EMB13: *embed r1 r3 f13* **and** *EMB23*: *embed r2 r3 f23*
shows (*f13*‘(*Field r1*), *f23*‘(*Field r2*)) ∈ (*ofilterIncl r3*)
proof−
have *OL13*: *r1* <*o r3*
using *OL12 OL23* **using** *ordLess-transitive* **by** *auto*
let *?A1* = *Field r1* **let** *?A2* =*Field r2* **let** *?A3* =*Field r3*
obtain *f12 g23* **where**
0: *Well-order r1* ∧ *Well-order r2* ∧ *Well-order r3* **and**
1: *embed r1 r2 f12* ∧ ¬(*bij-betw f12 ?A1 ?A2*) **and**
2: *embed r2 r3 g23* ∧ ¬(*bij-betw g23 ?A2 ?A3*)
using *OL12 OL23* **unfolding** *ordLess-def* **by** (*auto simp add*: *embedS-def*)
hence ∀ *a* ∈ *?A2*. *f23 a* = *g23 a*
using *EMB23 embed-unique*[*of r2 r3*] **by** *blast*
hence *3*: ¬(*bij-betw f23 ?A2 ?A3*)
using *2 bij-betw-cong*[*of ?A2 f23 g23*] **by** *blast*

have *4*: *ofilter r2* (*f12* ‘ *?A1*) ∧ *f12* ‘ *?A1* ≠ *?A2*
using *0 1 OL12* **by** (*auto simp add*: *embed-Field-ofilter ordLess-Field*)
have *5*: *ofilter r3* (*f23* ‘ *?A2*) ∧ *f23* ‘ *?A2* ≠ *?A3*
using *0 EMB23 OL23* **by** (*auto simp add*: *embed-Field-ofilter ordLess-Field*)
have *6*: *ofilter r3* (*f13* ‘ *?A1*) ∧ *f13* ‘ *?A1* ≠ *?A3*
using *0 EMB13 OL13* **by** (*auto simp add*: *embed-Field-ofilter ordLess-Field*)

have *f12* ‘ *?A1* < *?A2*
using *0 4* **by** (*auto simp add*: *wo-rel-def wo-rel.ofilter-def*)
moreover have *inj-on f23 ?A2*
using *EMB23 0* **by** (*auto simp add*: *wo-rel-def embed-inj-on*)
ultimately
have *f23* ‘ (*f12* ‘ *?A1*) < *f23* ‘ *?A2*
by (*auto simp add*: *inj-on-strict-subset*)
moreover
{have *embed r1 r3* (*f23 o f12*)
using *1 EMB23 0* **by** (*auto simp add*: *comp-embed*)
hence ∀ *a* ∈ *?A1*. *f23*(*f12 a*) = *f13 a*
using *EMB13 0 embed-unique*[*of r1 r3 f23 o f12 f13*] **by** *auto*
hence *f23* ‘ (*f12* ‘ *?A1*) = *f13* ‘ *?A1* **by** *force*
}
ultimately
have *f13* ‘ *?A1* < *f23* ‘ *?A2* **by** *simp*

with *5 6* **show** *?thesis*
unfolding *ofilterIncl-def* **by** *auto*
qed

lemma *ordLess-iff-ordIso-Restr*:
assumes *WELL*: *Well-order r* **and** *WELL′*: *Well-order r′*
shows $(r' <o\ r) = (\exists\, a \in \mathit{Field}\ r.\ r' =o\ \mathit{Restr}\ r\ (\mathit{underS}\ r\ a))$
proof(*auto*)
 fix *a* **assume** $*$: $a \in \mathit{Field}\ r$ **and** $**$: $r' =o\ \mathit{Restr}\ r\ (\mathit{underS}\ r\ a)$
 hence $\mathit{Restr}\ r\ (\mathit{underS}\ r\ a) <o\ r$ **using** *WELL underS-Restr-ordLess*[*of r*] **by** *blast*
 thus $r' <o\ r$ **using** $**$ *ordIso-ordLess-trans* **by** *blast*
next
 assume $r' <o\ r$
 then obtain *f* **where** *1*: $\mathit{Well\text{-}order}\ r \wedge \mathit{Well\text{-}order}\ r'$ **and**
  2: $\mathit{embed}\ r'\ r\ f \wedge f\ `\ (\mathit{Field}\ r') \neq \mathit{Field}\ r$
 unfolding *ordLess-def embedS-def-raw bij-betw-def* **using** *embed-inj-on* **by** *blast*
 hence $\mathit{ofilter}\ r\ (f\ `\ (\mathit{Field}\ r'))$ **using** *embed-Field-ofilter* **by** *blast*
 then obtain *a* **where** *3*: $a \in \mathit{Field}\ r$ **and** *4*: $\mathit{underS}\ r\ a = f\ `\ (\mathit{Field}\ r')$
 using *1 2* **by** (*auto simp add*: *wo-rel.ofilter-underS-Field wo-rel-def*)
 have $\mathit{iso}\ r'\ (\mathit{Restr}\ r\ (f\ `\ (\mathit{Field}\ r')))\ f$
 using *embed-implies-iso-Restr 2 assms* **by** *blast*
 moreover have $\mathit{Well\text{-}order}\ (\mathit{Restr}\ r\ (f\ `\ (\mathit{Field}\ r')))$
 using *WELL Well-order-Restr* **by** *blast*
 ultimately have $r' =o\ \mathit{Restr}\ r\ (f\ `\ (\mathit{Field}\ r'))$
 using *WELL′* **unfolding** *ordIso-def* **by** *auto*
 hence $r' =o\ \mathit{Restr}\ r\ (\mathit{underS}\ r\ a)$ **using** *4* **by** *auto*
 thus $\exists\, a \in \mathit{Field}\ r.\ r' =o\ \mathit{Restr}\ r\ (\mathit{underS}\ r\ a)$ **using** *3* **by** *auto*
qed

lemma *internalize-ordLess*:
$(r' <o\ r) = (\exists\, p.\ \mathit{Field}\ p < \mathit{Field}\ r \wedge r' =o\ p \wedge p <o\ r)$
proof
 assume $*$: $r' <o\ r$
 hence *0*: $\mathit{Well\text{-}order}\ r \wedge \mathit{Well\text{-}order}\ r'$ **unfolding** *ordLess-def* **by** *auto*
 with $*$ **obtain** *a* **where** *1*: $a \in \mathit{Field}\ r$ **and** *2*: $r' =o\ \mathit{Restr}\ r\ (\mathit{underS}\ r\ a)$
 using *ordLess-iff-ordIso-Restr* **by** *blast*
 let $?p = \mathit{Restr}\ r\ (\mathit{underS}\ r\ a)$
 have $\mathit{ofilter}\ r\ (\mathit{underS}\ r\ a)$ **using** *0*
 by (*auto simp add*: *wo-rel-def wo-rel.underS-ofilter*)
 hence $\mathit{Field}\ ?p = \mathit{underS}\ r\ a$ **using** *0 Field-Restr-ofilter* **by** *blast*
 hence $\mathit{Field}\ ?p < \mathit{Field}\ r$ **using** *rel.underS-Field2 1* **by** *fastforce*
 moreover have $?p <o\ r$ **using** *underS-Restr-ordLess*[*of r a*] *0 1* **by** *blast*
 ultimately
 show $\exists\, p.\ \mathit{Field}\ p < \mathit{Field}\ r \wedge r' =o\ p \wedge p <o\ r$ **using** *2* **by** *blast*
next
 assume $\exists\, p.\ \mathit{Field}\ p < \mathit{Field}\ r \wedge r' =o\ p \wedge p <o\ r$
 thus $r' <o\ r$ **using** *ordIso-ordLess-trans* **by** *blast*
qed

lemma *internalize-ordLeq*:

$(r' \leq o\ r) = (\exists\, p.\ Field\ p \leq Field\ r \wedge r' =o\ p \wedge p \leq o\ r)$
proof
  assume $*$: $r' \leq o\ r$
  moreover
  {**assume** $r' <o\ r$
   then obtain p **where** $Field\ p < Field\ r \wedge r' =o\ p \wedge p <o\ r$
   using *internalize-ordLess*[*of* r' r] **by** *blast*
   hence $\exists\, p.\ Field\ p \leq Field\ r \wedge r' =o\ p \wedge p \leq o\ r$
   using *ordLeq-iff-ordLess-or-ordIso* **by** *blast*
  }
  moreover
  have $r \leq o\ r$ **using** $*$ *ordLeq-def ordLeq-reflexive* **by** *blast*
  ultimately show $\exists\, p.\ Field\ p \leq Field\ r \wedge r' =o\ p \wedge p \leq o\ r$
  using *ordLeq-iff-ordLess-or-ordIso* **by** *blast*
next
  assume $\exists\, p.\ Field\ p \leq Field\ r \wedge r' =o\ p \wedge p \leq o\ r$
  thus $r' \leq o\ r$ **using** *ordIso-ordLeq-trans* **by** *blast*
qed

lemma *ordLeq-iff-ordLess-Restr*:
assumes *WELL*: *Well-order* r **and** *WELL'*: *Well-order* r'
shows $(r \leq o\ r') = (\forall\, a \in Field\ r.\ Restr\ r\ (underS\ r\ a) <o\ r')$
proof(*auto*)
  assume $*$: $r \leq o\ r'$
  fix a **assume** $a \in Field\ r$
  hence $Restr\ r\ (underS\ r\ a) <o\ r$
  using *WELL underS-Restr-ordLess*[*of* r] **by** *blast*
  thus $Restr\ r\ (underS\ r\ a) <o\ r'$
  using $*$ *ordLess-ordLeq-trans* **by** *blast*
next
  assume $*$: $\forall\, a \in Field\ r.\ Restr\ r\ (underS\ r\ a) <o\ r'$
  {**assume** $r' <o\ r$
   then obtain a **where** $a \in Field\ r \wedge r' =o\ Restr\ r\ (underS\ r\ a)$
   using *assms ordLess-iff-ordIso-Restr* **by** *blast*
   hence *False* **using** $*$ *not-ordLess-ordIso ordIso-symmetric* **by** *blast*
  }
  thus $r \leq o\ r'$ **using** *ordLess-or-ordLeq assms* **by** *blast*
qed

lemma *finite-ordLess-infinite*:
assumes *WELL*: *Well-order* r **and** *WELL'*: *Well-order* r' **and**
    FIN: *finite*(*Field* r) **and** *INF*: *infinite*(*Field* r')
shows $r <o\ r'$
proof−
  {**assume** $r' \leq o\ r$
   then obtain h **where** $inj\text{-}on\ h\ (Field\ r') \wedge h\ `\ (Field\ r') \leq Field\ r$
   unfolding *ordLeq-def* **using** *assms embed-inj-on embed-Field* **by** *blast*

hence *False* **using** *finite-imageD finite-subset FIN INF* **by** *blast*
}
thus *?thesis* **using** *WELL WELL′ ordLess-or-ordLeq* **by** *blast*
qed

lemma *finite-well-order-on-ordIso*:
assumes *FIN*: *finite A* **and**
WELL: *well-order-on A r* **and** *WELL′*: *well-order-on A r′*
shows $r =o\ r'$
proof−
have *0*: *Well-order r* ∧ *Well-order r′* ∧ *Field r* = *A* ∧ *Field r′* = *A*
using *assms rel.well-order-on-Well-order* **by** *blast*
moreover
have ∀ *r r′*. *well-order-on A r* ∧ *well-order-on A r′* ∧ $r \leq o\ r'$
⟶ $r =o\ r'$
proof(*clarify*)
fix *r r′* **assume** ∗: *well-order-on A r* **and** ∗∗: *well-order-on A r′*
have *2*: *Well-order r* ∧ *Well-order r′* ∧ *Field r* = *A* ∧ *Field r′* = *A*
using ∗ ∗∗ *rel.well-order-on-Well-order* **by** *blast*
assume $r \leq o\ r'$
then obtain *f* **where** *1*: *embed r r′ f* **and**
inj-on f A ∧ *f ‘ A* ≤ *A*
unfolding *ordLeq-def* **using** *2 embed-inj-on embed-Field* **by** *blast*
hence *bij-betw f A A* **unfolding** *bij-betw-def* **using** *FIN endo-inj-surj* **by** *blast*
thus $r =o\ r'$ **unfolding** *ordIso-def iso-def-raw* **using** *1 2* **by** *auto*
qed
ultimately show *?thesis* **using** *assms ordLeq-total ordIso-symmetric* **by** *blast*
qed

7.5 $<o$ is well-founded

Of course, it only makes sense to state that the $<o$ is well-founded on the restricted type *′a rel rel*. We prove this by first showing that, for any set of well-orders all embedded in a fixed well-order, the function mapping each well-order in the set to an order filter of the fixed well-order is compatible w.r.t. to $<o$ versus *strict inclusion*; and we already know that strict inclusion of order filters is well-founded.

definition *ord-to-filter* :: *′a rel* ⇒ *′a rel* ⇒ *′a set*
where *ord-to-filter r0 r* ≡ (*SOME f*. *embed r r0 f*) ‘ (*Field r*)

lemma *ord-to-filter-compat*:
compat (*ordLess Int* (*ordLess*ˆ−*1*‘‘{*r0*} × *ordLess*ˆ−*1*‘‘{*r0*}))
(*ofilterIncl r0*)
(*ord-to-filter r0*)
proof(*unfold compat-def ord-to-filter-def*, *clarify*)
fix *r1*::*′a rel* **and** *r2*::*′a rel*

let *?A1* = *Field r1* **let** *?A2* =*Field r2* **let** *?A0* =*Field r0*
let *?phi10* = λ *f10*. *embed r1 r0 f10* **let** *?f10* = *SOME f*. *?phi10 f*
let *?phi20* = λ *f20*. *embed r2 r0 f20* **let** *?f20* = *SOME f*. *?phi20 f*
assume $*$: *r1* <*o r0 r2* <*o r0* **and** $**$: *r1* <*o r2*
hence ($\exists$ *f*. *?phi10 f*) $\wedge$ ($\exists$ *f*. *?phi20 f*)
unfolding *ordLess-def* **by** (*auto simp add*: *embedS-def*)
hence *?phi10 ?f10* $\wedge$ *?phi20 ?f20* **by** (*auto simp add*: *someI-ex*)
thus (*?f10* ‘ *?A1*, *?f20* ‘ *?A2*) $\in$ *ofilterIncl r0*
using $*$ $**$ **by** (*auto simp add*: *embed-ordLess-ofilterIncl*)
qed

theorem *wf-ordLess*: *wf ordLess*
proof−
{**fix** *r0*

let *?ordLess* = *ordLess*::('*d rel* $*$ '*d rel*) *set*
let *?R* = *?ordLess Int* (*?ordLess*ˆ−*1*‘‘{*r0*} $\times$ *?ordLess*ˆ−*1*‘‘{*r0*})
{**assume** *Case1*: *Well-order r0*
hence *wf ?R*
using *wf-ofilterIncl*[*of r0*]
compat-wf[*of ?R ofilterIncl r0 ord-to-filter r0*]
ord-to-filter-compat[*of r0*] **by** *auto*
}
moreover
{**assume** *Case2*: $\neg$ *Well-order r0*
hence *?R* = {} **unfolding** *ordLess-def* **by** *auto*
hence *wf ?R* **using** *wf-empty* **by** *simp*
}
ultimately have *wf ?R* **by** *blast*
}
thus *?thesis* **by** (*auto simp add*: *trans-wf-iff ordLess-trans*)
qed

corollary *exists-minim-Well-order*:
assumes *NE*: $R \neq \{\}$ **and** *WELL*: $\forall\, r \in R$. *Well-order r*
shows $\exists\, r \in R.\ \forall\, r' \in R.\ r \leq o\ r'$
proof−
obtain *r* **where** $r \in R \wedge (\forall\, r' \in R.\ \neg\ r' <o\ r)$
using *assms wf-ordLess* **unfolding** *wf-eq-minimal*[*of ordLess*] **by** *force*

with *not-ordLeq-iff-ordLess assms* **show** *?thesis* **by** *blast*
qed

7.6 Copy via direct images

The direct image operator is the dual of the inverse image operator *inv-image* from *Relation.thy*. It is useful for transporting a well-order between different types.

definition *dir-image* :: $'a$ *rel* $\Rightarrow$ $('a \Rightarrow 'a')$ $\Rightarrow$ $'a'$ *rel*
where
dir-image r f = $\{(f\ a,\ f\ b)|\ a\ b.\ (a,b) \in r\}$

lemma *dir-image-Field*:
Field(*dir-image r f*) $\leq$ *f* ‘ (*Field r*)
unfolding *dir-image-def Field-def* **by** *auto*

lemma *Id-dir-image*: *dir-image Id f* $\leq$ *Id*
unfolding *dir-image-def* **by** *auto*

lemma *Un-dir-image*:
dir-image (*r1* $\cup$ *r2*) *f* = (*dir-image r1 f*) $\cup$ (*dir-image r2 f*)
unfolding *dir-image-def* **by** *auto*

lemma *Int-dir-image*:
assumes *inj-on f* (*Field r1* $\cup$ *Field r2*)
shows *dir-image* (*r1 Int r2*) *f* = (*dir-image r1 f*) *Int* (*dir-image r2 f*)
proof
  show *dir-image* (*r1 Int r2*) *f* $\leq$ (*dir-image r1 f*) *Int* (*dir-image r2 f*)
  using *assms* **unfolding** *dir-image-def inj-on-def* **by** *auto*
next
  show (*dir-image r1 f*) *Int* (*dir-image r2 f*) $\leq$ *dir-image* (*r1 Int r2*) *f*
  proof(*clarify*)
    fix a' b'
    assume $(a',b') \in$ *dir-image r1 f* $(a',b') \in$ *dir-image r2 f*
    then obtain *a1 b1 a2 b2*
    where *1*: $a' = f\ a1 \wedge b' = f\ b1 \wedge a' = f\ a2 \wedge b' = f\ b2$ **and**
        2: $(a1,b1) \in r1 \wedge (a2,b2) \in r2$ **and**
        3: $\{a1,b1\} \leq$ *Field r1* $\wedge$ $\{a2,b2\} \leq$ *Field r2*
    unfolding *dir-image-def Field-def* **by** *blast*
    hence $a1 = a2 \wedge b1 = b2$ **using** *assms* **unfolding** *inj-on-def* **by** *auto*
    hence $a' = f\ a1 \wedge b' = f\ b1 \wedge (a1,b1) \in r1$ *Int* $r2 \wedge (a2,b2) \in r1$ *Int* $r2$
    using *1 2* **by** *auto*
    thus $(a',b') \in$ *dir-image* $(r1 \cap r2)$ *f*
    unfolding *dir-image-def* **by** *blast*
  qed
qed

lemma *dir-image-minus-Id*:
inj-on f (*Field r*) $\Longrightarrow$ (*dir-image r f*) $-$ *Id* = *dir-image* ($r - Id$) *f*
unfolding *inj-on-def Field-def dir-image-def* **by** *auto*

lemma *Refl-dir-image*:
assumes *Refl r*
shows *Refl*(*dir-image r f*)
proof−
{**fix** a' b'
assume (a',b') ∈ *dir-image r f*
then obtain *a b* **where** *1*: $a' = f\ a \wedge b' = f\ b \wedge (a,b) \in r$
unfolding *dir-image-def* **by** *blast*
hence $a \in$ *Field r* $\wedge$ $b \in$ *Field r* **using** *Field-def* **by** *fastforce*
hence $(a,a) \in r \wedge (b,b) \in r$ **using** *assms* **by** (*auto simp add*: *refl-on-def*)
with *1* **have** (a',a') ∈ *dir-image r f* ∧ (b',b') ∈ *dir-image r f*
unfolding *dir-image-def* **by** *auto*
}
thus *?thesis*
by(*unfold refl-on-def Field-def Domain-def Range-def*, *auto*)
qed

lemma *trans-dir-image*:
assumes *TRANS*: *trans r* **and** *INJ*: *inj-on f* (*Field r*)
shows *trans*(*dir-image r f*)
proof(*unfold trans-def*, *auto*)
fix a' b' c'
assume (a',b') ∈ *dir-image r f* (b',c') ∈ *dir-image r f*
then obtain *a b1 b2 c* **where** *1*: $a' = f\ a \wedge b' = f\ b1 \wedge b' = f\ b2 \wedge c' = f\ c$
and
2: $(a,b1) \in r \wedge (b2,c) \in r$
unfolding *dir-image-def* **by** *blast*
hence $b1 \in$ *Field r* ∧ $b2 \in$ *Field r*
unfolding *Field-def* **by** *auto*
hence $b1 = b2$ **using** *1 INJ* **unfolding** *inj-on-def* **by** *auto*
hence (a,c): *r* **using** *2 TRANS* **unfolding** *trans-def* **by** *blast*
thus (a',c') ∈ *dir-image r f*
unfolding *dir-image-def* **using** *1* **by** *auto*
qed

lemma *Preorder-dir-image*:
⟦*Preorder r*; *inj-on f* (*Field r*)⟧ ⟹ *Preorder* (*dir-image r f*)
by(*unfold preorder-on-def*, *auto simp add*: *Refl-dir-image trans-dir-image*)

lemma *antisym-dir-image*:
assumes *AN*: *antisym r* **and** *INJ*: *inj-on f* (*Field r*)
shows *antisym*(*dir-image r f*)
proof(*unfold antisym-def*, *auto*)
fix a' b'
assume (a',b') ∈ *dir-image r f* (b',a') ∈ *dir-image r f*

then obtain *a1 b1 a2 b2* **where** *1*: $a' = f\ a1 \wedge a' = f\ a2 \wedge b' = f\ b1 \wedge b' = f\ b2$ **and**
2: $(a1,b1) \in r \wedge (b2,a2) \in r$ **and**
3: $\{a1,a2,b1,b2\} \leq Field\ r$
unfolding *dir-image-def Field-def* **by** *blast*
hence $a1 = a2 \wedge b1 = b2$ **using** *INJ* **unfolding** *inj-on-def* **by** *auto*
hence $a1 = b2$ **using** *2 AN* **unfolding** *antisym-def* **by** *auto*
thus $a' = b'$ **using** *1* **by** *auto*
qed

lemma *Partial-order-dir-image*:
$[\![Partial\text{-}order\ r;\ inj\text{-}on\ f\ (Field\ r)]\!] \Longrightarrow Partial\text{-}order\ (dir\text{-}image\ r\ f)$
by(*unfold partial-order-on-def*, *auto simp add*: *Preorder-dir-image antisym-dir-image*)

lemma *Total-dir-image*:
assumes *TOT*: *Total r* **and** *INJ*: *inj-on f* (*Field r*)
shows *Total*(*dir-image r f*)
proof(*unfold total-on-def*, *intro ballI impI*)
fix $a'\ b'$
assume $a' \in Field\ (dir\text{-}image\ r\ f)\ b' \in Field\ (dir\text{-}image\ r\ f)$
then obtain *a* **and** *b* **where** *1*: $a \in Field\ r \wedge b \in Field\ r \wedge f\ a = a' \wedge f\ b = b'$
using *dir-image-Field*[*of r f*] **by** *blast*
moreover assume $a' \neq b'$
ultimately have $a \neq b$ **using** *INJ* **unfolding** *inj-on-def* **by** *auto*
hence $(a,b) \in r \vee (b,a) \in r$ **using** *1 TOT* **unfolding** *total-on-def* **by** *auto*
thus $(a',b') \in dir\text{-}image\ r\ f \vee (b',a') \in dir\text{-}image\ r\ f$
using *1* **unfolding** *dir-image-def* **by** *auto*
qed

lemma *Linear-order-dir-image*:
$[\![Linear\text{-}order\ r;\ inj\text{-}on\ f\ (Field\ r)]\!] \Longrightarrow Linear\text{-}order\ (dir\text{-}image\ r\ f)$
by(*unfold linear-order-on-def*, *auto simp add*: *Partial-order-dir-image Total-dir-image*)

lemma *wf-dir-image*:
assumes *WF*: *wf r* **and** *INJ*: *inj-on f* (*Field r*)
shows *wf*(*dir-image r f*)
proof(*unfold wf-eq-minimal2*, *intro allI impI*, *elim conjE*)
fix $A'::'b\ set$
assume *SUB*: $A' \leq Field(dir\text{-}image\ r\ f)$ **and** *NE*: $A' \neq \{\}$
obtain *A* **where** *A-def*: $A = \{a \in Field\ r.\ f\ a \in A'\}$ **by** *blast*
have $A \neq \{\} \wedge A \leq Field\ r$
using *A-def dir-image-Field*[*of r f*] *SUB NE* **by** *blast*
then obtain *a* **where** *1*: $a \in A \wedge (\forall b \in A.\ (b,a) \notin r)$
using *WF* **unfolding** *wf-eq-minimal2* **by** *blast*
have $\forall b' \in A'.\ (b',f\ a) \notin dir\text{-}image\ r\ f$

proof(*clarify*)
  fix b' **assume** $*$: $b' \in A'$ **and** $**$: $(b',f\ a) \in$ *dir-image r f*
  obtain *b1 a1* **where** *2*: $b' = f\ b1 \wedge f\ a = f\ a1$ **and**
    3: $(b1,a1) \in r \wedge \{a1,b1\} \leq$ *Field r*
  using $**$ **unfolding** *dir-image-def Field-def* **by** *blast*
  hence $a = a1$ **using** *1 A-def INJ* **unfolding** *inj-on-def* **by** *auto*
  hence $b1 \in A \wedge (b1,a) \in r$ **using** *2 3 A-def* $*$ **by** *auto*
  with *1* **show** *False* **by** *auto*
qed
thus $\exists\, a' \in A'.\ \forall\, b' \in A'.\ (b',\ a') \notin$ *dir-image r f*
using *A-def 1* **by** *blast*
qed

lemma *Well-order-dir-image*:
$[\![$*Well-order r*; *inj-on f (Field r)*$]\!] \Longrightarrow$ *Well-order (dir-image r f)*
using *assms* **unfolding** *well-order-on-def*
using *Linear-order-dir-image*[*of r f*] *wf-dir-image*[*of r − Id f*]
  dir-image-minus-Id[*of f r*]
  subset-inj-on[*of f Field r Field*($r - Id$)]
  mono-Field[*of r − Id r*] **by** *auto*

lemma *dir-image-Field2*:
Refl r $\Longrightarrow$ *Field*(*dir-image r f*) = *f ‘ (Field r)*
unfolding *Field-def dir-image-def refl-on-def Domain-def Range-def* **by** *blast*

lemma *dir-image-bij-betw*:
$[\![$*Well-order r*; *inj-on f (Field r)*$]\!] \Longrightarrow$ *bij-betw f (Field r) (Field (dir-image r f))*
unfolding *bij-betw-def*
by (*auto simp add*: *dir-image-Field2 order-on-defs*)

lemma *dir-image-compat*:
compat r (dir-image r f) f
unfolding *compat-def dir-image-def* **by** *auto*

lemma *dir-image-iso*:
$[\![$*Well-order r*; *inj-on f (Field r)*$]\!] \Longrightarrow$ *iso r (dir-image r f) f*
using *iso-iff3 dir-image-compat dir-image-bij-betw Well-order-dir-image* **by** *blast*

lemma *dir-image-ordIso*:
$[\![$*Well-order r*; *inj-on f (Field r)*$]\!] \Longrightarrow$ *r =o dir-image r f*
unfolding *ordIso-def* **using** *dir-image-iso Well-order-dir-image* **by** *blast*

lemma *Well-order-iso-copy*:
assumes *WELL*: *well-order-on A r* **and** *BIJ*: *bij-betw f A A'*
shows $\exists\, r'$. *well-order-on* A' r' $\wedge$ $r =o\ r'$
proof−
  let *?r'* = *dir-image r f*
  have *1*: *A* = *Field r* ∧ *Well-order r*
  using *WELL rel.well-order-on-Well-order* **by** *blast*
  hence *2*: *iso r ?r' f*
  using *dir-image-iso* **using** *BIJ* **unfolding** *bij-betw-def* **by** *auto*
  hence *f* ‘ (*Field r*) = *Field ?r'* **using** *1 iso-iff*[*of r ?r'*] **by** *blast*
  hence *Field ?r'* = *A'*
  using *1 BIJ* **unfolding** *bij-betw-def* **by** *auto*
  moreover have *Well-order ?r'*
  using *1 Well-order-dir-image BIJ* **unfolding** *bij-betw-def* **by** *blast*
  ultimately show *?thesis* **unfolding** *ordIso-def* **using** *1 2* **by** *blast*
qed

7.7 Ordinal-like sum of two (disjoint) well-orders

This is roughly obtained by "concatenating" the two well-orders – thus, all elements of the first will be smaller than all elements of the second. This construction only makes sense if the fields of the two well-order relations are disjoint.

definition *Osum* :: *'a rel* ⇒ *'a rel* ⇒ *'a rel* (**infix** *Osum 60*)
where
r Osum r' = *r* ∪ *r'* ∪ {(*a*,*a'*). *a* ∈ *Field r* ∧ *a'* ∈ *Field r'*}

abbreviation *Osum2* :: *'a rel* ⇒ *'a rel* ⇒ *'a rel* (**infix** ∪*o 60*)
where *r* ∪*o r'* ≡ *r Osum r'*

lemma *Field-Osum*: *Field*(*r Osum r'*) = *Field r* ∪ *Field r'*
unfolding *Osum-def Field-def* **by** *blast*

lemma *Osum-Refl*:
assumes *FLD*: *Field r Int Field r'* = {} **and**
    REFL: *Refl r* **and** *REFL'*: *Refl r'*
shows *Refl* (*r Osum r'*)
using *assms*
unfolding *refl-on-def Field-Osum* **unfolding** *Osum-def* **by** *blast*

lemma *Osum-trans*:
assumes *FLD*: *Field r Int Field r'* = {} **and**
    TRANS: *trans r* **and** *TRANS'*: *trans r'*
shows *trans* (*r Osum r'*)
proof(*unfold trans-def*, *auto*)

fix $x\ y\ z$ **assume** $*$: $(x,\ y) \in r \cup o\ r'$ **and** $**$: $(y,\ z) \in r \cup o\ r'$
show $(x,\ z) \in r \cup o\ r'$
proof−
{**assume** *Case1*: $(x,y) \in r$
hence *1*: $x \in$ *Field* $r \wedge y \in$ *Field* r **unfolding** *Field-def* **by** *auto*
have *?thesis*
proof−
{**assume** *Case11*: $(y,z) \in r$
hence $(x,z) \in r$ **using** *Case1 TRANS trans-def*[*of* r] **by** *blast*
hence *?thesis* **unfolding** *Osum-def* **by** *auto*
}
moreover
{**assume** *Case12*: $(y,z) \in r'$
hence $y \in$ *Field* r' **unfolding** *Field-def* **by** *auto*
hence *False* **using** *FLD 1* **by** *auto*
}
moreover
{**assume** *Case13*: $z \in$ *Field* r'
hence *?thesis* **using** *1* **unfolding** *Osum-def* **by** *auto*
}
ultimately show *?thesis* **using** $**$ **unfolding** *Osum-def* **by** *blast*
qed
}
moreover
{**assume** *Case2*: $(x,y) \in r'$
hence *2*: $x \in$ *Field* $r' \wedge y \in$ *Field* r' **unfolding** *Field-def* **by** *auto*
have *?thesis*
proof−
{**assume** *Case21*: $(y,z) \in r$
hence $y \in$ *Field* r **unfolding** *Field-def* **by** *auto*
hence *False* **using** *FLD 2* **by** *auto*
}
moreover
{**assume** *Case22*: $(y,z) \in r'$
hence $(x,z) \in r'$ **using** *Case2 TRANS' trans-def*[*of* r'] **by** *blast*
hence *?thesis* **unfolding** *Osum-def* **by** *auto*
}
moreover
{**assume** *Case23*: $y \in$ *Field* r
hence *False* **using** *FLD 2* **by** *auto*
}
ultimately show *?thesis* **using** $**$ **unfolding** *Osum-def* **by** *blast*
qed
}
moreover
{**assume** *Case3*: $x \in$ *Field* $r \wedge y \in$ *Field* r'
have *?thesis*
proof−
{**assume** *Case31*: $(y,z) \in r$

```
      hence y ∈ Field r unfolding Field-def by auto
      hence False using FLD Case3 by auto
     }
     moreover
     {assume Case32: (y,z) ∈ r′
      hence z ∈ Field r′ unfolding Field-def by blast
      hence ?thesis unfolding Osum-def using Case3 by auto
     }
     moreover
     {assume Case33: y ∈ Field r
      hence False using FLD Case3 by auto
     }
     ultimately show ?thesis using ** unfolding Osum-def by blast
   qed
  }
  ultimately show ?thesis using * unfolding Osum-def by blast
 qed
qed
```

```
lemma Osum-Preorder:
⟦Field r Int Field r′ = {}; Preorder r; Preorder r′⟧ ⟹ Preorder (r Osum r′)
unfolding preorder-on-def using Osum-Refl Osum-trans by blast
```

```
lemma Osum-antisym:
assumes FLD: Field r Int Field r′ = {} and
        AN: antisym r and AN′: antisym r′
shows antisym (r Osum r′)
proof(unfold antisym-def, auto)
  fix x y assume *: (x, y) ∈ r ∪o r′ and **: (y, x) ∈ r ∪o r′
  show x = y
  proof−
   {assume Case1: (x,y) ∈ r
    hence 1: x ∈ Field r ∧ y ∈ Field r unfolding Field-def by auto
    have ?thesis
    proof−
     have (y,x) ∈ r ⟹ ?thesis
     using Case1 AN antisym-def[of r] by blast
     moreover
     {assume (y,x) ∈ r′
      hence y ∈ Field r′ unfolding Field-def by auto
      hence False using FLD 1 by auto
     }
     moreover
     have x ∈ Field r′ ⟹ False using FLD 1 by auto
     ultimately show ?thesis using ** unfolding Osum-def by blast
    qed
   }
```

moreover
{**assume** *Case2*: $(x,y) \in r'$
hence *2*: $x \in$ *Field* $r' \wedge y \in$ *Field* r' **unfolding** *Field-def* **by** *auto*
have *?thesis*
proof−
{**assume** $(y,x) \in r$
hence $y \in$ *Field* r **unfolding** *Field-def* **by** *auto*
hence *False* **using** *FLD 2* **by** *auto*
}
moreover
have $(y,x) \in r' \Longrightarrow$ *?thesis*
using *Case2 AN′ antisym-def*[*of r′*] **by** *blast*
moreover
{**assume** $y \in$ *Field* r
hence *False* **using** *FLD 2* **by** *auto*
}
ultimately show *?thesis* **using** $**$ **unfolding** *Osum-def* **by** *blast*
qed
}
moreover
{**assume** *Case3*: $x \in$ *Field* $r \wedge y \in$ *Field* r'
have *?thesis*
proof−
{**assume** $(y,x) \in r$
hence $y \in$ *Field* r **unfolding** *Field-def* **by** *auto*
hence *False* **using** *FLD Case3* **by** *auto*
}
moreover
{**assume** *Case32*: $(y,x) \in r'$
hence $x \in$ *Field* r' **unfolding** *Field-def* **by** *blast*
hence *False* **using** *FLD Case3* **by** *auto*
}
moreover
have $\neg\ y \in$ *Field* r **using** *FLD Case3* **by** *auto*
ultimately show *?thesis* **using** $**$ **unfolding** *Osum-def* **by** *blast*
qed
}
ultimately show *?thesis* **using** $*$ **unfolding** *Osum-def* **by** *blast*
qed
qed

lemma *Osum-Partial-order*:
$[\![$*Field* r *Int Field* $r' = \{\}$; *Partial-order* r; *Partial-order* $r'$$]\!] \Longrightarrow$
Partial-order $(r$ *Osum* $r')$
unfolding *partial-order-on-def* **using** *Osum-Preorder Osum-antisym* **by** *blast*

lemma *Osum-Total*:

assumes *FLD*: *Field r Int Field r'* = {} **and**
TOT: *Total r* **and** *TOT'*: *Total r'*
shows *Total* (*r Osum r'*)
using *assms*
unfolding *total-on-def Field-Osum* **unfolding** *Osum-def* **by** *blast*

lemma *Osum-Linear-order*:
⟦*Field r Int Field r'* = {}; *Linear-order r*; *Linear-order r'*⟧ ⟹
Linear-order (*r Osum r'*)
unfolding *linear-order-on-def* **using** *Osum-Partial-order Osum-Total* **by** *blast*

lemma *Osum-wf*:
assumes *FLD*: *Field r Int Field r'* = {} **and**
WF: *wf r* **and** *WF'*: *wf r'*
shows *wf* (*r Osum r'*)
unfolding *wf-eq-minimal2* **unfolding** *Field-Osum*
proof(*intro allI impI*, *elim conjE*)
 fix *A* **assume** $*$: $A \subseteq$ *Field r* ∪ *Field r'* **and** $**$: $A \neq \{\}$
 obtain *B* **where** *B-def*: *B* = *A Int Field r* **by** *blast*
 show ∃ *a*∈*A*. ∀ *a'*∈*A*. (*a'*, *a*) ∉ *r* ∪*o r'*
 proof(*cases B* = {})
  assume *Case1*: *B* ≠ {}
  hence *B* ≠ {} ∧ *B* ≤ *Field r* **using** *B-def* **by** *auto*
  then obtain *a* **where** *1*: *a* ∈ *B* **and** *2*: ∀ *a1* ∈ *B*. (*a1*,*a*) ∉ *r*
  using *WF* **unfolding** *wf-eq-minimal2* **by** *blast*
  hence *3*: *a* ∈ *Field r* ∧ *a* ∉ *Field r'* **using** *B-def FLD* **by** *auto*

  have ∀ *a1* ∈ *A*. (*a1*,*a*) ∉ *r Osum r'*
  proof(*intro ballI*)
   fix *a1* **assume** $**$: *a1* ∈ *A*
   {**assume** *Case11*: *a1* ∈ *Field r*
    hence (*a1*,*a*) ∉ *r* **using** *B-def* $**$ *2* **by** *auto*
    moreover
    have (*a1*,*a*) ∉ *r'* **using** *3* **by** (*auto simp add*: *Field-def*)
    ultimately have (*a1*,*a*) ∉ *r Osum r'*
    using *3* **unfolding** *Osum-def* **by** *auto*
   }
   moreover
   {**assume** *Case12*: *a1* ∉ *Field r*
    hence (*a1*,*a*) ∉ *r* **unfolding** *Field-def* **by** *auto*
    moreover
    have (*a1*,*a*) ∉ *r'* **using** *3* **unfolding** *Field-def* **by** *auto*
    ultimately have (*a1*,*a*) ∉ *r Osum r'*
    using *3* **unfolding** *Osum-def* **by** *auto*
   }
   ultimately show (*a1*,*a*) ∉ *r Osum r'* **by** *blast*
  qed

thus *?thesis* **using** *1 B-def* **by** *auto*
next
assume *Case2*: $B = \{\}$
hence *1*: $A \neq \{\} \wedge A \leq$ *Field* r' **using** $*$ $**$ *B-def* **by** *auto*
then obtain a' **where** *2*: $a' \in A$ **and** *3*: $\forall a1' \in A.\ (a1',a') \notin r'$
using *WF'* **unfolding** *wf-eq-minimal2* **by** *blast*
hence *4*: $a' \in$ *Field* $r' \wedge a' \notin$ *Field* r **using** *1 FLD* **by** *blast*

have $\forall a1' \in A.\ (a1',a') \notin r$ *Osum* r'
proof(*unfold Osum-def*, *auto simp add*: *3*)
fix $a1'$ **assume** $(a1', a') \in r$
thus *False* **using** *4* **unfolding** *Field-def* **by** *blast*
next
fix $a1'$ **assume** $a1' \in A$ **and** $a1' \in$ *Field* r
thus *False* **using** *Case2 B-def* **by** *auto*
qed
thus *?thesis* **using** *2* **by** *blast*
qed
qed

lemma *Osum-minus-Id*:
assumes *TOT*: *Total* r **and** *TOT'*: *Total* r' **and**
NID: $\neg$ $(r \leq Id)$ **and** *NID'*: $\neg$ $(r' \leq Id)$
shows $(r$ *Osum* $r') - Id \leq (r - Id)$ *Osum* $(r' - Id)$
proof−
{**fix** a a' **assume** $*$: $(a,a') \in (r$ *Osum* $r')$ **and** $**$: $a \neq a'$
have $(a,a') \in (r - Id)$ *Osum* $(r' - Id)$
proof−
{**assume** $(a,a') \in r \vee (a,a') \in r'$
with $**$ **have** *?thesis* **unfolding** *Osum-def* **by** *auto*
}
moreover
{**assume** $a \in$ *Field* $r \wedge a' \in$ *Field* r'
hence $a \in$ *Field*$(r - Id) \wedge a' \in$ *Field* $(r' - Id)$
using *assms rel.Total-Id-Field* **by** *blast*
hence *?thesis* **unfolding** *Osum-def* **by** *auto*
}
ultimately show *?thesis* **using** $*$ **unfolding** *Osum-def* **by** *blast*
qed
}
thus *?thesis* **by**(*auto simp add*: *Osum-def*)
qed

lemma *wf-Int-Times*:
assumes A *Int* $B = \{\}$
shows $wf(A \times B)$
proof(*unfold wf-def*, *auto*)

fix $P\ x$
assume $*$: $\forall x.\ (\forall y.\ y \in A \wedge x \in B \longrightarrow P\ y) \longrightarrow P\ x$
moreover have $\forall y \in A.\ P\ y$ **using** *assms* $*$ **by** *blast*
ultimately show $P\ x$ **using** $*$ **by** (*case-tac* $x \in B$, *auto*)
qed

lemma *Osum-minus-Id1*:
assumes $r \leq Id$
shows $(r\ Osum\ r') - Id \leq (r' - Id) \cup (Field\ r \times Field\ r')$
proof−
let $?Left = (r\ Osum\ r') - Id$
let $?Right = (r' - Id) \cup (Field\ r \times Field\ r')$
{**fix** $a::'a$ **and** b **assume** $*$: $(a,b) \notin Id$
{**assume** $(a,b) \in r$
with $*$ **have** *False* **using** *assms* **by** *auto*
}
moreover
{**assume** $(a,b) \in r'$
with $*$ **have** $(a,b) \in r' - Id$ **by** *auto*
}
ultimately
have $(a,b) \in ?Left \Longrightarrow (a,b) \in ?Right$
unfolding *Osum-def* **by** *auto*
}
thus *?thesis* **by** *auto*
qed

lemma *Osum-minus-Id2*:
assumes $r' \leq Id$
shows $(r\ Osum\ r') - Id \leq (r - Id) \cup (Field\ r \times Field\ r')$
proof−
let $?Left = (r\ Osum\ r') - Id$
let $?Right = (r - Id) \cup (Field\ r \times Field\ r')$
{**fix** $a::'a$ **and** b **assume** $*$: $(a,b) \notin Id$
{**assume** $(a,b) \in r'$
with $*$ **have** *False* **using** *assms* **by** *auto*
}
moreover
{**assume** $(a,b) \in r$
with $*$ **have** $(a,b) \in r - Id$ **by** *auto*
}
ultimately
have $(a,b) \in ?Left \Longrightarrow (a,b) \in ?Right$
unfolding *Osum-def* **by** *auto*
}
thus *?thesis* **by** *auto*
qed

lemma *Osum-wf-Id*:
assumes *TOT*: *Total* r **and** *TOT'*: *Total* r' **and**
FLD: *Field* r *Int Field* $r' = \{\}$ **and**
WF: $wf(r - Id)$ **and** *WF'*: $wf(r' - Id)$
shows $wf\ ((r\ Osum\ r') - Id)$
proof($cases\ r \leq Id \vee r' \leq Id$)
assume *Case1*: $\neg(r \leq Id \vee r' \leq Id)$
have $Field(r - Id)\ Int\ Field(r' - Id) = \{\}$
using *FLD mono-Field*[*of* $r - Id\ r$] *mono-Field*[*of* $r' - Id\ r'$]
Diff-subset[*of* r *Id*] *Diff-subset*[*of* r' *Id*] **by** *blast*
thus *?thesis*
using *Case1 Osum-minus-Id*[*of* $r\ r'$] *assms Osum-wf*[*of* $r - Id\ r' - Id$]
wf-subset[*of* $(r - Id) \cup o\ (r' - Id)\ (r\ Osum\ r') - Id$] **by** *auto*
next
have *1*: $wf(Field\ r \times Field\ r')$
using *FLD* **by** (*auto simp add*: *wf-Int-Times*)
assume *Case2*: $r \leq Id \vee r' \leq Id$
moreover
{**assume** *Case21*: $r \leq Id$
hence $(r\ Osum\ r') - Id \leq (r' - Id) \cup (Field\ r \times Field\ r')$
using *Osum-minus-Id1*[*of* $r\ r'$] **by** *simp*
moreover
{**have** $Domain(Field\ r \times Field\ r')\ Int\ Range(r' - Id) = \{\}$
using *FLD* **unfolding** *Field-def* **by** *blast*
hence $wf((r' - Id) \cup (Field\ r \times Field\ r'))$
using *1 WF' wf-Un*[*of* $Field\ r \times Field\ r'\ r' - Id$]
by (*auto simp add*: *Un-commute*)
}
ultimately have *?thesis* **by** (*auto simp add*: *wf-subset*)
}
moreover
{**assume** *Case22*: $r' \leq Id$
hence $(r\ Osum\ r') - Id \leq (r - Id) \cup (Field\ r \times Field\ r')$
using *Osum-minus-Id2*[*of* $r'\ r$] **by** *simp*
moreover
{**have** $Range(Field\ r \times Field\ r')\ Int\ Domain(r - Id) = \{\}$
using *FLD* **unfolding** *Field-def* **by** *blast*
hence $wf((r - Id) \cup (Field\ r \times Field\ r'))$
using *1 WF wf-Un*[*of* $r - Id\ Field\ r \times Field\ r'$]
by (*auto simp add*: *Un-commute*)
}
ultimately have *?thesis* **by** (*auto simp add*: *wf-subset*)
}
ultimately show *?thesis* **by** *blast*
qed

lemma *Osum-Well-order*:
assumes *FLD*: *Field r Int Field r'* = {} **and**
WELL: *Well-order r* **and** *WELL'*: *Well-order r'*
shows *Well-order* (*r Osum r'*)
proof−
have *Total r* ∧ *Total r'* **using** *WELL WELL'*
by (*auto simp add*: *order-on-defs*)
thus *?thesis* **using** *assms* **unfolding** *well-order-on-def*
using *Osum-Linear-order Osum-wf-Id* **by** *blast*
qed

lemma *Osum-embed*:
assumes *FLD*: *Field r Int Field r'* = {} **and**
WELL: *Well-order r* **and** *WELL'*: *Well-order r'*
shows *embed r* (*r Osum r'*) *id*
proof−
have *1*: *Well-order* (*r Osum r'*)
using *assms* **by** (*auto simp add*: *Osum-Well-order*)
moreover
have *compat r* (*r Osum r'*) *id*
unfolding *compat-def Osum-def* **by** *auto*
moreover
have *inj-on id* (*Field r*) **by** *simp*
moreover
have *ofilter* (*r Osum r'*) (*Field r*)
using *1* **proof**(*auto simp add*: *wo-rel-def wo-rel.ofilter-def*
Field-Osum rel.under-def)
fix *a b* **assume** *2*: *a* ∈ *Field r* **and** *3*: (*b*,*a*) ∈ *r Osum r'*
moreover
{**assume** (*b*,*a*) ∈ *r'*
hence *a* ∈ *Field r'* **using** *Field-def*[*of r'*] **by** *blast*
hence *False* **using** *2 FLD* **by** *blast*
}
moreover
{**assume** *a* ∈ *Field r'*
hence *False* **using** *2 FLD* **by** *blast*
}
ultimately
show *b* ∈ *Field r* **by** (*auto simp add*: *Osum-def Field-def*)
qed
ultimately show *?thesis*
using *assms* **by** (*auto simp add*: *embed-iff-compat-inj-on-ofilter*)
qed

corollary *Osum-ordLeq*:
assumes *FLD*: *Field r Int Field r'* = {} **and**

    WELL: *Well-order r* **and** *WELL'*: *Well-order r'*
shows $r \leq o\ r\ Osum\ r'$
using *assms Osum-embed Osum-Well-order*
unfolding *ordLeq-def* **by** *blast*

lemma *Well-order-embed-copy*:
assumes *WELL*: *well-order-on A r* **and**
    INJ: *inj-on f A* **and** *SUB*: f ‘ $A \leq B$
shows $\exists\, r'$. *well-order-on B r'* $\wedge\ r \leq o\ r'$
proof−
  have *bij-betw f A* (*f* ‘ *A*)
  using *INJ inj-on-imp-bij-betw* **by** *blast*
  then obtain r'' **where** *well-order-on* (*f* ‘ *A*) r'' **and** *1*: $r =o\ r''$
  using *WELL Well-order-iso-copy* **by** *blast*
  hence *2*: *Well-order* $r'' \wedge$ *Field* r'' = (*f* ‘ *A*)
  using *rel.well-order-on-Well-order* **by** *blast*

  let *?C* = *B* − (*f* ‘ *A*)
  obtain r''' **where** *well-order-on ?C* r'''
  using *well-order-on* **by** *blast*
  hence *3*: *Well-order* $r''' \wedge$ *Field* r''' = *?C*
  using *rel.well-order-on-Well-order* **by** *blast*

  let *?r'* = r'' *Osum* r'''
  have *Field* r'' *Int Field* r''' = {}
  using *2 3* **by** *auto*
  hence $r'' \leq o$ *?r'* **using** *Osum-ordLeq*[*of* r'' r'''] *2 3* **by** *blast*
  hence *4*: $r \leq o$ *?r'* **using** *1 ordIso-ordLeq-trans* **by** *blast*

  hence *Well-order ?r'* **unfolding** *ordLeq-def* **by** *auto*
  moreover
  have *Field ?r'* = *B* **using** *2 3 SUB* **by** (*auto simp add*: *Field-Osum*)
  ultimately show *?thesis* **using** *4* **by** *blast*
qed

7.8 Bounded square

This construction essentially defines, for an order relation *r*, a lexicographic order *bsqr r* on (*Field r*) × (*Field r*), applying the following criteria (in this order):

- compare the maximums;
- compare the first components;
- compare the second components.

The only application of this construction that we are aware of is at proving that the square of an infinite set has the same cardinal as that set. The essential property required there (and which is ensured by this construction) is that any proper order filter of the product order is included in a rectangle, i.e., in a product of proper filters on the original relation (assumed to be a well-order).

definition *bsqr* :: $'a\ rel => ('a * 'a)rel$
where
$bsqr\ r = \{((a1,a2),(b1,b2)).$
$\quad \{a1,a2,b1,b2\} \leq Field\ r \wedge$
$\quad (a1 = b1 \wedge a2 = b2 \vee$
$\quad\ (max2\ r\ a1\ a2,\ max2\ r\ b1\ b2) \in r - Id \vee$
$\quad\ max2\ r\ a1\ a2 = max2\ r\ b1\ b2 \wedge (a1,b1) \in r - Id \vee$
$\quad\ max2\ r\ a1\ a2 = max2\ r\ b1\ b2 \wedge a1 = b1\ \wedge (a2,b2) \in r - Id$
$\quad)\}$

lemma *Field-bsqr*:
$Field\ (bsqr\ r) = Field\ r \times Field\ r$
proof
  show $Field\ (bsqr\ r) \leq Field\ r \times Field\ r$
  proof−
    {**fix** *a1 a2* **assume** $(a1,a2) \in Field\ (bsqr\ r)$
     moreover
     have $\bigwedge b1\ b2.\ ((a1,a2),(b1,b2)) \in bsqr\ r \vee ((b1,b2),(a1,a2)) \in bsqr\ r \Longrightarrow$
        $a1 \in Field\ r \wedge a2 \in Field\ r$ **unfolding** *bsqr-def* **by** *auto*
     ultimately have $a1 \in Field\ r \wedge a2 \in Field\ r$ **unfolding** *Field-def* **by** *auto*
    }
    thus *?thesis* **unfolding** *Field-def* **by** *force*
  qed
next
  show $Field\ r \times Field\ r \leq Field\ (bsqr\ r)$
  proof(*auto*)
    fix *a1 a2* **assume** $a1 \in Field\ r$ **and** $a2 \in Field\ r$
    hence $((a1,a2),(a1,a2)) \in bsqr\ r$ **unfolding** *bsqr-def* **by** *blast*
    thus $(a1,a2) \in Field\ (bsqr\ r)$ **unfolding** *Field-def* **by** *auto*
  qed
qed

lemma *bsqr-Refl*: *Refl*(*bsqr r*)
by(*unfold refl-on-def Field-bsqr*, *auto simp add*: *bsqr-def*)

lemma *bsqr-Trans*:
assumes *Well-order r*
shows *trans* (*bsqr r*)
proof(*unfold trans-def*, *auto*)

  have *Well*: *wo-rel r* **using** *assms wo-rel-def* **by** *auto*
  hence *Trans*: *trans r* **using** *wo-rel.TRANS* **by** *auto*

```
have Anti: antisym r using wo-rel.ANTISYM Well by auto
hence TransS: trans(r − Id) using Trans by (auto simp add: trans-diff-Id)

fix a1 a2 b1 b2 c1 c2
assume ∗: ((a1,a2),(b1,b2)) ∈ bsqr r and ∗∗: ((b1,b2),(c1,c2)) ∈ bsqr r
hence 0: {a1,a2,b1,b2,c1,c2} ≤ Field r unfolding bsqr-def by auto
have 1: a1 = b1 ∧ a2 = b2 ∨ (max2 r a1 a2, max2 r b1 b2) ∈ r − Id ∨
         max2 r a1 a2 = max2 r b1 b2 ∧ (a1,b1) ∈ r − Id ∨
         max2 r a1 a2 = max2 r b1 b2 ∧ a1 = b1 ∧ (a2,b2) ∈ r − Id
using ∗ unfolding bsqr-def by auto
have 2: b1 = c1 ∧ b2 = c2 ∨ (max2 r b1 b2, max2 r c1 c2) ∈ r − Id ∨
         max2 r b1 b2 = max2 r c1 c2 ∧ (b1,c1) ∈ r − Id ∨
         max2 r b1 b2 = max2 r c1 c2 ∧ b1 = c1 ∧ (b2,c2) ∈ r − Id
using ∗∗ unfolding bsqr-def by auto
show ((a1,a2),(c1,c2)) ∈ bsqr r
proof−
  {assume Case1: a1 = b1 ∧ a2 = b2
   hence ?thesis using ∗∗ by simp
  }
  moreover
  {assume Case2: (max2 r a1 a2, max2 r b1 b2) ∈ r − Id
   {assume Case21: b1 = c1 ∧ b2 = c2
    hence ?thesis using ∗ by simp
   }
   moreover
   {assume Case22: (max2 r b1 b2, max2 r c1 c2) ∈ r − Id
    hence (max2 r a1 a2, max2 r c1 c2) ∈ r − Id
    using Case2 TransS trans-def[of r − Id] by blast
    hence ?thesis using 0 unfolding bsqr-def by auto
   }
   moreover
   {assume Case23-4: max2 r b1 b2 = max2 r c1 c2
    hence ?thesis using Case2 0 unfolding bsqr-def by auto
   }
   ultimately have ?thesis using 0 2 by auto
  }
  moreover
  {assume Case3: max2 r a1 a2 = max2 r b1 b2 ∧ (a1,b1) ∈ r − Id
   {assume Case31: b1 = c1 ∧ b2 = c2
    hence ?thesis using ∗ by simp
   }
   moreover
   {assume Case32: (max2 r b1 b2, max2 r c1 c2) ∈ r − Id
    hence ?thesis using Case3 0 unfolding bsqr-def by auto
   }
   moreover
   {assume Case33: max2 r b1 b2 = max2 r c1 c2 ∧ (b1,c1) ∈ r − Id
    hence (a1,c1) ∈ r − Id
    using Case3 TransS trans-def[of r − Id] by blast
```

hence *?thesis* **using** *Case3 Case33 0* **unfolding** *bsqr-def* **by** *auto*
}
moreover
{**assume** *Case33*: $max2\ r\ b1\ b2 = max2\ r\ c1\ c2 \wedge b1 = c1$
hence *?thesis* **using** *Case3 0* **unfolding** *bsqr-def* **by** *auto*
}
ultimately have *?thesis* **using** *0 2* **by** *auto*
}
moreover
{**assume** *Case4*: $max2\ r\ a1\ a2 = max2\ r\ b1\ b2 \wedge a1 = b1 \wedge (a2,b2) \in r - Id$
{**assume** *Case41*: $b1 = c1 \wedge b2 = c2$
hence *?thesis* **using** $*$ **by** *simp*
}
moreover
{**assume** *Case42*: $(max2\ r\ b1\ b2,\ max2\ r\ c1\ c2) \in r - Id$
hence *?thesis* **using** *Case4 0* **unfolding** *bsqr-def* **by** *auto*
}
moreover
{**assume** *Case43*: $max2\ r\ b1\ b2 = max2\ r\ c1\ c2 \wedge (b1,c1) \in r - Id$
hence *?thesis* **using** *Case4 0* **unfolding** *bsqr-def* **by** *auto*
}
moreover
{**assume** *Case44*: $max2\ r\ b1\ b2 = max2\ r\ c1\ c2 \wedge b1 = c1 \wedge (b2,c2) \in r -$
Id
hence $(a2,c2) \in r - Id$
using *Case4 TransS trans-def*[*of* $r - Id$] **by** *blast*
hence *?thesis* **using** *Case4 Case44 0* **unfolding** *bsqr-def* **by** *auto*
}
ultimately have *?thesis* **using** *0 2* **by** *auto*
}
ultimately show *?thesis* **using** *0 1* **by** *auto*
qed
qed

lemma *bsqr-antisym*:
assumes *Well-order r*
shows *antisym* (*bsqr r*)
proof(*unfold antisym-def*, *clarify*)

have *Well*: *wo-rel r* **using** *assms wo-rel-def* **by** *auto*
hence *Trans*: *trans r* **using** *wo-rel.TRANS* **by** *auto*
have *Anti*: *antisym r* **using** *wo-rel.ANTISYM Well* **by** *auto*
hence *TransS*: *trans*$(r - Id)$ **using** *Trans* **by** (*auto simp add*: *trans-diff-Id*)
hence *IrrS*: $\forall\ a\ b.\ \neg((a,b) \in r - Id \wedge (b,a) \in r - Id)$
using *Anti trans-def*[*of* $r - Id$] *antisym-def*[*of* $r - Id$] **by** *blast*

fix *a1 a2 b1 b2*
assume $*$: $((a1,a2),(b1,b2)) \in bsqr\ r$ **and** $**$: $((b1,b2),(a1,a2)) \in bsqr\ r$

hence *0*: $\{a1,a2,b1,b2\} \leq$ *Field r* **unfolding** *bsqr-def* **by** *auto*
have *1*: $a1 = b1 \land a2 = b2 \lor (max2\ r\ a1\ a2,\ max2\ r\ b1\ b2) \in r - Id \lor$
$max2\ r\ a1\ a2 = max2\ r\ b1\ b2 \land (a1,b1) \in r - Id \lor$
$max2\ r\ a1\ a2 = max2\ r\ b1\ b2 \land a1 = b1 \land (a2,b2) \in r - Id$
using $*$ **unfolding** *bsqr-def* **by** *auto*
have *2*: $b1 = a1 \land b2 = a2 \lor (max2\ r\ b1\ b2,\ max2\ r\ a1\ a2) \in r - Id \lor$
$max2\ r\ b1\ b2 = max2\ r\ a1\ a2 \land (b1,a1) \in r - Id \lor$
$max2\ r\ b1\ b2 = max2\ r\ a1\ a2 \land b1 = a1 \land (b2,a2) \in r - Id$
using $**$ **unfolding** *bsqr-def* **by** *auto*
show $a1 = b1 \land a2 = b2$
proof$-$
{assume *Case1*: $(max2\ r\ a1\ a2,\ max2\ r\ b1\ b2) \in r - Id$
{assume *Case11*: $(max2\ r\ b1\ b2,\ max2\ r\ a1\ a2) \in r - Id$
hence *False* **using** *Case1 IrrS* **by** *blast*
}
moreover
{assume *Case12-3*: $max2\ r\ b1\ b2 = max2\ r\ a1\ a2$
hence *False* **using** *Case1* **by** *auto*
}
ultimately have *?thesis* **using** *0 2* **by** *auto*
}
moreover
{assume *Case2*: $max2\ r\ a1\ a2 = max2\ r\ b1\ b2 \land (a1,b1) \in r - Id$
{assume *Case21*: $(max2\ r\ b1\ b2,\ max2\ r\ a1\ a2) \in r - Id$
hence *False* **using** *Case2* **by** *auto*
}
moreover
{assume *Case22*: $(b1,a1) \in r - Id$
hence *False* **using** *Case2 IrrS* **by** *blast*
}
moreover
{assume *Case23*: $b1 = a1$
hence *False* **using** *Case2* **by** *auto*
}
ultimately have *?thesis* **using** *0 2* **by** *auto*
}
moreover
{assume *Case3*: $max2\ r\ a1\ a2 = max2\ r\ b1\ b2 \land a1 = b1 \land (a2,b2) \in r - Id$
moreover
{assume *Case31*: $(max2\ r\ b1\ b2,\ max2\ r\ a1\ a2) \in r - Id$
hence *False* **using** *Case3* **by** *auto*
}
moreover
{assume *Case32*: $(b1,a1) \in r - Id$
hence *False* **using** *Case3* **by** *auto*
}
moreover
{assume *Case33*: $(b2,a2) \in r - Id$
hence *False* **using** *Case3 IrrS* **by** *blast*

}
ultimately have *?thesis* **using** *0 2* **by** *auto*
}
ultimately show *?thesis* **using** *0 1* **by** *blast*
qed
qed

lemma *bsqr-Total*:
assumes *Well-order r*
shows *Total*(*bsqr r*)
proof−

have *Well*: *wo-rel r* **using** *assms wo-rel-def* **by** *auto*
hence *Total*: $\forall a \in Field\ r.\ \forall b \in Field\ r.\ (a,b) \in r \vee (b,a) \in r$
using *wo-rel.TOTALS* **by** *auto*

{**fix** *a1 a2 b1 b2* **assume** $\{(a1,a2), (b1,b2)\} \leq Field(bsqr\ r)$
hence *0*: $a1 \in Field\ r \wedge a2 \in Field\ r \wedge b1 \in Field\ r \wedge b2 \in Field\ r$
using *Field-bsqr* **by** *blast*
have $((a1,a2) = (b1,b2) \vee ((a1,a2),(b1,b2)) \in bsqr\ r \vee ((b1,b2),(a1,a2)) \in bsqr\ r)$
proof(*rule wo-rel.cases-Total*[*of r a1 a2*], *clarsimp simp add*: *Well*, *simp add*: *0*)

assume *Case1*: $(a1,a2) \in r$
hence *1*: $max2\ r\ a1\ a2 = a2$
using *Well 0* **by** (*auto simp add*: *wo-rel.max2-equals2*)
show *?thesis*
proof(*rule wo-rel.cases-Total*[*of r b1 b2*], *clarsimp simp add*: *Well*, *simp add*: *0*)
assume *Case11*: $(b1,b2) \in r$
hence *2*: $max2\ r\ b1\ b2 = b2$
using *Well 0* **by** (*auto simp add*: *wo-rel.max2-equals2*)
show *?thesis*
proof(*rule wo-rel.cases-Total3*[*of r a2 b2*], *clarsimp simp add*: *Well*, *simp add*: *0*)
assume *Case111*: $(a2,b2) \in r - Id \vee (b2,a2) \in r - Id$
thus *?thesis* **using** *0 1 2* **unfolding** *bsqr-def* **by** *auto*
next
assume *Case112*: $a2 = b2$
show *?thesis*
proof(*rule wo-rel.cases-Total3*[*of r a1 b1*], *clarsimp simp add*: *Well*, *simp add*: *0*)
assume *Case1121*: $(a1,b1) \in r - Id \vee (b1,a1) \in r - Id$
thus *?thesis* **using** *0 1 2 Case112* **unfolding** *bsqr-def* **by** *auto*
next
assume *Case1122*: $a1 = b1$
thus *?thesis* **using** *Case112* **by** *auto*

```
        qed
      qed
    next
      assume Case12: (b2,b1) ∈ r
    hence 3: max2 r b1 b2 = b1 using Well 0 by (auto simp add: wo-rel.max2-equals1)
      show ?thesis
       proof(rule wo-rel.cases-Total3[of r a2 b1], clarsimp simp add: Well, simp
add: 0)
        assume Case121: (a2,b1) ∈ r − Id ∨ (b1,a2) ∈ r − Id
        thus ?thesis using 0 1 3 unfolding bsqr-def by auto
      next
        assume Case122: a2 = b1
        show ?thesis
        proof(rule wo-rel.cases-Total3[of r a1 b1], clarsimp simp add: Well, simp
add: 0)
          assume Case1221: (a1,b1) ∈ r − Id ∨ (b1,a1) ∈ r − Id
          thus ?thesis using 0 1 3 Case122 unfolding bsqr-def by auto
        next
          assume Case1222: a1 = b1
          show ?thesis
         proof(rule wo-rel.cases-Total3[of r a2 b2], clarsimp simp add: Well, simp
add: 0)
            assume Case12221: (a2,b2) ∈ r − Id ∨ (b2,a2) ∈ r − Id
             thus ?thesis using 0 1 3 Case122 Case1222 unfolding bsqr-def by
auto
          next
            assume Case12222: a2 = b2
            thus ?thesis using Case122 Case1222 by auto
          qed
        qed
      qed
    qed
  next
    assume Case2: (a2,a1) ∈ r
   hence 1: max2 r a1 a2 = a1 using Well 0 by (auto simp add: wo-rel.max2-equals1)
    show ?thesis
    proof(rule wo-rel.cases-Total[of r b1 b2], clarsimp simp add: Well, simp add:
0)
      assume Case21: (b1,b2) ∈ r
    hence 2: max2 r b1 b2 = b2 using Well 0 by (auto simp add: wo-rel.max2-equals2)
      show ?thesis
       proof(rule wo-rel.cases-Total3[of r a1 b2], clarsimp simp add: Well, simp
add: 0)
        assume Case211: (a1,b2) ∈ r − Id ∨ (b2,a1) ∈ r − Id
        thus ?thesis using 0 1 2 unfolding bsqr-def by auto
      next
        assume Case212: a1 = b2
        show ?thesis
        proof(rule wo-rel.cases-Total3[of r a1 b1], clarsimp simp add: Well, simp
```

```
add: 0)
          assume Case2121: (a1,b1) ∈ r − Id ∨ (b1,a1) ∈ r − Id
          thus ?thesis using 0 1 2 Case212 unfolding bsqr-def by auto
        next
          assume Case2122: a1 = b1
          show ?thesis
         proof(rule wo-rel.cases-Total3[of r a2 b2], clarsimp simp add: Well, simp
add: 0)
            assume Case21221: (a2,b2) ∈ r − Id ∨ (b2,a2) ∈ r − Id
              thus ?thesis using 0 1 2 Case212 Case2122 unfolding bsqr-def by
auto
          next
            assume Case21222: a2 = b2
            thus ?thesis using Case2122 Case212 by auto
          qed
        qed
      qed
    next
      assume Case22: (b2,b1) ∈ r
    hence 3: max2 r b1 b2 = b1 using Well 0 by (auto simp add: wo-rel.max2-equals1)
      show ?thesis
       proof(rule wo-rel.cases-Total3[of r a1 b1], clarsimp simp add: Well, simp
add: 0)
        assume Case221: (a1,b1) ∈ r − Id ∨ (b1,a1) ∈ r − Id
        thus ?thesis using 0 1 3 unfolding bsqr-def by auto
      next
        assume Case222: a1 = b1
        show ?thesis
        proof(rule wo-rel.cases-Total3[of r a2 b2], clarsimp simp add: Well, simp
add: 0)
          assume Case2221: (a2,b2) ∈ r − Id ∨ (b2,a2) ∈ r − Id
          thus ?thesis using 0 1 3 Case222 unfolding bsqr-def by auto
        next
          assume Case2222: a2 = b2
          thus ?thesis using Case222 by auto
        qed
      qed
    qed
  qed
 }
 thus ?thesis unfolding total-on-def by fast
qed


lemma bsqr-Linear-order:
assumes Well-order r
shows Linear-order(bsqr r)
unfolding order-on-defs
using assms bsqr-Refl bsqr-Trans bsqr-antisym bsqr-Total by blast
```

lemma *bsqr-Well-order*:
assumes *Well-order r*
shows *Well-order*(*bsqr r*)
using *assms*
proof(*simp add*: *bsqr-Linear-order Linear-order-Well-order-iff*, *intro allI impI*)
have *0*: $\forall A \leq$ *Field r*. $A \neq \{\} \longrightarrow (\exists a \in A.\ \forall a' \in A.\ (a,a') \in r)$
using *assms well-order-on-def Linear-order-Well-order-iff* **by** *blast*
fix *D* **assume** $*$: $D \leq$ *Field* (*bsqr r*) **and** $**$: $D \neq \{\}$
hence *1*: $D \leq$ *Field r* $\times$ *Field r* **unfolding** *Field-bsqr* **by** *simp*

obtain *M* **where** *M-def*: $M = \{max2\ r\ a1\ a2|\ a1\ a2.\ (a1,a2) \in D\}$ **by** *blast*
have $M \neq \{\}$ **using** *1 M-def* $**$ **by** *auto*
moreover
have $M \leq$ *Field r* **unfolding** *M-def*
using *1 assms wo-rel-def*[*of r*] *wo-rel.max2-among*[*of r*] **by** *fastforce*
ultimately obtain *m* **where** *m-min*: $m \in M \wedge (\forall a \in M.\ (m,a) \in r)$
using *0* **by** *blast*

obtain *A1* **where** *A1-def*: $A1 = \{a1.\ \exists a2.\ (a1,a2) \in D \wedge max2\ r\ a1\ a2 = m\}$
by *blast*
have $A1 \leq$ *Field r* **unfolding** *A1-def* **using** *1* **by** *auto*
moreover have $A1 \neq \{\}$ **unfolding** *A1-def* **using** *m-min* **unfolding** *M-def*
by *blast*
ultimately obtain *a1* **where** *a1-min*: $a1 \in A1 \wedge (\forall a \in A1.\ (a1,a) \in r)$
using *0* **by** *blast*

obtain *A2* **where** *A2-def*: $A2 = \{a2.\ (a1,a2) \in D \wedge max2\ r\ a1\ a2 = m\}$ **by**
blast
have $A2 \leq$ *Field r* **unfolding** *A2-def* **using** *1* **by** *auto*
moreover have $A2 \neq \{\}$ **unfolding** *A2-def*
using *m-min a1-min* **unfolding** *A1-def M-def* **by** *blast*
ultimately obtain *a2* **where** *a2-min*: $a2 \in A2 \wedge (\forall a \in A2.\ (a2,a) \in r)$
using *0* **by** *blast*

have *2*: $max2\ r\ a1\ a2 = m$
using *a1-min a2-min* **unfolding** *A1-def A2-def* **by** *auto*
have *3*: $(a1,a2) \in D$ **using** *a2-min* **unfolding** *A2-def* **by** *auto*

moreover
{fix *b1 b2* **assume** $***$: $(b1,b2) \in D$
hence *4*: $\{a1,a2,b1,b2\} \leq$ *Field r* **using** *1 3* **by** *blast*
have *5*: $(max2\ r\ a1\ a2,\ max2\ r\ b1\ b2) \in r$
using $***$ *a1-min a2-min m-min* **unfolding** *A1-def A2-def M-def* **by** *auto*
have $((a1,a2),(b1,b2)) \in bsqr\ r$
proof(*cases* $max2\ r\ a1\ a2 = max2\ r\ b1\ b2$)
assume *Case1*: $max2\ r\ a1\ a2 \neq max2\ r\ b1\ b2$
thus *?thesis* **unfolding** *bsqr-def* **using** *4 5* **by** *auto*

next
assume *Case2*: *max2 r a1 a2* = *max2 r b1 b2*
hence *b1* ∈ *A1* **unfolding** *A1-def* **using** *2* ∗∗∗ **by** *auto*
hence *6*: (*a1*,*b1*) ∈ *r* **using** *a1-min* **by** *auto*
show *?thesis*
proof(*cases a1* = *b1*)
assume *Case21*: *a1* ≠ *b1*
thus *?thesis* **unfolding** *bsqr-def* **using** *4 Case2 6* **by** *auto*
next
assume *Case22*: *a1* = *b1*
hence *b2* ∈ *A2* **unfolding** *A2-def* **using** *2* ∗∗∗ *Case2* **by** *auto*
hence *7*: (*a2*,*b2*) ∈ *r* **using** *a2-min* **by** *auto*
thus *?thesis* **unfolding** *bsqr-def* **using** *4 7 Case2 Case22* **by** *auto*
qed
qed
}

ultimately show ∃ *d* ∈ *D*. ∀ *d′* ∈ *D*. (*d*,*d′*) ∈ *bsqr r* **by** *fastforce*
qed

lemma *bsqr-max2*:
assumes *WELL*: *Well-order r* **and** *LEQ*: ((*a1*,*a2*),(*b1*,*b2*)) ∈ *bsqr r*
shows (*max2 r a1 a2*, *max2 r b1 b2*) ∈ *r*
proof−
have {(*a1*,*a2*),(*b1*,*b2*)} ≤ *Field*(*bsqr r*)
using *LEQ* **unfolding** *Field-def* **by** *auto*
hence {*a1*,*a2*,*b1*,*b2*} ≤ *Field r* **unfolding** *Field-bsqr* **by** *auto*
hence {*max2 r a1 a2*, *max2 r b1 b2*} ≤ *Field r*
using *WELL wo-rel-def*[*of r*] *wo-rel.max2-among*[*of r*] **by** *fastforce*
moreover have (*max2 r a1 a2*, *max2 r b1 b2*) ∈ *r* ∨ *max2 r a1 a2* = *max2 r b1 b2*
using *LEQ* **unfolding** *bsqr-def* **by** *auto*
ultimately show *?thesis* **using** *WELL* **unfolding** *order-on-defs refl-on-def* **by** *auto*
qed

lemma *bsqr-ofilter*:
assumes *WELL*: *Well-order r* **and**
OF: *ofilter* (*bsqr r*) *D* **and** *SUB*: *D* < *Field r* × *Field r* **and**
NE: ¬ (∃ *a*. *Field r* = *under r a*)
shows ∃ *A*. *ofilter r A* ∧ *A* < *Field r* ∧ *D* ≤ *A* × *A*
proof−
let *?r′* = *bsqr r*
have *Well*: *wo-rel r* **using** *WELL wo-rel-def* **by** *blast*
hence *Trans*: *trans r* **using** *wo-rel.TRANS* **by** *blast*
have *Well′*: *Well-order ?r′* ∧ *wo-rel ?r′*
using *WELL bsqr-Well-order wo-rel-def* **by** *blast*

have $D < Field\ ?r'$ **unfolding** *Field-bsqr* **using** *SUB* **.**
with *OF* **obtain** *a1* **and** *a2* **where**
$(a1,a2) \in Field\ ?r'$ **and** 1: $D = underS\ ?r'\ (a1,a2)$
using *Well′ wo-rel.ofilter-underS-Field*[*of ?r′ D*] **by** *auto*
hence 2: $\{a1,a2\} \leq Field\ r$ **unfolding** *Field-bsqr* **by** *auto*
let $?m = max2\ r\ a1\ a2$
have $D \leq (under\ r\ ?m) \times (under\ r\ ?m)$
proof(*unfold 1*)
{**fix** *b1 b2*
let $?n = max2\ r\ b1\ b2$
assume $(b1,b2) \in underS\ ?r'\ (a1,a2)$
hence 3: $((b1,b2),(a1,a2)) \in ?r'$
unfolding *rel.underS-def* **by** *blast*
hence $(?n,?m) \in r$ **using** *WELL* **by** (*auto simp add*: *bsqr-max2*)
moreover
{**have** $(b1,b2) \in Field\ ?r'$ **using** *3* **unfolding** *Field-def* **by** *auto*
hence $\{b1,b2\} \leq Field\ r$ **unfolding** *Field-bsqr* **by** *auto*
hence $(b1,?n) \in r \wedge (b2,?n) \in r$
using *Well* **by** (*auto simp add*: *wo-rel.max2-greater*)
}
ultimately have $(b1,?m) \in r \wedge (b2,?m) \in r$
using *Trans trans-def*[*of r*] **by** *blast*
hence $(b1,b2) \in (under\ r\ ?m) \times (under\ r\ ?m)$ **unfolding** *rel.under-def* **by** *simp*}
thus $underS\ ?r'\ (a1,a2) \leq (under\ r\ ?m) \times (under\ r\ ?m)$ **by** *auto*
qed
moreover have *ofilter r* $(under\ r\ ?m)$
using *Well* **by** (*auto simp add*: *wo-rel.under-ofilter*)
moreover have $under\ r\ ?m < Field\ r$
using *NE rel.under-Field*[*of r ?m*] **by** *blast*
ultimately show *?thesis* **by** *blast*
qed

7.9 The maxim among a finite set of ordinals

The correct phrazing would be "a maxim of ...", as $\leq o$ is only a preorder.

definition *isOmax* :: $'a\ rel\ set \Rightarrow 'a\ rel \Rightarrow bool$
where
$isOmax\ \ R\ r == r \in R \wedge (ALL\ r' : R.\ r' \leq o\ r)$

definition *omax* :: $'a\ rel\ set \Rightarrow 'a\ rel$
where
$omax\ R == SOME\ r.\ isOmax\ R\ r$

lemma *exists-isOmax*:
assumes *finite R* **and** $R \neq \{\}$ **and** $\forall\ r \in R.\ Well\text{-}order\ r$

shows $\exists\ r.\ isOmax\ R\ r$
proof−
  have *finite R* $\Longrightarrow R \neq \{\} \longrightarrow (\forall\ r \in R.\ Well\text{-}order\ r) \longrightarrow (\exists\ r.\ isOmax\ R\ r)$
  apply(*erule finite-induct*) **apply**(*simp add*: *isOmax-def*)
  proof(*clarsimp*)
    fix *r R* **assume** $*$: *finite R* **and** $**$: $r \notin R$
    and $***$: *Well-order r* **and** $****$: $\forall\, r{\in}R.\ Well\text{-}order\ r$
    and *IH*: $R \neq \{\} \longrightarrow (\exists\, p.\ isOmax\ R\ p)$
    let $?R' = insert\ r\ R$
    show $\exists\, p'.\ (isOmax\ ?R'\ p')$
    proof(*cases* $R = \{\}$)
      assume *Case1*: $R = \{\}$
      thus *?thesis* **unfolding** *isOmax-def* **using** $***$
      by (*simp add*: *ordLeq-reflexive*)
    next
      assume *Case2*: $R \neq \{\}$
      then obtain *p* **where** *p*: *isOmax R p* **using** *IH* **by** *auto*
      hence *1*: *Well-order p* **using** $****$ **unfolding** *isOmax-def* **by** *simp*
      {assume *Case21*: $r \leq o\ p$
       hence *isOmax ?R′ p* **using** *p* **unfolding** *isOmax-def* **by** *simp*
       hence *?thesis* **by** *auto*
      }
      moreover
      {assume *Case22*: $p \leq o\ r$
       {fix r' **assume** $r' \in ?R'$
        moreover
        {assume $r' \in R$
         hence $r' \leq o\ p$ **using** *p* **unfolding** *isOmax-def* **by** *simp*
         hence $r' \leq o\ r$ **using** *Case22* **by**(*rule ordLeq-transitive*)
        }
        moreover have $r \leq o\ r$ **using** $***$ **by**(*rule ordLeq-reflexive*)
        ultimately have $r' \leq o\ r$ **by** *auto*
       }
       hence *isOmax ?R′ r* **unfolding** *isOmax-def* **by** *simp*
       hence *?thesis* **by** *auto*
      }
      moreover have $r \leq o\ p \vee p \leq o\ r$
      using *1* $***$ *ordLeq-total* **by** *auto*
      ultimately show *?thesis* **by** *blast*
    qed
  qed
  thus *?thesis* **using** *assms* **by** *auto*
qed

lemma *omax-isOmax*:
assumes *finite R* **and** $R \neq \{\}$ **and** $\forall\ r \in R.\ Well\text{-}order\ r$
shows *isOmax R* (*omax R*)
unfolding *omax-def* **using** *assms*

by(*simp add*: *exists-isOmax someI-ex*)

lemma *omax-in*:
assumes *finite R* **and** $R \neq \{\}$ **and** $\forall\ r \in R.\ Well\text{-}order\ r$
shows *omax* $R \in R$
using *assms omax-isOmax* **unfolding** *isOmax-def* **by** *blast*

lemma *Well-order-omax*:
assumes *finite R* **and** $R \neq \{\}$ **and** $\forall r{\in}R.\ Well\text{-}order\ r$
shows *Well-order* (*omax R*)
using *assms* **apply** − **apply**(*drule omax-in*) **by** *auto*

lemma *omax-maxim*:
assumes *finite R* **and** $\forall\ r \in R.\ Well\text{-}order\ r$ **and** $r \in R$
shows $r \leq o$ *omax R*
using *assms omax-isOmax* **unfolding** *isOmax-def* **by** *blast*

lemma *omax-ordLeq*:
assumes *finite R* **and** $R \neq \{\}$ **and** $*$: $\forall\ r \in R.\ r \leq o\ p$
shows *omax R* $\leq o\ p$
proof−
  have $\forall\ r \in R.\ Well\text{-}order\ r$ **using** $*$ **unfolding** *ordLeq-def* **by** *simp*
  thus *?thesis* **using** *assms omax-in* **by** *auto*
qed

lemma *omax-ordLess*:
assumes *finite R* **and** $R \neq \{\}$ **and** $*$: $\forall\ r \in R.\ r < o\ p$
shows *omax R* $< o\ p$
proof−
  have $\forall\ r \in R.\ Well\text{-}order\ r$ **using** $*$ **unfolding** *ordLess-def* **by** *simp*
  thus *?thesis* **using** *assms omax-in* **by** *auto*
qed

lemma *omax-ordLeq-elim*:
assumes *finite R* **and** $\forall\ r \in R.\ Well\text{-}order\ r$
and *omax R* $\leq o\ p$ **and** $r \in R$
shows $r \leq o\ p$
using *assms omax-maxim*[*of R r*] **apply** *simp*
using *ordLeq-transitive* **by** *blast*

lemma *omax-ordLess-elim*:
assumes *finite R* **and** $\forall\ r \in R.\ Well\text{-}order\ r$

and *omax* $R <o\ p$ **and** $r \in R$
shows $r <o\ p$
using *assms omax-maxim*[*of* R r] **apply** *simp*
using *ordLeq-ordLess-trans* **by** *blast*

lemma *ordLeq-omax*:
assumes *finite* R **and** $\forall\ r \in R.$ *Well-order* r
and $r \in R$ **and** $p \leq o\ r$
shows $p \leq o$ *omax* R
using *assms omax-maxim*[*of* R r] **apply** *simp*
using *ordLeq-transitive* **by** *blast*

lemma *ordLess-omax*:
assumes *finite* R **and** $\forall\ r \in R.$ *Well-order* r
and $r \in R$ **and** $p <o\ r$
shows $p <o$ *omax* R
using *assms omax-maxim*[*of* R r] **apply** *simp*
using *ordLess-ordLeq-trans* **by** *blast*

lemma *omax-ordLeq-mono*:
assumes P: *finite* P **and** R: *finite* R
and *NE-P*: $P \neq \{\}$ **and** *Well-R*: $\forall\ r \in R.$ *Well-order* r
and *LEQ*: $\forall\ p \in P.\ \exists\ r \in R.\ p \leq o\ r$
shows *omax* $P \leq o$ *omax* R
proof−
  let *?mp* = *omax* P **let** *?mr* = *omax* R
  {fix p **assume** $p : P$
   then obtain r **where** r: $r : R$ **and** $p \leq o\ r$
   using *LEQ* **by** *blast*
   moreover have r <=*o* *?mr*
   using *r R Well-R omax-maxim* **by** *blast*
   ultimately have p <=*o* *?mr*
   using *ordLeq-transitive* **by** *blast*
  }
  thus *?mp* <=*o* *?mr*
  using *NE-P P* **using** *omax-ordLeq* **by** *blast*
qed

lemma *omax-ordLess-mono*:
assumes P: *finite* P **and** R: *finite* R
and *NE-P*: $P \neq \{\}$ **and** *Well-R*: $\forall\ r \in R.$ *Well-order* r
and *LEQ*: $\forall\ p \in P.\ \exists\ r \in R.\ p <o\ r$
shows *omax* $P <o$ *omax* R
proof−
  let *?mp* = *omax* P **let** *?mr* = *omax* R

```
{fix p assume p : P
 then obtain r where r: r : R and p <o r
 using LEQ by blast
 moreover have r <=o ?mr
 using r R Well-R omax-maxim by blast
 ultimately have p <o ?mr
 using ordLess-ordLeq-trans by blast
}
thus ?mp <o ?mr
using NE-P P omax-ordLess by blast
qed
```

end

8 Cardinal-order relations

theory *Cardinal-Order-Relation* **imports** *Constructions-on-Wellorders*
begin

In this section, we define cardinal-order relations to be minim well-orders on their field. Then we define the cardinal of a set to be *some* cardinal-order relation on that set, which will be unique up to order isomorphism. Then we study the connection between cardinals and:

- standard set-theoretic constructions: products, sums, unions, lists, powersets, set-of finite sets operator;
- finiteness and infiniteness (in particular, with the numeric cardinal operator for finite sets, *card*, from the theory *Finite-Sets.thy*).

On the way, we define the canonical ω cardinal and finite cardinals. We also define (again, up to order isomorphism) the successor of a cardinal, and show that any cardinal admits a successor.

Main results of this section are the existence of cardinal relations and the facts that, in the presence of infiniteness, most of the standard set-theoretic constructions (except for the powerset) *do not increase cardinality*. In particular, e.g., the set of words/lists over any infinite set has the same cardinality (hence, is in bijection) with that set.

8.1 Cardinal orders

A cardinal order in our setting shall be a well-order *minim* w.r.t. the order-embedding relation, $\leq o$ (which is the same as being *minimal* w.r.t. the strict order-embedding relation, $<o$), among all the well-orders on its field.

definition *card-order-on* :: *'a set* ⇒ *'a rel* ⇒ *bool*
where
card-order-on A r ≡ *well-order-on A r* ∧ (∀ *r'*. *well-order-on A r'* ⟶ *r* ≤*o* *r'*)

abbreviation *Card-order r* ≡ *card-order-on* (*Field r*) *r*
abbreviation *card-order r* ≡ *card-order-on UNIV r*

lemma *card-order-on-well-order-on*[*simp*]:
assumes *card-order-on A r*
shows *well-order-on A r*
using *assms* **unfolding** *card-order-on-def* **by** *simp*

lemma *card-order-on-Card-order*:
card-order-on A r ⟹ *A* = *Field r* ∧ *Card-order r*
unfolding *card-order-on-def* **using** *rel.well-order-on-Field* **by** *blast*

The existence of a cardinal relation on any given set (which will mean that any set has a cardinal) follows from two facts:

- Zermelo's theorem (proved in *Zorn.thy* as theorem *well-order-on*), which states that on any given set there exists a well-order;
- The well-founded-ness of <*o*, ensuring that then there exists a minimal such well-order, i.e., a cardinal order.

theorem *card-order-on*: ∃ *r*. *card-order-on A r*
proof−
  obtain *R* **where** *R-def*: *R* = {*r*. *well-order-on A r*} **by** *blast*
  have *1*: *R* ≠ {} ∧ (∀ *r* ∈ *R*. *Well-order r*)
  using *well-order-on*[*of A*] *R-def rel.well-order-on-Well-order* **by** *blast*
  hence ∃ *r* ∈ *R*. ∀ *r'* ∈ *R*. *r* ≤*o* *r'*
  using *exists-minim-Well-order*[*of R*] **by** *auto*
  thus *?thesis* **using** *R-def* **unfolding** *card-order-on-def* **by** *auto*
qed

lemma *card-order-on-ordIso*:
assumes *CO*: *card-order-on A r* **and** *CO'*: *card-order-on A r'*
shows *r* =*o* *r'*
using *assms* **unfolding** *card-order-on-def*
using *ordIso-iff-ordLeq* **by** *blast*

lemma *Card-order-ordIso*:
assumes *CO*: *Card-order r* **and** *ISO*: *r'* =*o* *r*
shows *Card-order r'*

using *ISO* **unfolding** *ordIso-def*
proof(*unfold card-order-on-def*, *auto*)
  fix p' **assume** *well-order-on* (*Field* r') p'
  hence *0*: *Well-order* $p' \wedge$ *Field* p' = *Field* r'
  using *rel.well-order-on-Well-order* **by** *blast*
  obtain *f* **where** *1*: *iso* r' *r f* **and** *2*: *Well-order* $r \wedge$ *Well-order* r'
  using *ISO* **unfolding** *ordIso-def* **by** *auto*
  hence *3*: *inj-on f* (*Field* r') $\wedge$ *f* ‘ (*Field* r') = *Field r*
  by (*auto simp add*: *iso-iff embed-inj-on*)
  let *?p* = *dir-image* p' *f*
  have *4*: p' =*o ?p* $\wedge$ *Well-order ?p*
  using *0 2 3* **by** (*auto simp add*: *dir-image-ordIso Well-order-dir-image*)
  moreover have *Field ?p* = *Field r*
  using *0 3* **by** (*auto simp add*: *dir-image-Field2 order-on-defs*)
  ultimately have *well-order-on* (*Field r*) *?p* **by** *auto*
  hence $r \leq o$ *?p* **using** *CO* **unfolding** *card-order-on-def* **by** *auto*
  thus $r' \leq o\ p'$
  using *ISO 4 ordLeq-ordIso-trans ordIso-ordLeq-trans ordIso-symmetric* **by** *blast*
qed

lemma *Card-order-ordIso2*:
assumes *CO*: *Card-order r* **and** *ISO*: *r* =*o* r'
shows *Card-order* r'
using *assms Card-order-ordIso ordIso-symmetric* **by** *blast*

8.2 Cardinal of a set

We define the cardinal of set to be *some* cardinal order on that set. We shall prove that this notion is unique up to order isomorphism, meaning that order isomorphism shall be the true identity of cardinals.

definition *card-of* :: *'a set* ⇒ *'a rel* (|-|)
where *card-of A* = (*SOME r*. *card-order-on A r*)

lemma *card-of-card-order-on*[*simp*]: *card-order-on A* |*A*|
unfolding *card-of-def* **by** (*auto simp add*: *card-order-on someI-ex*)

lemma *card-of-well-order-on*[*simp*]: *well-order-on A* |*A*|
using *card-of-card-order-on card-order-on-def* **by** *blast*

lemma *Field-card-of*[*simp*]: *Field* |*A*| = *A*
using *card-of-card-order-on*[*of A*] **unfolding** *card-order-on-def*
using *rel.well-order-on-Field* **by** *blast*

lemma *card-of-Card-order*[*simp*]: *Card-order* |*A*|

by *auto*

corollary *ordIso-card-of-imp-Card-order*:
$r =o |A| \Longrightarrow$ *Card-order r*
using *card-of-Card-order Card-order-ordIso* **by** *blast*

lemma *card-of-Well-order*[*simp*]: *Well-order* $|A|$
using *card-of-Card-order* **unfolding** *card-order-on-def* **by** *auto*

lemma *card-of-refl*: $|A| =o |A|$
using *card-of-Well-order ordIso-reflexive* **by** *blast*

lemma *card-of-least*[*simp*]: *well-order-on A r* $\Longrightarrow |A| \leq o\ r$
using *card-of-card-order-on* **unfolding** *card-order-on-def* **by** *blast*

lemma *card-of-ordIso*:
$(\exists f.$ *bij-betw f A B*$) = (|A| =o |B|)$
proof(*auto*)
  fix *f* **assume** $*$: *bij-betw f A B*
  then obtain *r* **where** *well-order-on B r* $\wedge |A| =o\ r$
  using *Well-order-iso-copy card-of-well-order-on* **by** *blast*
  hence $|B| \leq o |A|$ **using** *card-of-least*
  ordLeq-ordIso-trans ordIso-symmetric **by** *blast*
  moreover
  {let *?g* = *inv-into A f*
   have *bij-betw ?g B A* **using** $*$ *bij-betw-inv-into* **by** *blast*
   then obtain *r* **where** *well-order-on A r* $\wedge |B| =o\ r$
   using *Well-order-iso-copy card-of-well-order-on* **by** *blast*
   hence $|A| \leq o |B|$ **using** *card-of-least*
   ordLeq-ordIso-trans ordIso-symmetric **by** *blast*
  }
  ultimately show $|A| =o |B|$ **using** *ordIso-iff-ordLeq* **by** *blast*
next
  assume $|A| =o |B|$
  then obtain *f* **where** *iso* $(|A|)\ (|B|)$ *f*
  unfolding *ordIso-def* **by** *auto*
  hence *bij-betw f A B* **unfolding** *iso-def Field-card-of* **by** *simp*
  thus $\exists f.$ *bij-betw f A B* **by** *auto*
qed

lemma *card-of-ordLeq*:
$(\exists f.$ *inj-on f A* $\wedge f$ ‘ $A \leq B) = (|A| \leq o |B|)$
proof(*auto*)

fix f **assume** $*$: *inj-on* f A **and** $**$: f ' $A \leq B$
{assume $|B| <o\ |A|$
hence $|B| \leq o\ |A|$ **using** *ordLeq-iff-ordLess-or-ordIso* **by** *blast*
then obtain g **where** *embed* ($|B|$) ($|A|$) g
unfolding *ordLeq-def* **by** *auto*
hence *1*: *inj-on* g $B \wedge g$ ' $B \leq A$ **using** *embed-inj-on*[*of* $|B|$ $|A|$ g]
card-of-Well-order[*of* B] *Field-card-of*[*of* B] *Field-card-of*[*of* A]
embed-Field[*of* $|B|$ $|A|$ g] **by** *auto*
obtain h **where** *bij-betw* h A B
using $*$ $**$ *1* *Cantor-Bernstein*[*of* f] **by** *fastforce*
hence $|A| =o\ |B|$ **using** *card-of-ordIso* **by** *blast*
hence $|A| \leq o\ |B|$ **using** *ordIso-iff-ordLeq* **by** *auto*
}
thus $|A| \leq o\ |B|$ **using** *ordLess-or-ordLeq*[*of* $|B|$ $|A|$] **by** *auto*
next
assume $*$: $|A| \leq o\ |B|$
obtain f **where** *embed* ($|A|$) ($|B|$) f
using $*$ **unfolding** *ordLeq-def* **by** *auto*
hence *inj-on* f $A \wedge f$ ' $A \leq B$ **using** *embed-inj-on*[*of* $|A|$ $|B|$ f]
card-of-Well-order[*of* A] *Field-card-of*[*of* A] *Field-card-of*[*of* B]
embed-Field[*of* $|A|$ $|B|$ f] **by** *auto*
thus $\exists f.$ *inj-on* f $A \wedge f$ ' $A \leq B$ **by** *auto*
qed

lemma *card-of-ordLeq2*:
$A \neq \{\} \Longrightarrow (\exists g.\ g$ ' $B = A) = (|A| \leq o\ |B|)$
using *card-of-ordLeq*[*of* A B] *inj-on-iff-surjective*[*of* A B] **by** *auto*

lemma *card-of-inj-rel*: **assumes** *INJ*: !! x y y'. $[\![(x,y) : R;\ (x,y') : R]\!] \Longrightarrow y = y'$
shows $|\{y.\ EX\ x.\ (x,y) : R\}| <=o\ |\{x.\ EX\ y.\ (x,y) : R\}|$
proof−
let $?Y = \{y.\ EX\ x.\ (x,y) : R\}$ **let** $?X = \{x.\ EX\ y.\ (x,y) : R\}$
let $?f = \%\ y.\ SOME\ x.\ (x,y) : R$
have $?f$ ' $?Y <= ?X$ **using** *someI* **by** *force*
moreover have *inj-on* $?f$ $?Y$
unfolding *inj-on-def* **proof**(*auto*)
fix *y1 x1 y2 x2*
assume $*$: $(x1,\ y1) \in R$ $(x2,\ y2) \in R$ **and** $**$: $?f\ y1 = ?f\ y2$
hence $(?f\ y1,y1) : R$ **using** *someI*[*of* $\%\ x.\ (x,y1) : R$] **by** *auto*
moreover have $(?f\ y2,y2) : R$ **using** $*$ *someI*[*of* $\%\ x.\ (x,y2) : R$] **by** *auto*
ultimately show $y1 = y2$ **using** $**$ *INJ* **by** *auto*
qed
ultimately show $|?Y| <=o\ |?X|$ **using** *card-of-ordLeq* **by** *blast*
qed

lemma *card-of-ordLess*:

$(\neg(\exists f.\ \textit{inj-on}\ f\ A \wedge f\ `\ A \leq B)) = (\ |B| <o\ |A|\ )$
proof−
  have $(\neg(\exists f.\ \textit{inj-on}\ f\ A \wedge f\ `\ A \leq B)) = (\neg\ |A| \leq o\ |B|\ )$
  using *card-of-ordLeq* **by** *blast*
  also have $\ldots = (\ |B| <o\ |A|\ )$
  using *card-of-Well-order*[*of A*] *card-of-Well-order*[*of B*]
      not-ordLeq-iff-ordLess **by** *blast*
  finally show *?thesis* **.**
qed

lemma *card-of-ordLess2*:
$B \neq \{\} \Longrightarrow (\neg(\exists f.\ f\ `\ A = B)) = (\ |A| <o\ |B|\ )$
using *card-of-ordLess*[*of B A*] *inj-on-iff-surjective*[*of B A*] **by** *auto*

lemma *card-of-unique*[*simp*]:
$\textit{card-order-on}\ A\ r \Longrightarrow r =o\ |A|$
by (*auto simp add*: *card-order-on-ordIso*)

lemma *card-of-unique2*: $[\![\textit{card-order-on}\ B\ r;\ \textit{bij-betw}\ f\ A\ B]\!] \Longrightarrow r =o\ |A|$
using *card-of-ordIso card-of-unique ordIso-equivalence* **by** *blast*

lemma *card-of-mono1*[*simp*]:
$A \leq B \Longrightarrow |A| \leq o\ |B|$
using *inj-on-id*[*of A*] *card-of-ordLeq*[*of A B*] **by** *fastforce*

lemma *card-of-mono2*[*simp*]:
assumes $r \leq o\ r'$
shows $|\textit{Field}\ r| \leq o\ |\textit{Field}\ r'|$
proof−
  obtain f **where**
  1: $\textit{well-order-on}\ (\textit{Field}\ r)\ r \wedge \textit{well-order-on}\ (\textit{Field}\ r)\ r \wedge \textit{embed}\ r\ r'\ f$
  using *assms* **unfolding** *ordLeq-def*
  by (*auto simp add*: *rel.well-order-on-Well-order*)
  hence $\textit{inj-on}\ f\ (\textit{Field}\ r) \wedge f\ `\ (\textit{Field}\ r) \leq \textit{Field}\ r'$
  by (*auto simp add*: *embed-inj-on embed-Field*)
  thus $|\textit{Field}\ r| \leq o\ |\textit{Field}\ r'|$ **using** *card-of-ordLeq* **by** *blast*
qed

lemma *card-of-cong*[*simp*]: $r =o\ r' \Longrightarrow |\textit{Field}\ r| =o\ |\textit{Field}\ r'|$
by (*auto simp add*: *ordIso-iff-ordLeq*)

lemma *card-of-Field-ordLess*[*simp*]: $\textit{Well-order}\ r \Longrightarrow |\textit{Field}\ r| \leq o\ r$

using *card-of-least card-of-well-order-on rel.well-order-on-Well-order* **by** *blast*

lemma *card-of-Field-ordIso*[*simp*]:
assumes *Card-order r*
shows $|Field\ r| =o\ r$
proof−
  have *card-order-on* (*Field r*) *r*
  using *assms card-order-on-Card-order* **by** *blast*
  moreover have *card-order-on* (*Field r*) $|Field\ r|$
  using *card-of-card-order-on* **by** *blast*
  ultimately show *?thesis* **using** *card-order-on-ordIso* **by** *blast*
qed

lemma *Card-order-iff-ordIso-card-of*:
Card-order r = ($r =o\ |Field\ r|$)
using *ordIso-card-of-imp-Card-order card-of-Field-ordIso ordIso-symmetric* **by** *blast*

lemma *Card-order-iff-ordLeq-card-of*:
Card-order r = ($r \leq o\ |Field\ r|$)
proof−
  have *Card-order r* = ($r =o\ |Field\ r|$)
  unfolding *Card-order-iff-ordIso-card-of* **by** *simp*
  also have ... = ($r \leq o\ |Field\ r| \wedge |Field\ r| \leq o\ r$)
  unfolding *ordIso-iff-ordLeq* **by** *simp*
  also have ... = ($r \leq o\ |Field\ r|$)
  using *card-of-Field-ordLess* **by** *auto*
  finally show *?thesis* **.**
qed

lemma *Card-order-iff-Restr-underS*:
assumes *Well-order r*
shows *Card-order r* = ($\forall\ a \in Field\ r.\ Restr\ r\ (underS\ r\ a) <o\ |Field\ r|$)
using *assms* **unfolding** *Card-order-iff-ordLeq-card-of*
using *ordLeq-iff-ordLess-Restr card-of-Well-order* **by** *blast*

lemma *card-of-underS*[*simp*]:
assumes *r*: *Card-order r* **and** *a*: *a* : *Field r*
shows $|underS\ r\ a| <o\ r$
proof−
  let *?A* = *underS r a* **let** *?r′* = *Restr r ?A*
  have *1*: *Well-order r*
  using *r* **unfolding** *card-order-on-def* **by** *simp*
  have *Well-order ?r′* **using** *1 Well-order-Restr* **by** *auto*
  moreover have *card-order-on* (*Field ?r′*) $|Field\ ?r'|$

using *card-of-card-order-on* **.**
ultimately have $|Field\ ?r'| \le o\ ?r'$
unfolding *card-order-on-def* **by** *simp*
moreover have $Field\ ?r' = ?A$
using *1 wo-rel.underS-ofilter Field-Restr-ofilter*
unfolding *wo-rel-def* **by** *fastforce*
ultimately have $|?A| \le o\ ?r'$ **by** *simp*
also have $?r' <o\ |Field\ r|$
using *1 a r Card-order-iff-Restr-underS* **by** *blast*
also have $|Field\ r| =o\ r$
using *r ordIso-symmetric* **unfolding** *Card-order-iff-ordIso-card-of* **by** *auto*
finally show *?thesis* **.**
qed

lemma *ordLess-Field*[*simp*]:
assumes $r <o\ r'$
shows $|Field\ r| <o\ r'$
proof−
have *well-order-on* (*Field r*) *r* **using** *assms* **unfolding** *ordLess-def*
by (*auto simp add*: *rel.well-order-on-Well-order*)
hence $|Field\ r| \le o\ r$ **using** *card-of-least* **by** *blast*
thus *?thesis* **using** *assms ordLeq-ordLess-trans* **by** *blast*
qed

lemma *internalize-card-of-ordLess*:
$(|A| <o\ r) = (\exists B < Field\ r.\ |A| =o\ |B| \wedge |B| <o\ r)$
proof
assume $|A| <o\ r$
then obtain *p* **where** *1*: $Field\ p < Field\ r \wedge |A| =o\ p \wedge p <o\ r$
using *internalize-ordLess*[*of* $|A|$ *r*] **by** *blast*
hence *Card-order p* **using** *card-of-Card-order Card-order-ordIso2* **by** *blast*
hence $|Field\ p| =o\ p$ **using** *card-of-Field-ordIso* **by** *blast*
hence $|A| =o\ |Field\ p| \wedge |Field\ p| <o\ r$
using *1 ordIso-equivalence ordIso-ordLess-trans* **by** *blast*
thus $\exists B < Field\ r.\ |A| =o\ |B| \wedge |B| <o\ r$ **using** *1* **by** *blast*
next
assume $\exists B < Field\ r.\ |A| =o\ |B| \wedge |B| <o\ r$
thus $|A| <o\ r$ **using** *ordIso-ordLess-trans* **by** *blast*
qed

lemma *internalize-card-of-ordLess2*:
$(|A| <o\ |C|) = (\exists B < C.\ |A| =o\ |B| \wedge |B| <o\ |C|)$
using *internalize-card-of-ordLess*[*of A* $|C|$] *Field-card-of*[*of C*] **by** *auto*

lemma *internalize-card-of-ordLeq*:

$(|A| \leq o\ r) = (\exists B \leq Field\ r.\ |A| =o\ |B| \wedge |B| \leq o\ r)$
proof
  assume $|A| \leq o\ r$
  then obtain p **where** *1*: $Field\ p \leq Field\ r \wedge |A| =o\ p \wedge p \leq o\ r$
  using *internalize-ordLeq*[*of* $|A|$ r] **by** *blast*
  hence *Card-order p* **using** *card-of-Card-order Card-order-ordIso2* **by** *blast*
  hence $|Field\ p| =o\ p$ **using** *card-of-Field-ordIso* **by** *blast*
  hence $|A| =o\ |Field\ p| \wedge |Field\ p| \leq o\ r$
  using *1 ordIso-equivalence ordIso-ordLeq-trans* **by** *blast*
  thus $\exists B \leq Field\ r.\ |A| =o\ |B| \wedge |B| \leq o\ r$ **using** *1* **by** *blast*
next
  assume $\exists B \leq Field\ r.\ |A| =o\ |B| \wedge |B| \leq o\ r$
  thus $|A| \leq o\ r$ **using** *ordIso-ordLeq-trans* **by** *blast*
qed

lemma *internalize-card-of-ordLeq2*:
$(|A| \leq o\ |C|) = (\exists B \leq C.\ |A| =o\ |B| \wedge |B| \leq o\ |C|)$
using *internalize-card-of-ordLeq*[*of A* $|C|$] *Field-card-of*[*of C*] **by** *auto*

lemma *Card-order-omax*:
assumes *finite R* **and** $R \neq \{\}$ **and** $\forall r \in R.$ *Card-order r*
shows *Card-order* (*omax R*)
proof−
  have $\forall r \in R.$ *Well-order r*
  using *assms* **unfolding** *card-order-on-def* **by** *simp*
  thus *?thesis* **using** *assms* **apply** − **apply**(*drule omax-in*) **by** *auto*
qed

lemma *Card-order-omax2*:
assumes *finite I* **and** $I \neq \{\}$
shows *Card-order* (*omax* $\{|A\ i| \mid i.\ i \in I\}$)
proof−
  let $?R = \{|A\ i| \mid i.\ i \in I\}$
  have *finite ?R* **and** $?R \neq \{\}$ **using** *assms* **by** *auto*
  moreover have $\forall r \in ?R.$ *Card-order r*
  using *card-of-Card-order* **by** *auto*
  ultimately show *?thesis* **by**(*rule Card-order-omax*)
qed

8.3 Cardinals versus set operations on arbitrary sets

Here we embark in a long journey of simple results showing that the standard set-theoretic operations are well-behaved w.r.t. the notion of cardinal – essentially, this means that they preserve the "cardinal identity" $=o$ and are monotonic w.r.t. $\leq o$.

lemma *subset-ordLeq-strict*:
assumes $A \leq B$ **and** $|A| <o\ |B|$
shows $A < B$
proof−
  {**assume** $\neg(A < B)$
   hence $A = B$ **using** *assms*(*1*) **by** *blast*
   hence *False* **using** *assms*(*2*) *not-ordLess-ordIso card-of-refl* **by** *blast*
  }
  thus *?thesis* **by** *blast*
qed

corollary *subset-ordLeq-diff*:
assumes $A \leq B$ **and** $|A| <o\ |B|$
shows $B - A \neq \{\}$
using *assms subset-ordLeq-strict* **by** *blast*

lemma *card-of-empty*[*simp*]: $|\{\}| \leq o\ |A|$
using *card-of-ordLeq inj-on-id* **by** *blast*

lemma *card-of-empty1*[*simp*]:
assumes *Well-order r* $\vee$ *Card-order r*
shows $|\{\}| \leq o\ r$
proof−
  have *Well-order r* **using** *assms* **unfolding** *card-order-on-def* **by** *auto*
  hence $|\mathit{Field}\ r| <=o\ r$
  using *assms card-of-Field-ordLess* **by** *blast*
  moreover have $|\{\}| \leq o\ |\mathit{Field}\ r|$ **by** *simp*
  ultimately show *?thesis* **using** *ordLeq-transitive* **by** *blast*
qed

corollary *Card-order-empty*:
Card-order r $\Longrightarrow |\{\}| \leq o\ r$ **by** *simp*

lemma *card-of-empty2*:
assumes *LEQ*: $|A| =o\ |\{\}|$
shows $A = \{\}$
using *assms card-of-ordIso*[*of A*] *bij-betw-empty2* **by** *blast*

lemma *card-of-empty3*:
assumes *LEQ*: $|A| \leq o\ |\{\}|$
shows $A = \{\}$
using *assms* **by** (*auto simp add*: *ordIso-iff-ordLeq card-of-empty2*)

lemma *card-of-empty4*:
$|\{\}::'b\ set| <o\ |A::'a\ set| = (A \neq \{\})$
proof(*intro iffI notI*)
  assume $*$: $|\{\}::'b\ set| <o\ |A|$ **and** $A = \{\}$
  hence $|A| =o\ |\{\}::'b\ set|$
  using *card-of-ordIso* **unfolding** *bij-betw-def inj-on-def* **by** *blast*
  hence $|\{\}::'b\ set| =o\ |A|$ **using** *ordIso-symmetric* **by** *blast*
  with $*$ **show** *False* **using** *not-ordLess-ordIso*[*of* $|\{\}::'b\ set|$ $|A|$] **by** *blast*
next
  assume $A \neq \{\}$
  hence $(\neg\ (\exists f.\ inj\text{-}on\ f\ A \wedge f\ `\ A \subseteq \{\}))$
  unfolding *inj-on-def* **by** *blast*
  thus $|\ \{\}\ | <o\ |\ A\ |$
  using *card-of-ordLess* **by** *blast*
qed

lemma *card-of-empty5*:
$|A| <o\ |B| \Longrightarrow B \neq \{\}$
using *card-of-empty not-ordLess-ordLeq* **by** *blast*

lemma *Well-order-card-of-empty*:
Well-order $r \Longrightarrow |\{\}| \leq o\ r$ **by** *simp*

lemma *card-of-empty-ordIso*:
$|\{\}::'a\ set| =o\ |\{\}::'b\ set|$
using *card-of-ordIso* **unfolding** *bij-betw-def inj-on-def* **by** *blast*

lemma *card-of-image*[*simp*]:
$|f\ `\ A| <=o\ |A|$
proof(*cases* $A = \{\}$, *simp*)
  assume $A \sim= \{\}$
  hence $f\ `\ A \sim= \{\}$ **by** *auto*
  thus $|f\ `\ A| \leq o\ |A|$
  using *card-of-ordLeq2*[*of* $f\ `\ A\ A$] **by** *auto*
qed

lemma *surj-imp-ordLeq*:
assumes $B <= f\ `\ A$
shows $|B| <=o\ |A|$
proof−
  have $|B| <=o\ |f\ `\ A|$ **using** *assms card-of-mono1* **by** *auto*
  thus *?thesis* **using** *card-of-image ordLeq-transitive* **by** *blast*

qed

lemma *card-of-UNIV*[*simp*]:
$|A :: 'a\ set| \leq o\ |UNIV :: 'a\ set|$
using *card-of-mono1*[*of A*] **by** *simp*

lemma *card-of-UNIV2*[*simp*]:
Card-order r $\Longrightarrow$ $(r :: 'a\ rel) \leq o\ |UNIV :: 'a\ set|$
using *card-of-UNIV*[*of Field r*] *card-of-Field-ordIso*
ordIso-symmetric ordIso-ordLeq-trans **by** *blast*

lemma *card-of-singl-ordLeq*[*simp*]:
assumes $A \neq \{\}$
shows $|\{b\}| \leq o\ |A|$
proof−
obtain *a* **where** $*$: $a \in A$ **using** *assms* **by** *auto*
let $?h = \lambda\ b'::'b.\ if\ b' = b\ then\ a\ else\ undefined$
have *inj-on* $?h\ \{b\} \wedge ?h\ `\ \{b\} \leq A$
using $*$ **unfolding** *inj-on-def* **by** *auto*
thus *?thesis* **using** *card-of-ordLeq* **by** *blast*
qed

corollary *Card-order-singl-ordLeq*[*simp*]:
$[\![$*Card-order r*; *Field r* $\neq \{\}]\!] \Longrightarrow |\{b\}| \leq o\ r$
using *card-of-singl-ordLeq*[*of Field r b*]
card-of-Field-ordIso[*of r*] *ordLeq-ordIso-trans* **by** *blast*

lemma *card-of-Pow*[*simp*]: $|A| <o\ |Pow\ A|$
using *card-of-ordLess2*[*of Pow A A*] *Cantors-paradox*[*of A*]
Pow-not-empty[*of A*] **by** *auto*

lemma *infinite-Pow*:
assumes *infinite A*
shows *infinite* (*Pow A*)
proof−
have $|A| \leq o\ |Pow\ A|$ **by** (*metis card-of-Pow ordLess-imp-ordLeq*)
thus *?thesis* **by** (*metis assms finite-Pow-iff*)
qed

corollary *Card-order-Pow*[*simp*]:
Card-order r $\Longrightarrow r <o\ |Pow(Field\ r)|$
using *card-of-Pow card-of-Field-ordIso ordIso-ordLess-trans ordIso-symmetric* **by**

blast

corollary *card-of-set-type*[*simp*]: $|UNIV::'a\ set| <o\ |UNIV::'a\ set\ set|$
using *card-of-Pow*[*of UNIV*::'*a set*] **by** *simp*

lemma *card-of-Pow-mono*[*simp*]:
assumes $|A| \leq o\ |B|$
shows $|Pow\ A| \leq o\ |Pow\ B|$
proof−
  obtain *f* **where** $inj\text{-}on\ f\ A \wedge f\ `\ A \leq B$
  using *assms card-of-ordLeq*[*of A B*] **by** *auto*
  hence $inj\text{-}on\ (image\ f)\ (Pow\ A) \wedge (image\ f)\ `\ (Pow\ A) \leq (Pow\ B)$
  by (*auto simp add*: *inj-on-image-Pow image-Pow-mono*)
  thus *?thesis* **using** *card-of-ordLeq*[*of Pow A*] **by** *auto*
qed

lemma *ordIso-Pow-mono*[*simp*]:
assumes $r \leq o\ r'$
shows $|Pow(Field\ r)| \leq o\ |Pow(Field\ r')|$
using *assms card-of-mono2 card-of-Pow-mono* **by** *blast*

lemma *card-of-Pow-cong*[*simp*]:
assumes $|A| =o\ |B|$
shows $|Pow\ A| =o\ |Pow\ B|$
proof−
  obtain *f* **where** *bij-betw f A B*
  using *assms card-of-ordIso*[*of A B*] **by** *auto*
  hence *bij-betw* (*image f*) (*Pow A*) (*Pow B*)
  by (*auto simp add*: *bij-betw-image-Pow*)
  thus *?thesis* **using** *card-of-ordIso*[*of Pow A*] **by** *auto*
qed

lemma *ordIso-Pow-cong*[*simp*]:
assumes $r =o\ r'$
shows $|Pow(Field\ r)| =o\ |Pow(Field\ r')|$
using *assms card-of-cong card-of-Pow-cong* **by** *blast*

lemma *card-of-Plus1*[*simp*]: $|A| \leq o\ |A <+> B|$
proof−
  have $Inl\ `\ A \leq A <+> B$ **by** *auto*
  thus *?thesis* **using** *inj-Inl*[*of A*] *card-of-ordLeq* **by** *blast*
qed

corollary *Card-order-Plus1* [*simp*]:
Card-order r $\Longrightarrow$ *r* $\leq o$ |(*Field r*) <+> *B*|
using *card-of-Plus1 card-of-Field-ordIso ordIso-ordLeq-trans ordIso-symmetric* **by** *blast*

lemma *card-of-Plus2* [*simp*]: |*B*| $\leq o$ |*A* <+> *B*|
proof −
  have *Inr* ' *B* $\leq$ *A* <+> *B* **by** *auto*
  thus *?thesis* **using** *inj-Inr*[*of B*] *card-of-ordLeq* **by** *blast*
qed

corollary *Card-order-Plus2* [*simp*]:
Card-order r $\Longrightarrow$ *r* $\leq o$ |*A* <+> (*Field r*)|
using *card-of-Plus2 card-of-Field-ordIso ordIso-ordLeq-trans ordIso-symmetric* **by** *blast*

lemma *card-of-Plus-empty1*: |*A*| =*o* |*A* <+> {}|
proof −
  have *bij-betw Inl A* (*A* <+> {}) **unfolding** *bij-betw-def inj-on-def* **by** *auto*
  thus *?thesis* **using** *card-of-ordIso* **by** *auto*
qed

corollary *Card-order-Plus-empty1*:
Card-order r $\Longrightarrow$ *r* =*o* |(*Field r*) <+> {}|
using *card-of-Plus-empty1 card-of-Field-ordIso ordIso-equivalence* **by** *blast*

lemma *card-of-Plus-empty2*: |*A*| =*o* |{} <+> *A*|
proof −
  have *bij-betw Inr A* ({} <+> *A*) **unfolding** *bij-betw-def inj-on-def* **by** *auto*
  thus *?thesis* **using** *card-of-ordIso* **by** *auto*
qed

corollary *Card-order-Plus-empty2*:
Card-order r $\Longrightarrow$ *r* =*o* |{} <+> (*Field r*)|
using *card-of-Plus-empty2 card-of-Field-ordIso ordIso-equivalence* **by** *blast*

lemma *card-of-Plus-commute*: |*A* <+> *B*| =*o* |*B* <+> *A*|
proof −
  let *?f* = λ(*c*::'*a* + '*b*). *case c of Inl a* $\Rightarrow$ *Inr a*
                                    | *Inr b* $\Rightarrow$ *Inl b*
  have *bij-betw ?f* (*A* <+> *B*) (*B* <+> *A*)

unfolding *bij-betw-def inj-on-def* **by** *force*
thus *?thesis* **using** *card-of-ordIso* **by** *blast*
qed

lemma *card-of-Plus-assoc*:
fixes $A :: 'a\ set$ **and** $B :: 'b\ set$ **and** $C :: 'c\ set$
shows $|(A <+> B) <+> C| =o |A <+> B <+> C|$
proof −
def $f \equiv \lambda(k::('a + 'b) + 'c).$
case k of Inl ab $\Rightarrow$ (*case ab of Inl a* $\Rightarrow$ *Inl a*
$|$*Inr b* $\Rightarrow$ *Inr* (*Inl b*))
$|$*Inr c* $\Rightarrow$ *Inr* (*Inr c*)
have $A <+> B <+> C \subseteq f\ `\ ((A <+> B) <+> C)$
proof
fix x **assume** x: $x \in A <+> B <+> C$
show $x \in f\ `\ ((A <+> B) <+> C)$
proof(*cases x*)
case (*Inl a*)
hence $a \in A$ $x = f$ (*Inl* (*Inl a*))
using x **unfolding** *f-def* **by** *auto*
thus *?thesis* **by** *auto*
next
case (*Inr bc*) **note** *1* = *Inr* **show** *?thesis*
proof(*cases bc*)
case (*Inl b*)
hence $b \in B$ $x = f$ (*Inl* (*Inr b*))
using *x 1* **unfolding** *f-def* **by** *auto*
thus *?thesis* **by** *auto*
next
case (*Inr c*)
hence $c \in C$ $x = f$ (*Inr c*)
using *x 1* **unfolding** *f-def* **by** *auto*
thus *?thesis* **by** *auto*
qed
qed
qed
hence *bij-betw f* $((A <+> B) <+> C)$ $(A <+> B <+> C)$
unfolding *bij-betw-def inj-on-def f-def* **by** *auto*
thus *?thesis* **using** *card-of-ordIso* **by** *blast*
qed

lemma *card-of-Plus-mono1*[*simp*]:
assumes $|A| \leq o\ |B|$
shows $|A <+> C| \leq o\ |B <+> C|$
proof−
obtain f **where** *1*: *inj-on f A* $\wedge$ $f\ `\ A \leq B$
using *assms card-of-ordLeq*[*of A*] **by** *fastforce*

obtain *g* **where** *g-def*:
$g = (\lambda d.\ case\ d\ of\ Inl\ a \Rightarrow Inl(f\ a) \mid Inr\ (c::'c) \Rightarrow Inr\ c)$ **by** *blast*
have $inj\text{-}on\ g\ (A <+> C) \wedge g\ `\ (A <+> C) \leq (B <+> C)$
proof−
{**fix** *d1* **and** *d2* **assume** $d1 \in A <+> C \wedge d2 \in A <+> C$ **and**
$g\ d1 = g\ d2$
hence $d1 = d2$ **using** *1* **unfolding** *inj-on-def*
by(*case-tac d1*, *case-tac d2*, *auto simp add*: *g-def*)
}
moreover
{**fix** *d* **assume** $d \in A <+> C$
hence $g\ d \in B <+> C$ **using** *1*
by(*case-tac d*, *auto simp add*: *g-def*)
}
ultimately show *?thesis* **unfolding** *inj-on-def* **by** *auto*
qed
thus *?thesis* **using** *card-of-ordLeq* **by** *auto*
qed

corollary *ordLeq-Plus-mono1*:
assumes $r \leq o\ r'$
shows $|(Field\ r) <+> C| \leq o\ |(Field\ r') <+> C|$
using *assms card-of-mono2 card-of-Plus-mono1* **by** *blast*

lemma *card-of-Plus-mono2*[*simp*]:
assumes $|A| \leq o\ |B|$
shows $|C <+> A| \leq o\ |C <+> B|$
using *assms card-of-Plus-mono1*[*of A B C*]
card-of-Plus-commute[*of C A*] *card-of-Plus-commute*[*of B C*]
ordIso-ordLeq-trans[*of* $|C <+> A|$] *ordLeq-ordIso-trans*[*of* $|C <+> A|$]
by *blast*

corollary *ordLeq-Plus-mono2*:
assumes $r \leq o\ r'$
shows $|A <+> (Field\ r)| \leq o\ |A <+> (Field\ r')|$
using *assms card-of-mono2 card-of-Plus-mono2* **by** *blast*

lemma *card-of-Plus-mono*[*simp*]:
assumes $|A| \leq o\ |B|$ **and** $|C| \leq o\ |D|$
shows $|A <+> C| \leq o\ |B <+> D|$
using *assms card-of-Plus-mono1*[*of A B C*] *card-of-Plus-mono2*[*of C D B*]
ordLeq-transitive[*of* $|A <+> C|$] **by** *blast*

corollary *ordLeq-Plus-mono*:

assumes $r \leq o\ r'$ **and** $p \leq o\ p'$
shows $|(Field\ r) <+> (Field\ p)| \leq o\ |(Field\ r') <+> (Field\ p')|$
using *assms card-of-mono2[of r r′] card-of-mono2[of p p′] card-of-Plus-mono* **by** *blast*

lemma *card-of-Plus-cong1*:
assumes $|A| =o\ |B|$
shows $|A <+> C| =o\ |B <+> C|$
using *assms*
by (*auto simp add*: *ordIso-iff-ordLeq*)

corollary *ordIso-Plus-cong1*:
assumes $r =o\ r'$
shows $|(Field\ r) <+> C| =o\ |(Field\ r') <+> C|$
using *assms card-of-cong card-of-Plus-cong1* **by** *blast*

lemma *card-of-Plus-cong2[simp]*:
assumes $|A| =o\ |B|$
shows $|C <+> A| =o\ |C <+> B|$
using *assms*
by (*auto simp add*: *ordIso-iff-ordLeq*)

corollary *ordIso-Plus-cong2*:
assumes $r =o\ r'$
shows $|A <+> (Field\ r)| =o\ |A <+> (Field\ r')|$
using *assms card-of-cong card-of-Plus-cong2* **by** *blast*

lemma *card-of-Plus-cong[simp]*:
assumes $|A| =o\ |B|$ **and** $|C| =o\ |D|$
shows $|A <+> C| =o\ |B <+> D|$
using *assms*
by (*auto simp add*: *ordIso-iff-ordLeq*)

corollary *ordIso-Plus-cong*:
assumes $r =o\ r'$ **and** $p =o\ p'$
shows $|(Field\ r) <+> (Field\ p)| =o\ |(Field\ r') <+> (Field\ p')|$
using *assms card-of-cong[of r r′] card-of-cong[of p p′] card-of-Plus-cong* **by** *blast*

lemma *card-of-Un1[simp]*:
shows $|A| \leq o\ |A \cup B|$
using *inj-on-id[of A] card-of-ordLeq[of A -]* **by** *fastforce*

lemma *Card-order-Un1*:
shows *Card-order r* $\Longrightarrow$ $|Field\ r| \leq o\ |(Field\ r) \cup B|$
using *card-of-Un1 card-of-Field-ordIso ordIso-symmetric ordIso-ordLeq-trans* **by** *auto*

lemma *card-of-Un2*[*simp*]:
shows $|A| \leq o\ |B \cup A|$
using *inj-on-id*[*of A*] *card-of-ordLeq*[*of A -*] **by** *fastforce*

lemma *Card-order-Un2*:
shows *Card-order r* $\Longrightarrow$ $|Field\ r| \leq o\ |A \cup (Field\ r)|$
using *card-of-Un2 card-of-Field-ordIso ordIso-symmetric ordIso-ordLeq-trans* **by** *auto*

lemma *card-of-diff*[*simp*]:
shows $|A - B| \leq o\ |A|$
using *inj-on-id*[*of A* $-$ *B*] *card-of-ordLeq*[*of A* $-$ *B -*] **by** *fastforce*

lemma *Un-Plus-bij-betw*:
assumes *A Int B* = {}
shows $\exists f.$ *bij-betw f* $(A \cup B)$ $(A <+> B)$
proof$-$
  let *?f* = λ *c. if c* $\in$ *A then Inl c else Inr c*
  have *bij-betw ?f* $(A \cup B)$ $(A <+> B)$
  using *assms* **by**(*unfold bij-betw-def inj-on-def*, *auto*)
  thus *?thesis* **by** *blast*
qed

lemma *card-of-Un-Plus-ordIso*:
assumes *A Int B* = {}
shows $|A \cup B| =o\ |A <+> B|$
using *assms card-of-ordIso*[*of A* $\cup$ *B*] *Un-Plus-bij-betw*[*of A B*] **by** *auto*

lemma *card-of-Un-Plus-ordIso1*:
$|A \cup B| =o\ |A <+> (B - A)|$
using *card-of-Un-Plus-ordIso*[*of A B* $-$ *A*] **by** *auto*

lemma *card-of-Un-Plus-ordIso2*:
$|A \cup B| =o\ |(A - B) <+> B|$
using *card-of-Un-Plus-ordIso*[*of A* $-$ *B B*] **by** *auto*

lemma *card-of-Un-Plus-ordLeq*[*simp*]:
$|A \cup B| \leq o\ |A <+> B|$
proof−
  let $?f = \lambda\ c.\ if\ c \in A\ then\ Inl\ c\ else\ Inr\ c$
  have $inj\text{-}on\ ?f\ (A \cup B) \wedge\ ?f\ `\ (A \cup B) \leq A <+> B$
  unfolding *inj-on-def* **by** *auto*
  thus *?thesis* **using** *card-of-ordLeq* **by** *blast*
qed

lemma *card-of-Times1*[*simp*]:
assumes $A \neq \{\}$
shows $|B| \leq o\ |B \times A|$
proof(*cases* $B = \{\}$, *simp*)
  assume $*$: $B \neq \{\}$
  have $fst\ `(B \times A) = B$ **unfolding** *image-def* **using** *assms* **by** *auto*
  thus *?thesis* **using** *inj-on-iff-surjective*[*of* $B\ B \times A$]
    card-of-ordLeq[*of* $B\ B \times A$] $*$ **by** *blast*
qed

corollary *Card-order-Times1*:
$[\![Card\text{-}order\ r;\ B \neq \{\}]\!] \Longrightarrow r \leq o\ |(Field\ r) \times B|$
using *card-of-Times1*[*of B*] *card-of-Field-ordIso*
  ordIso-ordLeq-trans ordIso-symmetric **by** *blast*

lemma *card-of-Times-singl1*: $|A| =o\ |A \times \{b\}|$
proof−
  have $bij\text{-}betw\ fst\ (A \times \{b\})\ A$ **unfolding** *bij-betw-def inj-on-def* **by** *force*
  thus *?thesis* **using** *card-of-ordIso ordIso-symmetric* **by** *blast*
qed

corollary *Card-order-Times-singl1*:
$Card\text{-}order\ r \Longrightarrow r =o\ |(Field\ r) \times \{b\}|$
using *card-of-Times-singl1*[*of - b*] *card-of-Field-ordIso ordIso-equivalence* **by** *blast*

lemma *card-of-Times-singl2*: $|A| =o\ |\{b\} \times A|$
proof−
  have $bij\text{-}betw\ snd\ (\{b\} \times A)\ A$ **unfolding** *bij-betw-def inj-on-def* **by** *force*
  thus *?thesis* **using** *card-of-ordIso ordIso-symmetric* **by** *blast*
qed

corollary *Card-order-Times-singl2*:

Card-order r $\Longrightarrow$ *r* =*o* |{*a*} × (*Field r*)|
using *card-of-Times-singl2*[*of - a*] *card-of-Field-ordIso ordIso-equivalence* **by** *blast*

lemma *card-of-Times-commute*: |*A* × *B*| =*o* |*B* × *A*|
proof−
  let *?f* = λ(*a*::′*a*,*b*::′*b*). (*b*,*a*)
  have *bij-betw ?f* (*A* × *B*) (*B* × *A*)
  unfolding *bij-betw-def inj-on-def* **by** *auto*
  thus *?thesis* **using** *card-of-ordIso* **by** *blast*
qed

lemma *card-of-Times-assoc*: |(*A* × *B*) × *C*| =*o* |*A* × *B* × *C*|
proof −
  let *?f* = λ((*a*,*b*),*c*). (*a*,(*b*,*c*))
  have *A* × *B* × *C* ⊆ *?f* ‘ ((*A* × *B*) × *C*)
  proof
    fix *x* **assume** *x* ∈ *A* × *B* × *C*
    then obtain *a b c* **where** ∗: *a* ∈ *A* *b* ∈ *B* *c* ∈ *C* *x* = (*a*, *b*, *c*) **by** *blast*
    let *?x* = ((*a*, *b*), *c*)
    from ∗ **have** *?x* ∈ (*A* × *B*) × *C* *x* = *?f ?x* **by** *auto*
    thus *x* ∈ *?f* ‘ ((*A* × *B*) × *C*) **by** *blast*
  qed
  hence *bij-betw ?f* ((*A* × *B*) × *C*) (*A* × *B* × *C*)
  unfolding *bij-betw-def inj-on-def* **by** *auto*
  thus *?thesis* **using** *card-of-ordIso* **by** *blast*
qed

lemma *card-of-Times2*[*simp*]:
assumes *A* ≠ {}   **shows** |*B*| ≤*o* |*A* × *B*|
using *assms card-of-Times1*[*of A B*] *card-of-Times-commute*[*of B A*]
      ordLeq-ordIso-trans **by** *blast*

corollary *Card-order-Times2*:
⟦*Card-order r*; *A* ≠ {}⟧ $\Longrightarrow$ *r* ≤*o* |*A* × (*Field r*)|
using *card-of-Times2*[*of A*] *card-of-Field-ordIso*
      ordIso-ordLeq-trans ordIso-symmetric **by** *blast*

lemma *card-of-Times3*[*simp*]: |*A*| ≤*o* |*A* × *A*|
using *card-of-Times1*[*of A*]
by(*cases A* = {}, *simp*, *blast*)

corollary *Card-order-Times3*:
⟦*Card-order r*⟧ $\Longrightarrow$ |*Field r*| ≤*o* |(*Field r*) × (*Field r*)|

using *card-of-Times3 card-of-Field-ordIso*
ordIso-ordLeq-trans ordIso-symmetric **by** *blast*

lemma *card-of-Plus-Times-bool*: $|A <+> A| =o |A \times (UNIV::bool\ set)|$
proof−
let $?f = \lambda c::'a + 'a.\ case\ c\ of\ Inl\ a \Rightarrow (a,True)$
$|Inr\ a \Rightarrow (a,False)$
have *bij-betw* $?f\ (A <+> A)\ (A \times (UNIV::bool\ set))$
proof−
{**fix** *c1* **and** *c2* **assume** $?f\ c1 = ?f\ c2$
hence $c1 = c2$
by(*case-tac c1*, *case-tac c2*, *auto*, *case-tac c2*, *auto*)
}
moreover
{**fix** *c* **assume** $c \in A <+> A$
hence $?f\ c \in A \times (UNIV::bool\ set)$
by(*case-tac c*, *auto*)
}
moreover
{**fix** *a bl* **assume** $*$: $(a,bl) \in A \times (UNIV::bool\ set)$
have $(a,bl) \in ?f\ `\ (A <+> A)$
proof(*cases bl*)
assume *bl* **hence** $?f(Inl\ a) = (a,bl)$ **by** *auto*
thus *?thesis* **using** $*$ **by** *force*
next
assume $\neg\ bl$ **hence** $?f(Inr\ a) = (a,bl)$ **by** *auto*
thus *?thesis* **using** $*$ **by** *force*
qed
}
ultimately show *?thesis* **unfolding** *bij-betw-def inj-on-def* **by** *auto*
qed
thus *?thesis* **using** *card-of-ordIso* **by** *blast*
qed

lemma *card-of-Times-mono1*[*simp*]:
assumes $|A| \leq o\ |B|$
shows $|A \times C| \leq o\ |B \times C|$
proof−
obtain *f* **where** *1*: $inj\text{-}on\ f\ A \wedge f\ `\ A \leq B$
using *assms card-of-ordLeq*[*of A*] **by** *fastforce*
obtain *g* **where** *g-def*:
$g = (\lambda(a,c::'c).\ (f\ a,c))$ **by** *blast*
have $inj\text{-}on\ g\ (A \times C) \wedge g\ `\ (A \times C) \leq (B \times C)$
using *1* **unfolding** *inj-on-def* **using** *g-def* **by** *auto*
thus *?thesis* **using** *card-of-ordLeq* **by** *auto*
qed

corollary *ordLeq-Times-mono1*:
assumes $r \le o\ r'$
shows $|(Field\ r) \times C| \le o\ |(Field\ r') \times C|$
using *assms card-of-mono2 card-of-Times-mono1* **by** *blast*

lemma *card-of-Times-mono2*[*simp*]:
assumes $|A| \le o\ |B|$
shows $|C \times A| \le o\ |C \times B|$
using *assms card-of-Times-mono1*[*of A B C*]
card-of-Times-commute[*of C A*] *card-of-Times-commute*[*of B C*]
ordIso-ordLeq-trans[*of* $|C \times A|$] *ordLeq-ordIso-trans*[*of* $|C \times A|$]
by *blast*

corollary *ordLeq-Times-mono2*:
assumes $r \le o\ r'$
shows $|A \times (Field\ r)| \le o\ |A \times (Field\ r')|$
using *assms card-of-mono2 card-of-Times-mono2* **by** *blast*

lemma *card-of-Times-mono*[*simp*]:
assumes $|A| \le o\ |B|$ **and** $|C| \le o\ |D|$
shows $|A \times C| \le o\ |B \times D|$
using *assms card-of-Times-mono1*[*of A B C*] *card-of-Times-mono2*[*of C D B*]
ordLeq-transitive[*of* $|A \times C|$] **by** *blast*

corollary *ordLeq-Times-mono*:
assumes $r \le o\ r'$ **and** $p \le o\ p'$
shows $|(Field\ r) \times (Field\ p)| \le o\ |(Field\ r') \times (Field\ p')|$
using *assms card-of-mono2*[*of r r'*] *card-of-mono2*[*of p p'*] *card-of-Times-mono* **by** *blast*

lemma *card-of-Times-cong1*[*simp*]:
assumes $|A| =o\ |B|$
shows $|A \times C| =o\ |B \times C|$
using *assms*
by (*auto simp add*: *ordIso-iff-ordLeq*)

corollary *ordIso-Times-cong1*:
assumes $r =o\ r'$
shows $|(Field\ r) \times C| =o\ |(Field\ r') \times C|$
using *assms card-of-cong card-of-Times-cong1* **by** *blast*

lemma *card-of-Times-cong2*[*simp*]:
assumes $|A| =o |B|$
shows $|C \times A| =o |C \times B|$
using *assms*
by (*auto simp add*: *ordIso-iff-ordLeq*)

corollary *ordIso-Times-cong2*:
assumes $r =o r'$
shows $|A \times (Field\ r)| =o |A \times (Field\ r')|$
using *assms card-of-cong card-of-Times-cong2* **by** *blast*

lemma *card-of-Times-cong*[*simp*]:
assumes $|A| =o |B|$ **and** $|C| =o |D|$
shows $|A \times C| =o |B \times D|$
using *assms*
by (*auto simp add*: *ordIso-iff-ordLeq*)

corollary *ordIso-Times-cong*:
assumes $r =o r'$ **and** $p =o p'$
shows $|(Field\ r) \times (Field\ p)| =o |(Field\ r') \times (Field\ p')|$
using *assms card-of-cong*[*of r r'*] *card-of-cong*[*of p p'*] *card-of-Times-cong* **by** *blast*

lemma *card-of-Sigma-mono1*:
assumes $\forall i \in I.\ |A\ i| \leq o\ |B\ i|$
shows $|SIGMA\ i : I.\ A\ i| \leq o\ |SIGMA\ i : I.\ B\ i|$
proof−
  have $\forall i.\ i \in I \longrightarrow (\exists f.\ inj\text{-}on\ f\ (A\ i) \wedge f\ `\ (A\ i) \leq B\ i)$
  using *assms* **by** (*auto simp add*: *card-of-ordLeq*)
  with *choice*[*of* $\lambda\ i\ f.\ i \in I \longrightarrow inj\text{-}on\ f\ (A\ i) \wedge f\ `\ (A\ i) \leq B\ i$]
  obtain *F* **where** *1*: $\forall i \in I.\ inj\text{-}on\ (F\ i)\ (A\ i) \wedge (F\ i)\ `\ (A\ i) \leq B\ i$ **by** *fastforce*
  obtain *g* **where** *g-def*: $g = (\lambda(i,a::'b).\ (i,F\ i\ a))$ **by** *blast*
  have $inj\text{-}on\ g\ (Sigma\ I\ A) \wedge g\ `\ (Sigma\ I\ A) \leq (Sigma\ I\ B)$
  using *1* **unfolding** *inj-on-def* **using** *g-def* **by** *force*
  thus *?thesis* **using** *card-of-ordLeq* **by** *auto*
qed

lemma *card-of-Sigma-mono2*:
assumes $inj\text{-}on\ f\ (I::'i\ set)$ **and** $f\ `\ I \leq (J::'j\ set)$
shows $|SIGMA\ i : I.\ (A::'j \Rightarrow 'a\ set)\ (f\ i)| \leq o\ |SIGMA\ j : J.\ A\ j|$
proof−
  let $?LEFT = SIGMA\ i : I.\ A\ (f\ i)$
  let $?RIGHT = SIGMA\ j : J.\ A\ j$
  obtain *u* **where** *u-def*: $u = (\lambda(i::'i,a::'a).\ (f\ i,a))$ **by** *blast*
  have $inj\text{-}on\ u\ ?LEFT \wedge u\ `?LEFT \leq ?RIGHT$

using *assms* **unfolding** *u-def inj-on-def* **by** *auto*
thus *?thesis* **using** *card-of-ordLeq* **by** *blast*
qed

lemma *card-of-Sigma-mono*:
assumes *INJ*: *inj-on f I* **and** *IM*: *f ' I* $\leq$ *J* **and**
LEQ: $\forall j \in J.\ |A\ j| \leq o\ |B\ j|$
shows $|SIGMA\ i : I.\ A\ (f\ i)| \leq o\ |SIGMA\ j : J.\ B\ j|$
proof−
have $\forall i \in I.\ |A(f\ i)| \leq o\ |B(f\ i)|$
using *IM LEQ* **by** *blast*
hence $|SIGMA\ i : I.\ A\ (f\ i)| \leq o\ |SIGMA\ i : I.\ B\ (f\ i)|$
using *card-of-Sigma-mono1*[*of I*] **by** *fastforce*
moreover have $|SIGMA\ i : I.\ B\ (f\ i)| \leq o\ |SIGMA\ j : J.\ B\ j|$
using *INJ IM card-of-Sigma-mono2* **by** *blast*
ultimately show *?thesis* **using** *ordLeq-transitive* **by** *blast*
qed

lemma *ordLeq-Sigma-mono1*:
assumes $\forall i \in I.\ p\ i \leq o\ r\ i$
shows $|SIGMA\ i : I.\ Field(p\ i)| \leq o\ |SIGMA\ i : I.\ Field(r\ i)|$
using *assms* **by** (*auto simp add*: *card-of-Sigma-mono1*)

lemma *ordLeq-Sigma-mono*:
assumes *inj-on f I* **and** *f ' I* $\leq$ *J* **and**
$\forall j \in J.\ p\ j \leq o\ r\ j$
shows $|SIGMA\ i : I.\ Field(p(f\ i))| \leq o\ |SIGMA\ j : J.\ Field(r\ j)|$
using *assms card-of-mono2 card-of-Sigma-mono*
[*of f I J* λ *i*. *Field*(*p i*) λ *j*. *Field*(*r j*)] **by** *blast*

lemma *card-of-Sigma-cong1*:
assumes $\forall i \in I.\ |A\ i| =o\ |B\ i|$
shows $|SIGMA\ i : I.\ A\ i| =o\ |SIGMA\ i : I.\ B\ i|$
using *assms* **by** (*auto simp add*: *card-of-Sigma-mono1 ordIso-iff-ordLeq*)

lemma *card-of-Sigma-cong2*:
assumes *bij-betw f* (*I*::'*i set*) (*J*::'*j set*)
shows $|SIGMA\ i : I.\ (A::'j \Rightarrow 'a\ set)\ (f\ i)| =o\ |SIGMA\ j : J.\ A\ j|$
proof−
let *?LEFT* = *SIGMA i* : *I*. *A* (*f i*)
let *?RIGHT* = *SIGMA j* : *J*. *A j*
obtain *u* **where** *u-def*: *u* = (λ(*i*::'*i*,*a*::'*a*). (*f i*,*a*)) **by** *blast*
have *bij-betw u ?LEFT ?RIGHT*
using *assms* **unfolding** *u-def bij-betw-def inj-on-def* **by** *auto*

thus *?thesis* **using** *card-of-ordIso* **by** *blast*
qed

lemma *card-of-Sigma-cong*:
assumes *BIJ*: *bij-betw f I J* **and**
ISO: $\forall j \in J.\ |A\ j| =o\ |B\ j|$
shows $|SIGMA\ i : I.\ A\ (f\ i)| =o\ |SIGMA\ j : J.\ B\ j|$
proof−
have $\forall i \in I.\ |A(f\ i)| =o\ |B(f\ i)|$
using *ISO BIJ* **unfolding** *bij-betw-def* **by** *blast*
hence $|SIGMA\ i : I.\ A\ (f\ i)| =o\ |SIGMA\ i : I.\ B\ (f\ i)|$
using *card-of-Sigma-cong1* **by** *fastforce*
moreover have $|SIGMA\ i : I.\ B\ (f\ i)| =o\ |SIGMA\ j : J.\ B\ j|$
using *BIJ card-of-Sigma-cong2* **by** *blast*
ultimately show *?thesis* **using** *ordIso-transitive* **by** *blast*
qed

lemma *ordIso-Sigma-cong1*:
assumes $\forall i \in I.\ p\ i =o\ r\ i$
shows $|SIGMA\ i : I.\ Field(p\ i)| =o\ |SIGMA\ i : I.\ Field(r\ i)|$
using *assms* **by** (*auto simp add*: *card-of-Sigma-cong1*)

lemma *ordLeq-Sigma-cong*:
assumes *bij-betw f I J* **and**
$\forall j \in J.\ p\ j =o\ r\ j$
shows $|SIGMA\ i : I.\ Field(p(f\ i))| =o\ |SIGMA\ j : J.\ Field(r\ j)|$
using *assms card-of-cong card-of-Sigma-cong*
[*of f I J* $\lambda\ j.\ Field(p\ j)\ \lambda\ j.\ Field(r\ j)$] **by** *blast*

corollary *card-of-Sigma-Times*:
$\forall i \in I.\ |A\ i| \leq o\ |B| \Longrightarrow |SIGMA\ i : I.\ A\ i| \leq o\ |I \times B|$
using *card-of-Sigma-mono1*[*of I A* $\lambda i.\ B$] **.**

corollary *ordLeq-Sigma-Times*:
$\forall i \in I.\ p\ i \leq o\ r \Longrightarrow |SIGMA\ i : I.\ Field\ (p\ i)| \leq o\ |I \times (Field\ r)|$
by (*auto simp add*: *card-of-Sigma-Times*)

lemma *card-of-UNION-Sigma*:
$|\bigcup i \in I.\ A\ i| \leq o\ |SIGMA\ i : I.\ A\ i|$
using *UNION-inj-on-Sigma*[*of I A*] *card-of-ordLeq* **by** *blast*

lemma *card-of-UNION-Sigma2*:

```
assumes
!! i j. ⟦{i,j} <= I; i ~= j⟧ ⟹ A i Int A j = {}
shows
|⋃ i∈I. A i| =o |Sigma I A|
proof−
  let ?L = ⋃ i∈I. A i  let ?R = Sigma I A
  have |?L| <=o |?R| using card-of-UNION-Sigma .
  moreover have |?R| <=o |?L|
  proof−
    have inj-on snd ?R
    unfolding inj-on-def using assms by auto
    moreover have snd ‘ ?R <= ?L by auto
    ultimately show ?thesis using card-of-ordLeq by blast
  qed
  ultimately show ?thesis by(simp add: ordIso-iff-ordLeq)
qed


lemma card-of-bool:
assumes a1 ≠ a2
shows |UNIV::bool set| =o |{a1,a2}|
proof−
  let ?f = λ bl. case bl of True ⇒ a1 | False ⇒ a2
  have bij-betw ?f UNIV {a1,a2}
  proof−
    {fix bl1 and bl2 assume ?f  bl1 = ?f bl2
     hence bl1 = bl2 using assms by (case-tac bl1, case-tac bl2, auto)
    }
    moreover
    {fix bl have ?f bl ∈ {a1,a2} by (case-tac bl, auto)
    }
    moreover
    {fix a assume *: a ∈ {a1,a2}
     have a ∈ ?f ‘ UNIV
     proof(cases a = a1)
       assume a = a1
       hence ?f True = a by auto  thus ?thesis by blast
     next
       assume a ≠ a1 hence a = a2 using * by auto
       hence ?f False = a by auto  thus ?thesis by blast
     qed
    }
    ultimately show ?thesis unfolding bij-betw-def inj-on-def by auto
  qed
  thus ?thesis using card-of-ordIso by blast
qed


lemma card-of-Plus-Times-aux:
```

```
assumes A2: a1 ≠ a2 ∧ {a1,a2} ≤ A and
        LEQ: |A| ≤o |B|
shows |A <+> B| ≤o |A × B|
proof−
  have 1: |UNIV::bool set| ≤o |A|
  using A2 card-of-mono1[of {a1,a2}] card-of-bool[of a1 a2]
        ordIso-ordLeq-trans[of |UNIV::bool set|] by blast

  have |A <+> B| ≤o |B <+> B|
  using LEQ card-of-Plus-mono1 by blast
  moreover have |B <+> B| =o |B × (UNIV::bool set)|
  using card-of-Plus-Times-bool by blast
  moreover have |B × (UNIV::bool set)| ≤o |B × A|
  using 1 by simp
  moreover have  |B × A| =o |A × B|
  using card-of-Times-commute by blast
  ultimately show |A <+> B| ≤o |A × B|
  using ordLeq-ordIso-trans[of |A <+> B| |B <+> B| |B × (UNIV::bool set)|]
        ordLeq-transitive[of |A <+> B| |B × (UNIV::bool set)| |B × A|]
        ordLeq-ordIso-trans[of |A <+> B| |B × A| |A × B|]
  by blast
qed


lemma card-of-Plus-Times:
assumes A2: a1 ≠ a2 ∧ {a1,a2} ≤ A and
        B2: b1 ≠ b2 ∧ {b1,b2} ≤ B
shows |A <+> B| ≤o |A × B|
proof−
  {assume |A| ≤o |B|
   hence ?thesis using assms by (auto simp add: card-of-Plus-Times-aux)
  }
  moreover
  {assume |B| ≤o |A|
   hence |B <+> A| ≤o |B × A|
   using assms by (auto simp add: card-of-Plus-Times-aux)
   hence ?thesis
   using card-of-Plus-commute card-of-Times-commute
         ordIso-ordLeq-trans ordLeq-ordIso-trans by blast
  }
  ultimately show ?thesis
  using card-of-Well-order[of A] card-of-Well-order[of B]
        ordLeq-total[of |A|] by blast
qed


corollary Plus-into-Times:
assumes A2: a1 ≠ a2 ∧ {a1,a2} ≤ A and
        B2: b1 ≠ b2 ∧ {b1,b2} ≤ B
```

shows $\exists f.$ *inj-on* f $(A <\!+\!> B) \wedge f$ ‘ $(A <\!+\!> B) \leq A \times B$
using *assms* **by** (*auto simp add*: *card-of-Plus-Times card-of-ordLeq*)

corollary *Plus-into-Times-types*:
assumes *A2*: $(a1::'a) \neq a2$ **and** *B2*: $(b1::'b) \neq b2$
shows $\exists (f::'a + 'b \Rightarrow 'a * 'b).$ *inj* f
using *assms Plus-into-Times*[*of a1 a2 UNIV b1 b2 UNIV*]
by *auto*

lemma *card-of-ordLeq-finite*:
assumes $|A| \leq o\ |B|$ **and** *finite B*
shows *finite A*
using *assms* **unfolding** *ordLeq-def*
using *embed-inj-on*[*of* $|A|$ $|B|$] *embed-Field*[*of* $|A|$ $|B|$]
Field-card-of[*of A*] *Field-card-of*[*of B*] *inj-on-finite*[*of - A B*] **by** *fastforce*

lemma *card-of-ordLeq-infinite*:
assumes $|A| \leq o\ |B|$ **and** *infinite A*
shows *infinite B*
using *assms card-of-ordLeq-finite* **by** *auto*

lemma *card-of-ordIso-finite*[*simp*]:
assumes $|A| =o\ |B|$
shows *finite A* = *finite B*
using *assms* **unfolding** *ordIso-def iso-def-raw*
by (*auto simp add*: *bij-betw-finite*)

lemma *card-of-ordIso-finite-Field*:
assumes *Card-order r* **and** $r =o\ |A|$
shows *finite*(*Field r*) = *finite A*
using *assms card-of-Field-ordIso card-of-ordIso-finite ordIso-equivalence* **by** *blast*

8.4 Cardinals versus set operations involving infinite sets

Here we show that, for infinite sets, most set-theoretic constructions do not increase the cardinality. The cornerstone for this is theorem *Card-order-Times-same-infinite*, which states that self-product does not increase cardinality – the proof of this fact adapts a standard set-theoretic argument, as presented, e.g., in the proof of theorem 1.5.11 at page 47 in [1]. Then everything else follows fairly easily.

lemma *infinite-iff-card-of-nat*:
infinite A = ($|UNIV::nat\ set| \leq o\ |A|$)
by (*auto simp add*: *infinite-iff-countable-subset card-of-ordLeq*)

lemma *finite-iff-cardOf-nat*:
finite A = (|*A*| <*o* |*UNIV* :: *nat set*|)
using *infinite-iff-card-of-nat*[*of A*]
not-ordLeq-iff-ordLess[*of* |*A*| |*UNIV* :: *nat set*|] **by** *fastforce*

lemma *finite-ordLess-infinite2*[*simp*]:
assumes *finite A* **and** *infinite B*
shows |*A*| <*o* |*B*|
using *assms*
finite-ordLess-infinite[*of* |*A*| |*B*|]
card-of-Well-order[*of A*] *card-of-Well-order*[*of B*]
Field-card-of[*of A*] *Field-card-of*[*of B*] **by** *auto*

The next two results correspond to the ZF fact that all infinite cardinals are limit ordinals:

lemma *Card-order-infinite-not-under*:
assumes *CARD*: *Card-order r* **and** *INF*: *infinite* (*Field r*)
shows ¬ (∃ *a*. *Field r* = *under r a*)
proof(*auto*)
  have *0*: *Well-order r* ∧ *wo-rel r* ∧ *Refl r*
  using *CARD* **unfolding** *wo-rel-def card-order-on-def order-on-defs* **by** *auto*
  fix *a* **assume** ∗: *Field r* = *under r a*
  show *False*
  proof(*cases a* ∈ *Field r*)
    assume *Case1*: *a* ∉ *Field r*
    hence *under r a* = {} **unfolding** *Field-def rel.under-def* **by** *auto*
    thus *False* **using** *INF* ∗ **by** *auto*
  next
    let *?r′* = *Restr r* (*underS r a*)
    assume *Case2*: *a* ∈ *Field r*
    hence *1*: *under r a* = *underS r a* ∪ {*a*} ∧ *a* ∉ *underS r a*
    using *0 rel.Refl-under-underS rel.underS-notIn* **by** *fastforce*
    have *2*: *ofilter r* (*underS r a*) ∧ *underS r a* < *Field r*
    using *0 wo-rel.underS-ofilter* ∗ *1 Case2* **by** *auto*
    hence *?r′* <*o r* **using** *0* **using** *ofilter-ordLess* **by** *blast*
    moreover
    have *Field ?r′* = *underS r a* ∧ *Well-order ?r′*
    using *2 0 Field-Restr-ofilter*[*of r*] *Well-order-Restr*[*of r*] **by** *blast*
    ultimately have |*underS r a*| <*o r* **using** *ordLess-Field*[*of ?r′*] **by** *auto*
    moreover have |*under r a*| =*o r* **using** ∗ *CARD card-of-Field-ordIso*[*of r*] **by** *auto*
    ultimately have |*underS r a*| <*o* |*under r a*|
    using *ordIso-symmetric ordLess-ordIso-trans* **by** *blast*
    moreover
    {**have** ∃ *f*. *bij-betw f* (*under r a*) (*underS r a*)
     using *infinite-imp-bij-betw*[*of Field r a*] *INF* ∗ *1* **by** *auto*

hence $|under\ r\ a| =o\ |underS\ r\ a|$ **using** *card-of-ordIso* **by** *blast*
}
ultimately show *False* **using** *not-ordLess-ordIso ordIso-symmetric* **by** *blast*
qed
qed

lemma *infinite-Card-order-limit*:
assumes *r*: *Card-order r* **and** *infinite (Field r)*
and *a*: *a : Field r*
shows $EX\ b : Field\ r.\ a \neq b \wedge (a,b) : r$
proof−
have $Field\ r \neq under\ r\ a$
using *assms Card-order-infinite-not-under* **by** *blast*
moreover have $under\ r\ a \leq Field\ r$
using *rel.under-Field* **.**
ultimately have $under\ r\ a < Field\ r$ **by** *blast*
then obtain *b* **where** *1*: $b : Field\ r \wedge \sim (b,a) : r$
unfolding *rel.under-def* **by** *blast*
moreover have *ba*: $b \neq a$
using *1 r* **unfolding** *card-order-on-def well-order-on-def*
linear-order-on-def partial-order-on-def preorder-on-def refl-on-def **by** *auto*
ultimately have $(a,b) : r$
using *a r* **unfolding** *card-order-on-def well-order-on-def linear-order-on-def*
total-on-def **by** *blast*
thus *?thesis* **using** *1 ba* **by** *auto*
qed

theorem *Card-order-Times-same-infinite*:
assumes *CO*: *Card-order r* **and** *INF*: *infinite(Field r)*
shows $|Field\ r \times Field\ r| \leq o\ r$
proof−
obtain *phi* **where** *phi-def*:
$phi = (\lambda r::'a\ rel.\ Card\text{-}order\ r \wedge infinite(Field\ r) \wedge$
$\neg\ |Field\ r \times Field\ r| \leq o\ r\ )$ **by** *blast*
have *temp1*: $\forall r.\ phi\ r \longrightarrow Well\text{-}order\ r$
unfolding *phi-def card-order-on-def* **by** *auto*
have *Ft*: $\neg(\exists r.\ phi\ r)$
proof
assume $\exists r.\ phi\ r$
hence $\{r.\ phi\ r\} \neq \{\} \wedge \{r.\ phi\ r\} \leq \{r.\ Well\text{-}order\ r\}$
using *temp1* **by** *auto*
then obtain *r* **where** *1*: *phi r* **and** *2*: $\forall r'.\ phi\ r' \longrightarrow r \leq o\ r'$ **and**
3: $Card\text{-}order\ r \wedge Well\text{-}order\ r$
using *exists-minim-Well-order[of {r. phi r}] temp1 phi-def* **by** *blast*
let $?A = Field\ r$ **let** $?r' = bsqr\ r$
have *4*: $Well\text{-}order\ ?r' \wedge Field\ ?r' = ?A \times ?A \wedge |?A| =o\ r$
using *3 bsqr-Well-order Field-bsqr card-of-Field-ordIso* **by** *blast*

have *5*: *Card-order* |?A × ?A| ∧ *Well-order* |?A × ?A|
using *card-of-Card-order card-of-Well-order* **by** *blast*

have *r* <o |?A × ?A|
using *1 3 5 ordLess-or-ordLeq* **unfolding** *phi-def* **by** *blast*
moreover have |?A × ?A| ≤o ?r′
using *card-of-least*[*of* ?A × ?A] *4* **by** *auto*
ultimately have *r* <o ?r′ **using** *ordLess-ordLeq-trans* **by** *auto*
then obtain *f* **where** *6*: *embed r ?r′ f* **and** *7*: ¬ *bij-betw f ?A* (?A × ?A)
unfolding *ordLess-def embedS-def-raw*
by (*auto simp add*: *Field-bsqr*)
let ?B = *f* ‘ ?A
have |?A| =o |?B|
using *3 6 embed-inj-on inj-on-imp-bij-betw card-of-ordIso* **by** *blast*
hence *8*: *r* =o |?B| **using** *4 ordIso-transitive ordIso-symmetric* **by** *blast*

have *ofilter ?r′ ?B*
using *6 embed-Field-ofilter 3 4* **by** *blast*
hence *ofilter ?r′ ?B* ∧ ?B ≠ ?A × ?A ∧ ?B ≠ *Field ?r′*
using *7* **unfolding** *bij-betw-def* **using** *6 3 embed-inj-on 4* **by** *auto*
hence *temp2*: *ofilter ?r′ ?B* ∧ ?B < ?A × ?A
using *4 wo-rel-def*[*of ?r′*] *wo-rel.ofilter-def*[*of ?r′ ?B*] **by** *blast*
have ¬ (∃ *a*. *Field r* = *under r a*)
using *1* **unfolding** *phi-def* **using** *Card-order-infinite-not-under*[*of r*] **by** *auto*
then obtain *A1* **where** *temp3*: *ofilter r A1* ∧ *A1* < ?A **and** *9*: ?B ≤ *A1* × *A1*
using *temp2 3 bsqr-ofilter*[*of r ?B*] **by** *blast*
hence |?B| ≤o |*A1* × *A1*| **using** *card-of-mono1* **by** *blast*
hence *10*: *r* ≤o |*A1* × *A1*| **using** *8 ordIso-ordLeq-trans* **by** *blast*
let ?r1 = *Restr r A1*
have ?r1 <o *r* **using** *temp3 ofilter-ordLess 3* **by** *blast*
moreover
{**have** *well-order-on A1 ?r1* **using** *3 temp3 well-order-on-Restr* **by** *blast*
hence |*A1*| ≤o ?r1 **using** *3 Well-order-Restr card-of-least* **by** *blast*
}
ultimately have *11*: |*A1*| <o *r* **using** *ordLeq-ordLess-trans* **by** *blast*

have *infinite* (*Field r*) **using** *1* **unfolding** *phi-def* **by** *simp*
hence *infinite ?B* **using** *8 3 card-of-ordIso-finite-Field*[*of r ?B*] **by** *blast*
hence *infinite A1* **using** *9 infinite-super finite-cartesian-product* **by** *blast*
moreover have *temp4*: *Field* |*A1*| = *A1* ∧ *Well-order* |*A1*| ∧ *Card-order* |*A1*|
using *card-of-Card-order*[*of A1*] *card-of-Well-order*[*of A1*] **by** *auto*
moreover have ¬ *r* ≤o | *A1* |
using *temp4 11 3* **using** *not-ordLeq-iff-ordLess* **by** *blast*
ultimately have *infinite*(*Field* |*A1*|) ∧ *Card-order* |*A1*| ∧ ¬ *r* ≤o | *A1* | **by** *simp*
hence |*Field* |*A1*| × *Field* |*A1*| | ≤o |*A1*|
using *2* **unfolding** *phi-def* **by** *blast*
hence |*A1* × *A1* | ≤o |*A1*| **using** *temp4* **by** *auto*

hence $r \le o\ |A1|$ **using** *10 ordLeq-transitive* **by** *blast*
thus *False* **using** *11 not-ordLess-ordLeq* **by** *auto*
qed
thus *?thesis* **using** *assms* **unfolding** *phi-def* **by** *blast*
qed

corollary *card-of-Times-same-infinite*[*simp*]:
assumes *infinite A*
shows $|A \times A| = o\ |A|$
proof−
let $?r = |A|$
have *Field ?r* $= A \wedge$ *Card-order ?r*
using *Field-card-of card-of-Card-order*[*of A*] **by** *fastforce*
hence $|A \times A| \le o\ |A|$
using *Card-order-Times-same-infinite*[*of ?r*] *assms* **by** *auto*
thus *?thesis* **using** *card-of-Times3 ordIso-iff-ordLeq* **by** *blast*
qed

corollary *Times-same-infinite-bij-betw*:
assumes *infinite A*
shows $\exists f.$ *bij-betw f* $(A \times A)$ *A*
using *assms* **by** (*auto simp add*: *card-of-ordIso*)

corollary *Times-same-infinite-bij-betw-types*:
assumes *INF*: *infinite*(*UNIV*::'*a set*)
shows $\exists (f::('a * 'a) => 'a).$ *bij f*
using *assms Times-same-infinite-bij-betw*[*of UNIV*::'*a set*]
using *bij-bij-betw* **by** *auto*

lemma *card-of-Times-infinite*:
assumes *INF*: *infinite A* **and** *NE*: $B \neq \{\}$ **and** *LEQ*: $|B| \le o\ |A|$
shows $|A \times B| = o\ |A| \wedge |B \times A| = o\ |A|$
proof−
have $|A| \le o\ |A \times B| \wedge |A| \le o\ |B \times A|$
using *assms* **by** *auto*
moreover
{**have** $|A \times B| \le o\ |A \times A| \wedge |B \times A| \le o\ |A \times A|$
using *LEQ card-of-Times-mono1 card-of-Times-mono2* **by** *blast*
moreover have $|A \times A| = o\ |A|$ **using** *INF card-of-Times-same-infinite* **by** *blast*
ultimately have $|A \times B| \le o\ |A| \wedge |B \times A| \le o\ |A|$
using *ordLeq-ordIso-trans*[*of* $|A \times B|$] *ordLeq-ordIso-trans*[*of* $|B \times A|$] **by** *auto*
}
ultimately show *?thesis* **by** (*auto simp add*: *ordIso-iff-ordLeq*)
qed

corollary *card-of-Times-infinite-simps*[*simp*]:
$[\![$*infinite* A; $B \neq \{\}$; $|B| \leq o\ |A|]\!] \Longrightarrow |A \times B| =o\ |A|$
$[\![$*infinite* A; $B \neq \{\}$; $|B| \leq o\ |A|]\!] \Longrightarrow |A| =o\ |A \times B|$
$[\![$*infinite* A; $B \neq \{\}$; $|B| \leq o\ |A|]\!] \Longrightarrow |B \times A| =o\ |A|$
$[\![$*infinite* A; $B \neq \{\}$; $|B| \leq o\ |A|]\!] \Longrightarrow |A| =o\ |B \times A|$
by(*auto simp add*: *card-of-Times-infinite ordIso-symmetric*)

corollary *Card-order-Times-infinite*:
assumes *INF*: *infinite*(*Field r*) **and** *CARD*: *Card-order r* **and**
NE: *Field p* $\neq \{\}$ **and** *LEQ*: $p \leq o\ r$
shows | (*Field r*) $\times$ (*Field p*) | $=o\ r \wedge$ | (*Field p*) $\times$ (*Field r*) | $=o\ r$
proof−
have |*Field r* $\times$ *Field p*| $=o$ |*Field r*| $\wedge$ |*Field p* $\times$ *Field r*| $=o$ |*Field r*|
using *assms* **by** (*auto simp add*: *card-of-Times-infinite*)
thus *?thesis*
using *assms card-of-Field-ordIso*[*of r*]
ordIso-transitive[*of* |*Field r* $\times$ *Field p*|]
ordIso-transitive[*of* - |*Field r*|] **by** *blast*
qed

corollary *Times-infinite-bij-betw*:
assumes *INF*: *infinite A* **and** *NE*: $B \neq \{\}$ **and** *INJ*: *inj-on g B* $\wedge$ *g* ‘ $B \leq A$
shows ($\exists f$. *bij-betw f* ($A \times B$) A) $\wedge$ ($\exists h$. *bij-betw h* ($B \times A$) A)
proof−
have $|B| \leq o\ |A|$ **using** *INJ card-of-ordLeq* **by** *blast*
thus *?thesis* **using** *INF NE*
by (*auto simp add*: *card-of-ordIso card-of-Times-infinite*)
qed

corollary *Times-infinite-bij-betw-types*:
assumes *INF*: *infinite*(*UNIV*::'*a set*) **and**
BIJ: *inj*(*g*::'*b* $\Rightarrow$ '*a*)
shows ($\exists$(*f*::('*b* $*$ '*a*) => '*a*). *bij f*) $\wedge$ ($\exists$(*h*::('*a* $*$ '*b*) => '*a*). *bij h*)
using *assms Times-infinite-bij-betw*[*of UNIV*::'*a set UNIV*::'*b set g*]
using *bij-bij-betw* **by** *auto*

lemma *card-of-Sigma-ordLeq-infinite*:
assumes *INF*: *infinite B* **and**
LEQ-I: $|I| \leq o\ |B|$ **and** *LEQ*: $\forall i \in I.\ |A\ i| \leq o\ |B|$
shows |*SIGMA* $i : I.\ A\ i| \leq o\ |B|$
proof(*cases* $I = \{\}$, *simp*)
assume $*$: $I \neq \{\}$
have |*SIGMA* $i : I.\ A\ i| \leq o\ |I \times B|$
using *LEQ card-of-Sigma-Times* **by** *blast*

moreover have $|I \times B| =o |B|$
using *INF* $*$ *LEQ-I* **by** (*auto simp add*: *card-of-Times-infinite*)
ultimately show *?thesis* **using** *ordLeq-ordIso-trans* **by** *blast*
qed

lemma *card-of-Sigma-ordLeq-infinite-Field*:
assumes *INF*: *infinite* (*Field r*) **and** *r*: *Card-order r* **and**
LEQ-I: $|I| \leq o\ r$ **and** *LEQ*: $\forall i \in I.\ |A\ i| \leq o\ r$
shows $|SIGMA\ i : I.\ A\ i| \leq o\ r$
proof−
let *?B* = *Field r*
have *1*: $r =o |?B| \wedge |?B| =o\ r$ **using** *r card-of-Field-ordIso*
ordIso-symmetric **by** *blast*
hence $|I| \leq o |?B|$ $\forall i \in I.\ |A\ i| \leq o |?B|$
using *LEQ-I LEQ ordLeq-ordIso-trans* **by** *blast+*
hence $|SIGMA\ i : I.\ A\ i| \leq o |?B|$ **using** *INF LEQ*
card-of-Sigma-ordLeq-infinite **by** *blast*
thus *?thesis* **using** *1 ordLeq-ordIso-trans* **by** *blast*
qed

lemma *card-of-Times-ordLeq-infinite*:
$[\![$*infinite C*; $|A| \leq o |C|$; $|B| \leq o |C|]\!]$
$\Longrightarrow |A <*> B| \leq o |C|$
by(*simp add*: *card-of-Sigma-ordLeq-infinite*)

lemma *card-of-Times-ordLeq-infinite-Field*:
$[\![$*infinite* (*Field r*); $|A| \leq o\ r$; $|B| \leq o\ r$; *Card-order r*$]\!]$
$\Longrightarrow |A <*> B| \leq o\ r$
by(*simp add*: *card-of-Sigma-ordLeq-infinite-Field*)

lemma *card-of-UNION-ordLeq-infinite*:
assumes *INF*: *infinite B* **and**
LEQ-I: $|I| \leq o |B|$ **and** *LEQ*: $\forall i \in I.\ |A\ i| \leq o |B|$
shows $|\bigcup\ i \in I.\ A\ i| \leq o |B|$
proof(*cases* $I = \{\}$, *simp*)
assume $*$: $I \neq \{\}$
have $|\bigcup\ i \in I.\ A\ i| \leq o |SIGMA\ i : I.\ A\ i|$
using *card-of-UNION-Sigma* **by** *blast*
moreover have $|SIGMA\ i : I.\ A\ i| \leq o |B|$
using *assms card-of-Sigma-ordLeq-infinite* **by** *blast*
ultimately show *?thesis* **using** *ordLeq-transitive* **by** *blast*
qed

corollary *card-of-UNION-ordLeq-infinite-Field*:

assumes *INF*: *infinite* (*Field r*) **and** *r*: *Card-order r* **and**
  LEQ-I: $|I| \leq o\ r$ **and** *LEQ*: $\forall\, i \in I.\ |A\ i| \leq o\ r$
shows $|\bigcup\ i \in I.\ A\ i| \leq o\ r$
proof−
  let *?B* = *Field r*
  have *1*: $r =o\ |?B| \wedge |?B| =o\ r$ **using** *r card-of-Field-ordIso*
  ordIso-symmetric **by** *blast*
  hence $|I| \leq o\ |?B|$ $\forall\, i \in I.\ |A\ i| \leq o\ |?B|$
  using *LEQ-I LEQ ordLeq-ordIso-trans* **by** *blast+*
  hence $|\bigcup\ i \in I.\ A\ i| \leq o\ |?B|$ **using** *INF LEQ*
  card-of-UNION-ordLeq-infinite **by** *blast*
  thus *?thesis* **using** *1 ordLeq-ordIso-trans* **by** *blast*
qed

lemma *card-of-Plus-infinite1*[*simp*]:
assumes *INF*: *infinite A* **and** *LEQ*: $|B| \leq o\ |A|$
shows $|A <+> B| =o\ |A|$
proof(*cases* $B = \{\}$, *simp add*: *card-of-Plus-empty1 card-of-Plus-empty2 ordIso-symmetric*)
  let $?Inl = Inl{::}'a \Rightarrow 'a + 'b$ **let** $?Inr = Inr{::}'b \Rightarrow 'a + 'b$
  assume $*$: $B \neq \{\}$
  then obtain *b1* **where** *1*: $b1 \in B$ **by** *blast*
  show *?thesis*
  proof(*cases* $B = \{b1\}$)
    assume *Case1*: $B = \{b1\}$
    have *2*: *bij-betw ?Inl A* ((*?Inl* ‘ *A*))
    unfolding *bij-betw-def inj-on-def* **by** *auto*
    hence *3*: *infinite* (*?Inl* ‘ *A*)
    using *INF bij-betw-finite*[*of ?Inl A*] **by** *blast*
    let $?A' = ?Inl$ ‘ $A \cup \{?Inr\ b1\}$
    obtain *g* **where** *bij-betw g* (*?Inl* ‘ *A*) *?A'*
    using *3 infinite-imp-bij-betw2*[*of ?Inl* ‘ *A*] **by** *auto*
    moreover have $?A' = A <+> B$ **using** *Case1* **by** *blast*
    ultimately have *bij-betw g* (*?Inl* ‘ *A*) ($A <+> B$) **by** *simp*
    hence *bij-betw* (*g o ?Inl*) *A* ($A <+> B$)
    using *2* **by** (*auto simp add*: *bij-betw-comp*)
    thus *?thesis* **using** *card-of-ordIso ordIso-symmetric* **by** *blast*
  next
    assume *Case2*: $B \neq \{b1\}$
    with $*$ *1* **obtain** *b2* **where** *3*: $b1 \neq b2 \wedge \{b1,b2\} \leq B$ **by** *fastforce*
    obtain *f* **where** *inj-on f B* $\wedge$ *f* ‘ $B \leq A$
    using *LEQ card-of-ordLeq*[*of B*] **by** *fastforce*
    with *3* **have** $f\ b1 \neq f\ b2 \wedge \{f\ b1, f\ b2\} \leq A$
    unfolding *inj-on-def* **by** *auto*
    with *3* **have** $|A <+> B| \leq o\ |A \times B|$
    by (*auto simp add*: *card-of-Plus-Times*)
    moreover have $|A \times B| =o\ |A|$
    using *assms* $*$ **by** *simp*
    ultimately have $|A <+> B| \leq o\ |A|$ **using** *ordLeq-ordIso-trans* **by** *auto*

thus *?thesis* **using** *card-of-Plus1 ordIso-iff-ordLeq* **by** *blast*
qed
qed

lemma *card-of-Plus-infinite2*[*simp*]:
assumes *INF*: *infinite A* **and** *LEQ*: $|B| \leq o\ |A|$
shows |*B* <+> *A*| =*o* |*A*|
using *assms card-of-Plus-commute card-of-Plus-infinite1*
ordIso-equivalence **by** *blast*

lemma *card-of-Plus-infinite*:
assumes *INF*: *infinite A* **and** *LEQ*: $|B| \leq o\ |A|$
shows |*A* <+> *B*| =*o* |*A*| ∧ |*B* <+> *A*| =*o* |*A*|
using *assms* **by** *auto*

corollary *Card-order-Plus-infinite*:
assumes *INF*: *infinite*(*Field r*) **and** *CARD*: *Card-order r* **and**
LEQ: $p \leq o\ r$
shows | (*Field r*) <+> (*Field p*) | =*o r* ∧ | (*Field p*) <+> (*Field r*) | =*o r*
proof−
have | *Field r* <+> *Field p* | =*o* | *Field r* | ∧
| *Field p* <+> *Field r* | =*o* | *Field r* |
using *assms* **by** (*auto simp add*: *card-of-Plus-infinite*)
thus *?thesis*
using *assms card-of-Field-ordIso*[*of r*]
ordIso-transitive[*of* |*Field r* <+> *Field p*|]
ordIso-transitive[*of* - |*Field r*|] **by** *blast*
qed

corollary *Plus-infinite-bij-betw*:
assumes *INF*: *infinite A* **and** *INJ*: *inj-on g B* ∧ *g* ‘ *B* ≤ *A*
shows (∃*f*. *bij-betw f* (*A* <+> *B*) *A*) ∧ (∃*h*. *bij-betw h* (*B* <+> *A*) *A*)
proof−
have $|B| \leq o\ |A|$ **using** *INJ card-of-ordLeq* **by** *blast*
thus *?thesis* **using** *INF*
by (*auto simp add*: *card-of-ordIso*)
qed

corollary *Plus-infinite-bij-betw-types*:
assumes *INF*: *infinite*(*UNIV*::*'a set*) **and**
BIJ: *inj*(*g*::*'b* ⇒ *'a*)
shows (∃(*f*::(*'b* + *'a*) => *'a*). *bij f*) ∧ (∃(*h*::(*'a* + *'b*) => *'a*). *bij h*)
using *assms Plus-infinite-bij-betw*[*of UNIV*::*'a set g UNIV*::*'b set*]
using *bij-bij-betw* **by** *auto*

lemma *card-of-Un-infinite*:
assumes *INF*: *infinite* A **and** *LEQ*: $|B| \leq o\ |A|$
shows $|A \cup B| =o\ |A| \wedge |B \cup A| =o\ |A|$
proof−
  have $|A \cup B| \leq o\ |A <+> B|$ **by** *simp*
  moreover have $|A <+> B| =o\ |A|$
  using *assms* **by** *simp*
  ultimately have $|A \cup B| \leq o\ |A|$ **using** *ordLeq-ordIso-trans* **by** *blast*
  hence $|A \cup B| =o\ |A|$ **using** *card-of-Un1 ordIso-iff-ordLeq* **by** *blast*
  thus *?thesis* **using** *Un-commute*[*of B A*] **by** *auto*
qed

lemma *card-of-Un-infinite-simps*[*simp*]:
$[\![$*infinite* A; $|B| \leq o\ |A|\ ]\!] \Longrightarrow |A \cup B| =o\ |A|$
$[\![$*infinite* A; $|B| \leq o\ |A|\ ]\!] \Longrightarrow |B \cup A| =o\ |A|$
using *card-of-Un-infinite* **by** *auto*

corollary *Card-order-Un-infinite*:
assumes *INF*: *infinite*(*Field r*) **and** *CARD*: *Card-order r* **and**
    LEQ: $p \leq o\ r$
shows | (*Field r*) ∪ (*Field p*) | $=o\ r \wedge$ | (*Field p*) ∪ (*Field r*) | $=o\ r$
proof−
  have | *Field r* ∪ *Field p* | $=o$ | *Field r* | ∧
    | *Field p* ∪ *Field r* | $=o$ | *Field r* |
  using *assms* **by** (*auto simp add*: *card-of-Un-infinite*)
  thus *?thesis*
  using *assms card-of-Field-ordIso*[*of r*]
    ordIso-transitive[*of* |*Field r* ∪ *Field p*|]
    ordIso-transitive[*of* - |*Field r*|] **by** *blast*
qed

lemma *card-of-Un-diff-infinite*:
assumes *INF*: *infinite* A **and** *LESS*: $|B| <o\ |A|$
shows $|A - B| =o\ |A|$
proof−
  obtain C **where** *C-def*: $C = A - B$ **by** *blast*
  have $|A \cup B| =o\ |A|$
  using *assms ordLeq-iff-ordLess-or-ordIso card-of-Un-infinite* **by** *blast*
  moreover have $C \cup B = A \cup B$ **unfolding** *C-def* **by** *auto*
  ultimately have *1*: $|C \cup B| =o\ |A|$ **by** *auto*

  {**assume** ∗: $|C| \leq o\ |B|$
   moreover
   {**assume** ∗∗: *finite* B
    hence *finite* C

using *card-of-ordLeq-finite* $*$ **by** *blast*
hence *False* **using** $**$ *INF card-of-ordIso-finite 1* **by** *blast*
}
hence *infinite B* **by** *auto*
ultimately have *False*
using *card-of-Un-infinite 1 ordIso-equivalence*
LESS not-ordLess-ordIso **by** *blast*
}
hence *2*: $|B| \leq o\ |C|$ **using** *card-of-Well-order ordLeq-total* **by** *blast*
{assume $*$: *finite C*
hence *finite B* **using** *card-of-ordLeq-finite 2* **by** *blast*
hence *False* **using** $*$ *INF card-of-ordIso-finite 1* **by** *blast*
}
hence *infinite C* **by** *auto*
hence $|C| =o\ |A|$
using *card-of-Un-infinite 1 2 ordIso-equivalence* **by** *blast*
thus *?thesis* **unfolding** *C-def* **.**
qed

corollary *subset-ordLeq-diff-infinite*:
assumes *INF*: *infinite B* **and** *SUB*: $A \leq B$ **and** *LESS*: $|A| <o\ |B|$
shows *infinite* $(B - A)$
using *assms card-of-Un-diff-infinite card-of-ordIso-finite* **by** *blast*

lemma *card-of-Times-ordLess-infinite*[*simp*]:
assumes *INF*: *infinite C* **and**
LESS1: $|A| <o\ |C|$ **and** *LESS2*: $|B| <o\ |C|$
shows $|A \times B| <o\ |C|$
proof(*cases* $A = \{\} \vee B = \{\}$)
assume *Case1*: $A = \{\} \vee B = \{\}$
hence $A \times B = \{\}$ **by** *blast*
moreover have $C \neq \{\}$ **using**
LESS1 card-of-empty5 **by** *blast*
ultimately show *?thesis* **by**(*auto simp add*: *card-of-empty4*)
next
assume *Case2*: $\neg(A = \{\} \vee B = \{\})$
{assume $*$: $|C| \leq o\ |A \times B|$
hence *infinite* $(A \times B)$ **using** *INF card-of-ordLeq-finite* **by** *blast*
hence *1*: *infinite A* $\vee$ *infinite B* **using** *finite-cartesian-product* **by** *blast*
{assume *Case21*: $|A| \leq o\ |B|$
hence *infinite B* **using** *1 card-of-ordLeq-finite* **by** *blast*
hence $|A \times B| =o\ |B|$ **using** *Case2 Case21*
by (*auto simp add*: *card-of-Times-infinite*)
hence *False* **using** *LESS2 not-ordLess-ordLeq* $*$ *ordLeq-ordIso-trans* **by** *blast*
}
moreover
{assume *Case22*: $|B| \leq o\ |A|$

hence *infinite A* **using** *1 card-of-ordLeq-finite* **by** *blast*
hence $|A \times B| =o\ |A|$ **using** *Case2 Case22*
by (*auto simp add*: *card-of-Times-infinite*)
hence *False* **using** *LESS1 not-ordLess-ordLeq* $*$ *ordLeq-ordIso-trans* **by** *blast*
}
ultimately have *False* **using** *ordLeq-total card-of-Well-order*[*of A*]
card-of-Well-order[*of B*] **by** *blast*
}
thus *?thesis* **using** *ordLess-or-ordLeq*[*of* $|A \times B|$ $|C|$]
card-of-Well-order[*of* $A \times B$] *card-of-Well-order*[*of C*] **by** *auto*
qed

lemma *card-of-Times-ordLess-infinite-Field*[*simp*]:
assumes *INF*: *infinite* (*Field r*) **and** *r*: *Card-order r* **and**
LESS1: $|A| <o\ r$ **and** *LESS2*: $|B| <o\ r$
shows $|A \times B| <o\ r$
proof−
let *?C* = *Field r*
have *1*: $r =o\ |?C| \wedge |?C| =o\ r$ **using** *r card-of-Field-ordIso*
ordIso-symmetric **by** *blast*
hence $|A| <o\ |?C|$ $|B| <o\ |?C|$
using *LESS1 LESS2 ordLess-ordIso-trans* **by** *blast*+
hence $|A <*> B| <o\ |?C|$ **using** *INF*
card-of-Times-ordLess-infinite **by** *blast*
thus *?thesis* **using** *1 ordLess-ordIso-trans* **by** *blast*
qed

lemma *card-of-Plus-ordLess-infinite*[*simp*]:
assumes *INF*: *infinite C* **and**
LESS1: $|A| <o\ |C|$ **and** *LESS2*: $|B| <o\ |C|$
shows $|A <+> B| <o\ |C|$
proof(*cases* $A = \{\} \vee B = \{\}$)
assume *Case1*: $A = \{\} \vee B = \{\}$
hence $|A| =o\ |A <+> B| \vee |B| =o\ |A <+> B|$
using *card-of-Plus-empty1 card-of-Plus-empty2* **by** *blast*
hence $|A <+> B| =o\ |A| \vee |A <+> B| =o\ |B|$
using *ordIso-symmetric*[*of* $|A|$] *ordIso-symmetric*[*of* $|B|$] **by** *blast*
thus *?thesis* **using** *LESS1 LESS2*
ordIso-ordLess-trans[*of* $|A <+> B|$ $|A|$]
ordIso-ordLess-trans[*of* $|A <+> B|$ $|B|$] **by** *blast*
next
assume *Case2*: $\neg(A = \{\} \vee B = \{\})$
{assume $*$: $|C| \leq o\ |A <+> B|$
hence *infinite* ($A <+> B$) **using** *INF card-of-ordLeq-finite* **by** *blast*
hence *1*: *infinite A* $\vee$ *infinite B* **using** *finite-Plus* **by** *blast*
{assume *Case21*: $|A| \leq o\ |B|$
hence *infinite B* **using** *1 card-of-ordLeq-finite* **by** *blast*

hence $|A <+> B| =o |B|$ **using** *Case2 Case21*
by (*auto simp add*: *card-of-Plus-infinite*)
hence *False* **using** *LESS2 not-ordLess-ordLeq* $*$ *ordLeq-ordIso-trans* **by** *blast*
}
moreover
{assume *Case22*: $|B| \leq o |A|$
hence *infinite A* **using** *1 card-of-ordLeq-finite* **by** *blast*
hence $|A <+> B| =o |A|$ **using** *Case2 Case22*
by (*auto simp add*: *card-of-Plus-infinite*)
hence *False* **using** *LESS1 not-ordLess-ordLeq* $*$ *ordLeq-ordIso-trans* **by** *blast*
}
ultimately have *False* **using** *ordLeq-total card-of-Well-order*[*of A*]
card-of-Well-order[*of B*] **by** *blast*
}
thus *?thesis* **using** *ordLess-or-ordLeq*[*of* $|A <+> B|$ $|C|$]
card-of-Well-order[*of A <+> B*] *card-of-Well-order*[*of C*] **by** *auto*
qed

lemma *card-of-Plus-ordLess-infinite-Field*[*simp*]:
assumes *INF*: *infinite* (*Field r*) **and** *r*: *Card-order r* **and**
LESS1: $|A| <o\ r$ **and** *LESS2*: $|B| <o\ r$
shows $|A <+> B| <o\ r$
proof−
let *?C* = *Field r*
have *1*: $r =o\ |?C| \wedge |?C| =o\ r$ **using** *r card-of-Field-ordIso*
ordIso-symmetric **by** *blast*
hence $|A| <o\ |?C|$ $|B| <o\ |?C|$
using *LESS1 LESS2 ordLess-ordIso-trans* **by** *blast+*
hence $|A <+> B| <o\ |?C|$ **using** *INF*
card-of-Plus-ordLess-infinite **by** *blast*
thus *?thesis* **using** *1 ordLess-ordIso-trans* **by** *blast*
qed

lemma *card-of-Un-ordLess-infinite*[*simp*]:
assumes *INF*: *infinite C* **and**
LESS1: $|A| <o\ |C|$ **and** *LESS2*: $|B| <o\ |C|$
shows $|A \cup B| <o\ |C|$
using *assms card-of-Plus-ordLess-infinite card-of-Un-Plus-ordLeq*
ordLeq-ordLess-trans **by** *blast*

lemma *card-of-Un-ordLess-infinite-Field*[*simp*]:
assumes *INF*: *infinite* (*Field r*) **and** *r*: *Card-order r* **and**
LESS1: $|A| <o\ r$ **and** *LESS2*: $|B| <o\ r$
shows $|A\ Un\ B| <o\ r$
proof−
let *?C* = *Field r*

have *1*: $r =o\ |?C| \wedge |?C| =o\ r$ **using** *r card-of-Field-ordIso ordIso-symmetric* **by** *blast*
hence $|A| <o\ |?C|\ \ |B| <o\ |?C|$
using *LESS1 LESS2 ordLess-ordIso-trans* **by** *blast+*
hence $|A\ Un\ B| <o\ |?C|$ **using** *INF card-of-Un-ordLess-infinite* **by** *blast*
thus *?thesis* **using** *1 ordLess-ordIso-trans* **by** *blast*
qed

lemma *card-of-Un-singl-ordLess-infinite1*:
assumes *infinite B* **and** $|A| <o\ |B|$
shows $|\{a\}\ Un\ A| <o\ |B|$
proof−
have $|\{a\}| <o\ |B|$ **using** *assms* **by** *auto*
thus *?thesis* **using** *assms card-of-Un-ordLess-infinite*[*of B*] **by** *fastforce*
qed

lemma *card-of-Un-singl-ordLess-infinite*:
assumes *infinite B*
shows $(\ |A| <o\ |B|\ ) = (\ |\{a\}\ Un\ A| <o\ |B|\ )$
using *assms card-of-Un-singl-ordLess-infinite1*[*of B A*]
proof(*auto*)
assume $|insert\ a\ A| <o\ |B|$
moreover have $|A| <=o\ |insert\ a\ A|$ **using** *card-of-mono1*[*of A*] **by** *blast*
ultimately show $|A| <o\ |B|$ **using** *ordLeq-ordLess-trans* **by** *blast*
qed

8.5 Cardinals versus lists

The next is an auxiliary operator, which shall be used for inductive proofs of facts concerning the cardinality of *List* :

definition *nlists* :: $'a\ set \Rightarrow nat \Rightarrow 'a\ list\ set$
where $nlists\ A\ n \equiv \{l.\ set\ l \le A \wedge length\ l = n\}$

lemma *lists-def2*:
$lists\ A = \{l.\ set\ l \le A\}$
using *in-listsI* **by** *blast*

lemma *lists-UNION-nlists*: $lists\ A = (\bigcup\ n.\ nlists\ A\ n)$
unfolding *lists-def2 nlists-def* **by** *blast*

lemma *card-of-lists*: $|A| \le o\ |lists\ A|$
proof−
let $?h = \lambda\ a.\ [a]$

have *inj-on ?h A* $\wedge$ *?h* ‘ *A* $\leq$ *lists A*
unfolding *inj-on-def lists-def2* **by** *auto*
thus *?thesis* **using** *card-of-ordLeq* **by** *blast*
qed

lemma *Card-order-lists*: *Card-order r* $\Longrightarrow$ *r* $\leq$*o* |*lists*(*Field r*) |
using *card-of-lists card-of-Field-ordIso ordIso-ordLeq-trans ordIso-symmetric* **by** *blast*

lemma *Union-set-lists*:
Union(*set* ‘ (*lists A*)) = *A*
unfolding *lists-def2* **proof**(*auto*)
fix *a* **assume** *a* $\in$ *A*
hence *set* [*a*] $\leq$ *A* $\wedge$ *a* $\in$ *set* [*a*] **by** *auto*
thus $\exists$ *l*. *set l* $\leq$ *A* $\wedge$ *a* $\in$ *set l* **by** *blast*
qed

lemma *inj-on-map-lists*:
assumes *inj-on f A*
shows *inj-on* (*map f*) (*lists A*)
using *assms Union-set-lists*[*of A*] *inj-on-mapI*[*of f lists A*] **by** *auto*

lemma *map-lists-mono*:
assumes *f* ‘ *A* $\leq$ *B*
shows (*map f*) ‘ (*lists A*) $\leq$ *lists B*
using *assms* **unfolding** *lists-def2* **by** (*auto*, *blast*)

lemma *map-lists-surjective*:
assumes *f* ‘ *A* = *B*
shows (*map f*) ‘ (*lists A*) = *lists B*
using *assms* **unfolding** *lists-def2*
proof (*auto*, *blast*)
fix *l'* **assume** $*$: *set l'* $\leq$ *f* ‘ *A*
have *set l'* $\leq$ *f* ‘ *A* $\longrightarrow$ *l'* $\in$ *map f* ‘ {*l*. *set l* $\leq$ *A*}
proof(*induct l'*, *auto*)
fix *l a*
assume *a* $\in$ *A* **and** *set l* $\leq$ *A* **and**
IH: *f* ‘ (*set l*) $\leq$ *f* ‘ *A*
hence *set* (*a* # *l*) $\leq$ *A* **by** *auto*
hence *map f* (*a* # *l*) $\in$ *map f* ‘ {*l*. *set l* $\leq$ *A*} **by** *blast*
thus *f a* # *map f l* $\in$ *map f* ‘ {*l*. *set l* $\leq$ *A*} **by** *auto*
qed
thus *l'* $\in$ *map f* ‘ {*l*. *set l* $\leq$ *A*} **using** $*$ **by** *auto*
qed

lemma *bij-betw-map-lists*:
assumes *bij-betw f A B*
shows *bij-betw* (*map f*) (*lists A*) (*lists B*)
using *assms* **unfolding** *bij-betw-def*
by(*auto simp add*: *inj-on-map-lists map-lists-surjective*)

lemma *card-of-lists-mono*[*simp*]:
assumes $|A| \leq o\ |B|$
shows $|$*lists A*$| \leq o\ |$*lists B*$|$
proof−
  obtain *f* **where** *inj-on f A* $\wedge$ *f* ‘ *A* $\leq$ *B*
  using *assms card-of-ordLeq*[*of A B*] **by** *auto*
  hence *inj-on* (*map f*) (*lists A*) $\wedge$ (*map f*) ‘ (*lists A*) $\leq$ (*lists B*)
  by (*auto simp add*: *inj-on-map-lists map-lists-mono*)
  thus *?thesis* **using** *card-of-ordLeq*[*of lists A*] **by** *auto*
qed

lemma *ordIso-lists-mono*:
assumes $r \leq o\ r'$
shows $|$*lists*(*Field r*)$| \leq o\ |$*lists*(*Field r*$'$)$|$
using *assms card-of-mono2 card-of-lists-mono* **by** *blast*

lemma *card-of-lists-cong*[*simp*]:
assumes $|A| =o\ |B|$
shows $|$*lists A*$| =o\ |$*lists B*$|$
proof−
  obtain *f* **where** *bij-betw f A B*
  using *assms card-of-ordIso*[*of A B*] **by** *auto*
  hence *bij-betw* (*map f*) (*lists A*) (*lists B*)
  by (*auto simp add*: *bij-betw-map-lists*)
  thus *?thesis* **using** *card-of-ordIso*[*of lists A*] **by** *auto*
qed

lemma *ordIso-lists-cong*:
assumes $r =o\ r'$
shows $|$*lists*(*Field r*)$| =o\ |$*lists*(*Field r*$'$)$|$
using *assms card-of-cong card-of-lists-cong* **by** *blast*

lemma *length-Suc*: ($\exists$ *n*. *length l* = *Suc n*) = ($\exists$ *a l*$'$. *l* = *a* # *l*$'$)
by(*induct l*, *auto*)

lemma *nlists-0*: *nlists A 0* = {[]}
unfolding *nlists-def* **by** *auto*

lemma *nlists-not-empty*:
assumes $A \neq \{\}$
shows *nlists A n* $\neq \{\}$
proof(*induct n*, *simp add*: *nlists-0*)
  fix *n* **assume** *nlists A n* $\neq \{\}$
  then obtain *a* **and** *l* **where** $a \in A \wedge l \in$ *nlists A n* **using** *assms* **by** *auto*
  hence *a* # *l* $\in$ *nlists A* (*Suc n*) **unfolding** *nlists-def* **by** *auto*
  thus *nlists A* (*Suc n*) $\neq \{\}$ **by** *auto*
qed

lemma *Nil-in-lists*: [] $\in$ *lists A*
unfolding *lists-def2* **by** *auto*

lemma *lists-not-empty*: *lists A* $\neq \{\}$
using *Nil-in-lists* **by** *blast*

lemma *card-of-nlists-Succ*: |*nlists A* (*Suc n*)| =*o* |*A* × (*nlists A n*)|
proof−
  let *?B* = *A* × (*nlists A n*)   **let** *?h* = λ(*a*,*l*). *a* # *l*
  have *inj-on ?h ?B* $\wedge$ *?h* ‘ *?B* $\leq$ *nlists A* (*Suc n*)
  unfolding *inj-on-def nlists-def* **by** *auto*
  moreover have *nlists A* (*Suc n*) $\leq$ *?h* ‘ *?B*
  proof(*auto*)
    fix *l* **assume** $l \in$ *nlists A* (*Suc n*)
    hence *1*: *length l* = *Suc n* $\wedge$ *set l* $\leq$ *A* **unfolding** *nlists-def* **by** *auto*
    then obtain *a* **and** l' **where** *2*: $l = a$ # l' **using** *length-Suc*[*of l*] **by** *auto*
    hence $a \in A \wedge$ *set* $l' \leq A \wedge$ *length* $l' = n$ **using** *1* **by** *auto*
    thus $l \in$ *?h* ‘ *?B* **using** *2* **unfolding** *nlists-def* **by** *auto*
  qed
  ultimately have *bij-betw ?h ?B* (*nlists A* (*Suc n*))
  unfolding *bij-betw-def* **by** *auto*
  thus *?thesis* **using** *card-of-ordIso ordIso-symmetric* **by** *blast*
qed

lemma *card-of-nlists-infinite*:
assumes *infinite A*
shows |*nlists A n*| $\leq$*o* |*A*|
proof(*induct n*)
  have $A \neq \{\}$ **using** *assms* **by** *auto*
  thus |*nlists A 0*| $\leq$*o* |*A*| **by**(*simp add*: *nlists-0*)
next

fix *n* **assume** *IH*: $|nlists\ A\ n| \leq o\ |A|$
have $|nlists\ A\ (Suc\ n)| =o\ |A \times (nlists\ A\ n)|$
using *card-of-nlists-Succ* **by** *blast*
moreover
{have *nlists A n* $\neq$ {} **using** *assms nlists-not-empty*[*of A*] **by** *blast*
hence $|A \times (nlists\ A\ n)| =o\ |A|$
using *assms IH* **by** (*auto simp add*: *card-of-Times-infinite*)
}
ultimately show $|nlists\ A\ (Suc\ n)| \leq o\ |A|$
using *ordIso-transitive ordIso-iff-ordLeq* **by** *blast*
qed

lemma *card-of-lists-infinite*[*simp*]:
assumes *infinite A*
shows $|lists\ A| =o\ |A|$
proof−
have $|lists\ A| \leq o\ |A|$
using *assms*
by (*auto simp add*: *lists-UNION-nlists card-of-UNION-ordLeq-infinite*
infinite-iff-card-of-nat card-of-nlists-infinite)
thus *?thesis* **using** *card-of-lists ordIso-iff-ordLeq* **by** *blast*
qed

lemma *Card-order-lists-infinite*:
assumes *Card-order r* **and** *infinite*(*Field r*)
shows $|lists(Field\ r)| =o\ r$
using *assms card-of-lists-infinite card-of-Field-ordIso ordIso-transitive* **by** *blast*

corollary *lists-infinite-bij-betw*:
assumes *infinite A*
shows $\exists f.$ *bij-betw f* (*lists A*) *A*
using *assms card-of-lists-infinite card-of-ordIso* **by** *blast*

corollary *lists-infinite-bij-betw-types*:
assumes *infinite*(*UNIV* :: *'a set*)
shows $\exists (f::'a\ list \Rightarrow 'a).$ *bij f*
using *assms assms lists-infinite-bij-betw*[*of UNIV*::*'a set*]
using *bij-bij-betw lists-UNIV* **by** *auto*

8.6 Cardinals versus the set-of-finite-sets operator

definition *Fpow* :: *'a set* $\Rightarrow$ *'a set set*
where *Fpow A* $\equiv \{X.\ X \leq A \wedge finite\ X\}$

lemma *Fpow-mono*: $A \leq B \Longrightarrow$ *Fpow* $A \leq$ *Fpow* B
unfolding *Fpow-def* **by** *auto*

lemma *empty-in-Fpow*: $\{\} \in$ *Fpow* A
unfolding *Fpow-def* **by** *auto*

lemma *Fpow-not-empty*: *Fpow* $A \neq \{\}$
using *empty-in-Fpow* **by** *blast*

lemma *Fpow-subset-Pow*: *Fpow* $A \leq$ *Pow* A
unfolding *Fpow-def* **by** *auto*

lemma *card-of-Fpow*[*simp*]: $|A| \leq o$ $|$*Fpow* $A|$
proof$-$
  let *?h* $= \lambda$ $a.$ $\{a\}$
  have *inj-on ?h* $A \wedge$ *?h* ‘ $A \leq$ *Fpow* A
  unfolding *inj-on-def Fpow-def* **by** *auto*
  thus *?thesis* **using** *card-of-ordLeq* **by** *blast*
qed

lemma *Card-order-Fpow*: *Card-order* $r \Longrightarrow r \leq o$ $|$*Fpow*(*Field* r) $|$
using *card-of-Fpow card-of-Field-ordIso ordIso-ordLeq-trans ordIso-symmetric* **by** *blast*

lemma *Fpow-Pow-finite*: *Fpow* $A =$ *Pow* A *Int* $\{A.$ *finite* $A\}$
unfolding *Fpow-def Pow-def* **by** *blast*

lemma *inj-on-image-Fpow*:
assumes *inj-on f A*
shows *inj-on* (*image f*) (*Fpow A*)
using *assms Fpow-subset-Pow*[*of A*] *subset-inj-on*[*of image f Pow A*]
    inj-on-image-Pow **by** *blast*

lemma *image-Fpow-mono*:
assumes f ‘ $A \leq B$
shows (*image f*) ‘ (*Fpow A*) $\leq$ *Fpow* B
using *assms* **by**(*unfold Fpow-def*, *auto*)

lemma *image-Fpow-surjective*:
assumes f ‘ $A = B$

shows (*image f*) ' (*Fpow A*) = *Fpow B*
using *assms* **proof**(*unfold Fpow-def*, *auto*)
  fix *Y* **assume** ∗: *Y* ≤ *f* ' *A* **and** ∗∗: *finite Y*
  hence ∀ *b* ∈ *Y*. ∃ *a*. *a* ∈ *A* ∧ *f a* = *b* **by** *auto*
  with *bchoice*[*of Y λb a. a* ∈ *A* ∧ *f a* = *b*]
  obtain *g* **where** *1*: ∀ *b* ∈ *Y*. *g b* ∈ *A* ∧ *f*(*g b*) = *b* **by** *blast*
  obtain *X* **where** *X-def*: *X* = *g* ' *Y* **by** *blast*
  have *f* ' *X* = *Y* ∧ *X* ≤ *A* ∧ *finite X*
  by(*unfold X-def*, *force simp add*: ∗∗ *1*)
  thus *Y* ∈ (*image f*) ' {*X*. *X* ≤ *A* ∧ *finite X*} **by** *auto*
qed

lemma *bij-betw-image-Fpow*:
assumes *bij-betw f A B*
shows *bij-betw* (*image f*) (*Fpow A*) (*Fpow B*)
using *assms* **unfolding** *bij-betw-def*
by (*auto simp add*: *inj-on-image-Fpow image-Fpow-surjective*)

lemma *card-of-Fpow-mono*[*simp*]:
assumes |*A*| ≤*o* |*B*|
shows |*Fpow A*| ≤*o* |*Fpow B*|
proof−
  obtain *f* **where** *inj-on f A* ∧ *f* ' *A* ≤ *B*
  using *assms card-of-ordLeq*[*of A B*] **by** *auto*
  hence *inj-on* (*image f*) (*Fpow A*) ∧ (*image f*) ' (*Fpow A*) ≤ (*Fpow B*)
  by (*auto simp add*: *inj-on-image-Fpow image-Fpow-mono*)
  thus *?thesis* **using** *card-of-ordLeq*[*of Fpow A*] **by** *auto*
qed

lemma *ordIso-Fpow-mono*:
assumes *r* ≤*o* *r′*
shows |*Fpow*(*Field r*)| ≤*o* |*Fpow*(*Field r′*)|
using *assms card-of-mono2 card-of-Fpow-mono* **by** *blast*

lemma *card-of-Fpow-cong*[*simp*]:
assumes |*A*| =*o* |*B*|
shows |*Fpow A*| =*o* |*Fpow B*|
proof−
  obtain *f* **where** *bij-betw f A B*
  using *assms card-of-ordIso*[*of A B*] **by** *auto*
  hence *bij-betw* (*image f*) (*Fpow A*) (*Fpow B*)
  by (*auto simp add*: *bij-betw-image-Fpow*)
  thus *?thesis* **using** *card-of-ordIso*[*of Fpow A*] **by** *auto*
qed

lemma *ordIso-Fpow-cong*:
assumes $r =o\ r'$
shows $|Fpow(Field\ r)| =o\ |Fpow(Field\ r')|$
using *assms card-of-cong card-of-Fpow-cong* **by** *blast*

lemma *card-of-Fpow-lists*: $|Fpow\ A| \leq o\ |lists\ A|$
proof−
  have *set* ‘ (*lists A*) = *Fpow A*
  unfolding *lists-def2 Fpow-def* **using** *finite-list finite-set* **by** *blast*
  thus *?thesis* **using** *card-of-ordLeq2*[*of Fpow A*] *Fpow-not-empty*[*of A*] **by** *blast*
qed

lemma *card-of-Fpow-infinite*[*simp*]:
assumes *infinite A*
shows $|Fpow\ A| =o\ |A|$
using *assms card-of-Fpow-lists card-of-lists-infinite card-of-Fpow*
    ordLeq-ordIso-trans ordIso-iff-ordLeq **by** *blast*

corollary *Fpow-infinite-bij-betw*:
assumes *infinite A*
shows $\exists f.$ *bij-betw f* (*Fpow A*) *A*
using *assms card-of-Fpow-infinite card-of-ordIso* **by** *blast*

8.7 The cardinal ω and the finite cardinals

The cardinal ω, of natural numbers, shall be the standard non-strict order relation on *nat*, that we abbreviate by *natLeq*. The finite cardinals shall be the restrictions of these relations to the numbers smaller than fixed numbers *n*, that we abbreviate by *natLeq-on n*.

abbreviation (*natLeq*::*nat* $*$ *nat* $\Rightarrow$ *bool*) $\equiv \{(x,y).\ x \leq y\}$
abbreviation (*natLess*::*nat* $*$ *nat* $\Rightarrow$ *bool*) $\equiv \{(x,y).\ x < y\}$

abbreviation *natLeq-on* :: *nat* $\Rightarrow$ (*nat* $*$ *nat* $\Rightarrow$ *bool*)
where *natLeq-on n* $\equiv \{(x,y).\ x < n \wedge y < n \wedge x \leq y\}$

lemma *infinite-cartesian-product*[*simp*]:
assumes *infinite A infinite B*
shows *infinite* $(A \times B)$
proof
  assume *finite* $(A \times B)$
  from *assms*(*1*) **have** $A \neq \{\}$ **by** *auto*
  with ‹*finite* $(A \times B)$› **have** *finite B* **using** *finite-cartesian-productD2* **by** *auto*
  with *assms*(*2*) **show** *False* **by** *simp*
qed

8.7.1 First as well-orders

lemma *Field-natLeq*: *Field natLeq* = (*UNIV*::*nat set*)
by(*unfold Field-def*, *auto*)

lemma *Field-natLess*: *Field natLess* = (*UNIV*::*nat set*)
by(*unfold Field-def*, *auto*)

lemma *natLeq-Refl*: *Refl natLeq*
unfolding *refl-on-def Field-def* **by** *auto*

lemma *natLeq-trans*: *trans natLeq*
unfolding *trans-def* **by** *auto*

lemma *natLeq-Preorder*: *Preorder natLeq*
unfolding *preorder-on-def*
by (*auto simp add*: *natLeq-Refl natLeq-trans*)

lemma *natLeq-antisym*: *antisym natLeq*
unfolding *antisym-def* **by** *auto*

lemma *natLeq-Partial-order*: *Partial-order natLeq*
unfolding *partial-order-on-def*
by (*auto simp add*: *natLeq-Preorder natLeq-antisym*)

lemma *natLeq-Total*: *Total natLeq*
unfolding *total-on-def* **by** *auto*

lemma *natLeq-Linear-order*: *Linear-order natLeq*
unfolding *linear-order-on-def*
by (*auto simp add*: *natLeq-Partial-order natLeq-Total*)

lemma *natLeq-natLess-Id*: *natLess* = *natLeq* − *Id*
by *auto*

lemma *natLeq-Well-order*: *Well-order natLeq*
unfolding *well-order-on-def*
using *natLeq-Linear-order wf-less natLeq-natLess-Id* **by** *auto*

corollary *natLeq-well-order-on*: *well-order-on UNIV natLeq*
using *natLeq-Well-order Field-natLeq* **by** *auto*

lemma *natLeq-wo-rel*: *wo-rel natLeq*
unfolding *wo-rel-def* **using** *natLeq-Well-order* **.**

lemma *natLeq-ofilter-less*: *ofilter natLeq* $\{0 ..< n\}$
by(*auto simp add*: *natLeq-wo-rel wo-rel.ofilter-def*,
  simp add: *Field-natLeq*, *unfold rel.under-def*, *auto*)

lemma *natLeq-ofilter-leq*: *ofilter natLeq* $\{0 .. n\}$
by(*auto simp add*: *natLeq-wo-rel wo-rel.ofilter-def*,
  simp add: *Field-natLeq*, *unfold rel.under-def*, *auto*)

lemma *natLeq-UNIV-ofilter*: *ofilter natLeq UNIV*
using *natLeq-wo-rel Field-natLeq wo-rel.Field-ofilter*[*of natLeq*] **by** *auto*

lemma *closed-nat-set-iff*:
assumes $\forall (m{::}nat)\ n.\ n \in A \wedge m \leq n \longrightarrow m \in A$
shows $A = UNIV \vee (\exists n.\ A = \{0 ..< n\})$
proof−
  {**assume** $A \neq UNIV$ **hence** $\exists n.\ n \notin A$ **by** *blast*
   moreover obtain n **where** *n-def*: $n = (LEAST\ n.\ n \notin A)$ **by** *blast*
   ultimately have *1*: $n \notin A \wedge (\forall m.\ m < n \longrightarrow m \in A)$
   using *LeastI-ex*[*of* $\lambda\ n.\ n \notin A$] *n-def Least-le*[*of* $\lambda\ n.\ n \notin A$] **by** *fastforce*
   have $A = \{0 ..< n\}$
   proof(*auto simp add*: *1*)
     fix m **assume** $*$: $m \in A$
     {**assume** $n \leq m$ **with** *assms* $*$ **have** $n \in A$ **by** *blast*
      hence *False* **using** *1* **by** *auto*
     }
     thus $m < n$ **by** *fastforce*
   qed
   hence $\exists n.\ A = \{0 ..< n\}$ **by** *blast*
  }
  thus *?thesis* **by** *blast*
qed

lemma *natLeq-ofilter-iff*:
ofilter natLeq $A = (A = UNIV \vee (\exists n.\ A = \{0 ..< n\}))$
proof(*rule iffI*)
  assume *ofilter natLeq A*
  hence $\forall m\ n.\ n \in A \wedge m \leq n \longrightarrow m \in A$

by(*auto simp add*: *natLeq-wo-rel wo-rel.ofilter-def rel.under-def*)
  thus $A = UNIV \lor (\exists n.\ A = \{0 ..< n\})$ **using** *closed-nat-set-iff* **by** *blast*
next
  assume $A = UNIV \lor (\exists n.\ A = \{0 ..< n\})$
  thus *ofilter natLeq A*
  by(*auto simp add*: *natLeq-ofilter-less natLeq-UNIV-ofilter*)
qed

lemma *Field-natLeq-on*: *Field* (*natLeq-on n*) = $\{0 ..< n\}$
unfolding *Field-def* **by** *auto*

lemma *natLeq-underS-less*: *underS natLeq n* = $\{0 ..< n\}$
unfolding *rel.underS-def* **by** *auto*

lemma *natLeq-under-leq*: *under natLeq n* = $\{0 .. n\}$
unfolding *rel.under-def* **by** *auto*

lemma *Restr-natLeq*: *Restr natLeq* $\{0 ..< n\}$ = *natLeq-on n*
by *auto*

lemma *Restr-natLeq2*:
Restr natLeq (*underS natLeq n*) = *natLeq-on n*
by (*auto simp add*: *Restr-natLeq natLeq-underS-less*)

lemma *natLeq-on-Well-order*: *Well-order*(*natLeq-on n*)
using *Restr-natLeq*[*of n*] *natLeq-Well-order*
      Well-order-Restr[*of natLeq* $\{0..<n\}$] **by** *auto*

corollary *natLeq-on-well-order-on*: *well-order-on* $\{0 ..< n\}$ (*natLeq-on n*)
using *natLeq-on-Well-order Field-natLeq-on* **by** *auto*

lemma *natLeq-on-wo-rel*: *wo-rel*(*natLeq-on n*)
unfolding *wo-rel-def* **using** *natLeq-on-Well-order* **.**

lemma *natLeq-on-ofilter-less-eq*:
$n \le m \Longrightarrow$ *ofilter*(*natLeq-on m*) $\{0 ..< n\}$
by(*auto simp add*: *natLeq-on-wo-rel wo-rel.ofilter-def*,
   simp add: *Field-natLeq-on*, *unfold rel.under-def*, *auto*)

corollary *natLeq-on-ofilter*:
ofilter(*natLeq-on n*) {*0* ..< *n*}
by (*auto simp add*: *natLeq-on-ofilter-less-eq*)

lemma *natLeq-on-ofilter-less*:
$n < m \Longrightarrow$ *ofilter* (*natLeq-on m*) {*0* .. *n*}
by(*auto simp add*: *natLeq-on-wo-rel wo-rel.ofilter-def*,
  simp add: *Field-natLeq-on*, *unfold rel.under-def*, *auto*)

lemma *natLeq-on-ordLess-natLeq*: *natLeq-on n* <*o natLeq*
using *Field-natLeq Field-natLeq-on*[*of n*] *nat-infinite*
  finite-ordLess-infinite[*of natLeq-on n natLeq*]
  natLeq-Well-order natLeq-on-Well-order[*of n*] **by** *auto*

lemma *natLeq-on-injective*:
natLeq-on m = *natLeq-on n* $\Longrightarrow$ $m = n$
using *Field-natLeq-on*[*of m*] *Field-natLeq-on*[*of n*]
  atLeastLessThan-injective[*of m n*] **by** *auto*

lemma *natLeq-on-injective-ordIso*:
(*natLeq-on m* =*o natLeq-on n*) = ($m = n$)
proof(*auto simp add*: *natLeq-on-Well-order ordIso-reflexive*)
 assume *natLeq-on m* =*o natLeq-on n*
 then obtain *f* **where** *bij-betw f* {*0*..<*m*} {*0*..<*n*}
 using *Field-natLeq-on assms* **unfolding** *ordIso-def iso-def-raw* **by** *auto*
 thus $m = n$ **using** *atLeastLessThan-injective2* **by** *blast*
qed

lemma *natLeq-on-ofilter-iff*:
ofilter (*natLeq-on m*) *A* = ($\exists n \leq m$. *A* = {*0* ..< *n*})
proof(*rule iffI*)
 assume ∗: *ofilter* (*natLeq-on m*) *A*
 hence *1*: $A \leq$ {*0*..<*m*}
 by (*auto simp add*: *natLeq-on-wo-rel wo-rel.ofilter-def rel.under-def Field-natLeq-on*)
 hence $\forall n1\ n2.\ n2 \in A \land n1 \leq n2 \longrightarrow n1 \in A$
 using ∗ **by**(*fastforce simp add*: *natLeq-on-wo-rel wo-rel.ofilter-def rel.under-def*)
 hence *A* = *UNIV* ∨ (∃ *n*. *A* = {*0* ..< *n*}) **using** *closed-nat-set-iff* **by** *blast*
 thus $\exists n \leq m$. *A* = {*0* ..< *n*} **using** *1 atLeastLessThan-less-eq* **by** *blast*
next
 assume ($\exists n \leq m$. *A* = {*0* ..< *n*})
 thus *ofilter* (*natLeq-on m*) *A* **by** (*auto simp add*: *natLeq-on-ofilter-less-eq*)
qed

8.7.2 Then as cardinals

lemma *natLeq-Card-order*: *Card-order natLeq*
proof(*auto simp add*: *natLeq-Well-order*
Card-order-iff-Restr-underS Restr-natLeq2, *simp add*: *Field-natLeq*)
fix *n* **have** *finite*(*Field* (*natLeq-on n*))
unfolding *Field-natLeq-on* **by** *auto*
moreover have *infinite*(*UNIV*::*nat set*) **by** *auto*
ultimately show *natLeq-on n* $<o$ |*UNIV*::*nat set*|
using *finite-ordLess-infinite*[*of natLeq-on n* |*UNIV*::*nat set*|]
Field-card-of[*of UNIV*::*nat set*]
card-of-Well-order[*of UNIV*::*nat set*] *natLeq-on-Well-order*[*of n*] **by** *auto*
qed

corollary *card-of-Field-natLeq*:
|*Field natLeq*| $=o$ *natLeq*
using *Field-natLeq natLeq-Card-order Card-order-iff-ordIso-card-of*[*of natLeq*]
ordIso-symmetric[*of natLeq*] **by** *blast*

corollary *card-of-nat*:
|*UNIV*::*nat set*| $=o$ *natLeq*
using *Field-natLeq card-of-Field-natLeq* **by** *auto*

corollary *infinite-iff-natLeq-ordLeq*:
infinite A = (*natLeq* $\leq o$ |*A*|)
using *infinite-iff-card-of-nat*[*of A*] *card-of-nat*
ordIso-ordLeq-trans ordLeq-ordIso-trans ordIso-symmetric **by** *blast*

lemma *ordIso-natLeq-infinite1*:
|*A*| $=o$ *natLeq* $\Longrightarrow$ *infinite A*
using *ordIso-symmetric ordIso-imp-ordLeq infinite-iff-natLeq-ordLeq* **by** *blast*

lemma *ordIso-natLeq-infinite2*:
natLeq $=o$ |*A*| $\Longrightarrow$ *infinite A*
using *ordIso-imp-ordLeq infinite-iff-natLeq-ordLeq* **by** *blast*

corollary *finite-iff-ordLess-natLeq*:
finite A = (|*A*| $<o$ *natLeq*)
using *infinite-iff-natLeq-ordLeq not-ordLeq-iff-ordLess*
card-of-Well-order natLeq-Well-order **by** *blast*

lemma *ordIso-natLeq-on-imp-finite*:
|*A*| $=o$ *natLeq-on n* $\Longrightarrow$ *finite A*

unfolding *ordIso-def iso-def-raw*
by (*auto simp add*: *Field-natLeq-on bij-betw-finite*)

lemma *natLeq-on-Card-order*: *Card-order* (*natLeq-on n*)
proof(*unfold card-order-on-def*,
  auto simp add: *natLeq-on-Well-order*, *simp add*: *Field-natLeq-on*)
 fix *r* **assume** *well-order-on* $\{0..<n\}$ *r*
 thus *natLeq-on n* $\leq o$ *r*
 using *finite-atLeastLessThan natLeq-on-well-order-on*
  finite-well-order-on-ordIso ordIso-iff-ordLeq **by** *blast*
qed

corollary *card-of-Field-natLeq-on*:
$|$*Field* (*natLeq-on n*)$|$ $=o$ *natLeq-on n*
using *Field-natLeq-on natLeq-on-Card-order*
  Card-order-iff-ordIso-card-of[*of natLeq-on n*]
  ordIso-symmetric[*of natLeq-on n*] **by** *blast*

corollary *card-of-less*:
$|\{0 ..< n\}|$ $=o$ *natLeq-on n*
using *Field-natLeq-on card-of-Field-natLeq-on* **by** *auto*

lemma *ordLeq-natLeq-on-imp-finite*:
assumes $|A|$ $\leq o$ *natLeq-on n*
shows *finite A*
proof−
 have $|A|$ $\leq o$ $|\{0 ..< n\}|$
 using *assms card-of-less ordIso-symmetric ordLeq-ordIso-trans* **by** *blast*
 thus *?thesis* **by** (*auto simp add*: *card-of-ordLeq-finite*)
qed

lemma *natLeq-on-ordLeq-less-eq*:
((*natLeq-on m*) $\leq o$ (*natLeq-on n*)) = ($m \leq n$)
proof
 assume *natLeq-on m* $\leq o$ *natLeq-on n*
 then obtain *f* **where** *inj-on f* $\{0..<m\} \wedge f$ ‘ $\{0..<m\} \leq \{0..<n\}$
 using *Field-natLeq-on*[*of m*] *Field-natLeq-on*[*of n*]
 unfolding *ordLeq-def* **using** *embed-inj-on*[*of natLeq-on m natLeq-on n*]
 embed-Field[*of natLeq-on m natLeq-on n*] **using** *natLeq-on-Well-order*[*of m*] **by**
fastforce
 thus $m \leq n$ **using** *atLeastLessThan-less-eq2* **by** *blast*
next
 assume $m \leq n$
 hence *inj-on id* $\{0..<m\} \wedge id$ ‘ $\{0..<m\} \leq \{0..<n\}$ **unfolding** *inj-on-def* **by**

auto
hence $|\{0..<m\}| \leq o\ |\{0..<n\}|$ **using** *card-of-ordLeq* **by** *blast*
thus *natLeq-on m* $\leq o$ *natLeq-on n*
using *card-of-less ordIso-ordLeq-trans ordLeq-ordIso-trans ordIso-symmetric* **by** *blast*
qed

lemma *natLeq-on-ordLeq-less*:
$((natLeq\text{-}on\ m) <o\ (natLeq\text{-}on\ n)) = (m < n)$
using *not-ordLeq-iff-ordLess*[*of natLeq-on m natLeq-on n*]
natLeq-on-Well-order natLeq-on-ordLeq-less-eq **by** *auto*

8.7.3 "Backwards compatibility" with the numeric cardinal operator for finite sets

lemma *finite-card-of-iff-card*:
assumes *FIN*: *finite A* **and** *FIN'*: *finite B*
shows ($|A| =o\ |B|$) = (*card A* = *card B*)
using *assms card-of-ordIso*[*of A B*] *bij-betw-iff-card*[*of A B*] **by** *blast*

lemma *finite-card-of-iff-card2*:
assumes *FIN*: *finite A* **and** *FIN'*: *finite B*
shows ($|A| \leq o\ |B|$) = (*card A* $\leq$ *card B*)
using *assms card-of-ordLeq*[*of A B*] *inj-on-iff-card*[*of A B*] **by** *blast*

lemma *finite-card-of-iff-card3*:
assumes *FIN*: *finite A* **and** *FIN'*: *finite B*
shows ($|A| <o\ |B|$) = (*card A* < *card B*)
proof−
have ($|A| <o\ |B|$) = (~ ($|B| \leq o\ |A|$)) **by** *simp*
also have ... = (~ (*card B* $\leq$ *card A*))
using *assms* **by**(*simp add*: *finite-card-of-iff-card2*)
also have ... = (*card A* < *card B*) **by** *auto*
finally show *?thesis* **.**
qed

lemma *finite-imp-card-of-natLeq-on*:
assumes *finite A*
shows $|A| =o$ *natLeq-on* (*card A*)
proof−
obtain *h* **where** *bij-betw h A* {*0* ..< *card A*}
using *assms ex-bij-betw-finite-nat* **by** *blast*
thus *?thesis* **using** *card-of-ordIso card-of-less ordIso-equivalence* **by** *blast*
qed

lemma *finite-iff-card-of-natLeq-on*:
finite $A = (\exists\, n.\ |A| =o\ natLeq\text{-}on\ n)$
using *finite-imp-card-of-natLeq-on*[*of A*]
by(*auto simp add*: *ordIso-natLeq-on-imp-finite*)

lemma *card-Field-natLeq-on*:
$card(Field(natLeq\text{-}on\ n)) = n$
using *Field-natLeq-on card-atLeastLessThan* **by** *auto*

8.8 The successor of a cardinal

First we define *isCardSuc* $r\ r'$, the notion of r' being a successor cardinal of r. Although the definition does not require r to be a cardinal, only this case will be meaningful.

definition *isCardSuc* :: $'a\ rel \Rightarrow\ 'a\ set\ rel \Rightarrow bool$
where
$isCardSuc\ r\ r' \equiv$
$Card\text{-}order\ r' \wedge r <o\ r' \wedge$
$(\forall\, (r''::'a\ set\ rel).\ Card\text{-}order\ r'' \wedge r <o\ r'' \longrightarrow r' \leq o\ r'')$

Now we introduce the cardinal-successor operator *cardSuc*, by picking *some* cardinal-order relation fulfilling *isCardSuc*. Again, the picked item shall be proved unique up to order-isomorphism.

definition *cardSuc* :: $'a\ rel \Rightarrow\ 'a\ set\ rel$
where
$cardSuc\ r \equiv SOME\ r'.\ isCardSuc\ r\ r'$

lemma *exists-minim-Card-order*:
$[\![R \neq \{\}; \forall\, r \in R.\ Card\text{-}order\ r]\!] \Longrightarrow \exists\, r \in R.\ \forall\, r' \in R.\ r \leq o\ r'$
unfolding *card-order-on-def* **using** *exists-minim-Well-order* **by** *blast*

lemma *exists-isCardSuc*:
assumes *Card-order r*
shows $\exists\, r'.\ isCardSuc\ r\ r'$
proof−
  let $?R = \{(r'::'a\ set\ rel).\ Card\text{-}order\ r' \wedge r <o\ r'\}$
  have $|Pow(Field\ r)| \in\ ?R \wedge (\forall\, r \in\ ?R.\ Card\text{-}order\ r)$ **using** *assms* **by** *simp*
  then obtain r **where** $r \in\ ?R \wedge (\forall\, r' \in\ ?R.\ r \leq o\ r')$
  using *exists-minim-Card-order*[*of ?R*] **by** *blast*
  thus *?thesis* **unfolding** *isCardSuc-def* **by** *auto*
qed

lemma *cardSuc-isCardSuc*:

assumes *Card-order r*
shows *isCardSuc r* (*cardSuc r*)
unfolding *cardSuc-def* **using** *assms*
by (*auto simp add*: *exists-isCardSuc someI-ex*)

lemma *cardSuc-Card-order*[*simp*]:
Card-order r $\Longrightarrow$ *Card-order*(*cardSuc r*)
using *cardSuc-isCardSuc* **unfolding** *isCardSuc-def* **by** *blast*

lemma *cardSuc-Well-order*[*simp*]:
Card-order r $\Longrightarrow$ *Well-order*(*cardSuc r*)
using *cardSuc-Card-order* **unfolding** *card-order-on-def* **by** *blast*

lemma *cardSuc-greater*[*simp*]:
Card-order r $\Longrightarrow$ *r <o cardSuc r*
using *cardSuc-isCardSuc* **unfolding** *isCardSuc-def* **by** *blast*

lemma *cardSuc-ordLeq*[*simp*]:
Card-order r $\Longrightarrow$ *r* $\leq$*o cardSuc r*
using *cardSuc-greater ordLeq-iff-ordLess-or-ordIso* **by** *blast*

The minimality property of *cardSuc* originally present in its definition is local to the type *'a set rel*, i.e., that of *cardSuc r*:

lemma *cardSuc-least-aux*:
$[\![$*Card-order* (*r*::*'a rel*); *Card-order* (*r'*::*'a set rel*); *r <o r'*$]\!]$ $\Longrightarrow$ *cardSuc r* $\leq$*o r'*
using *cardSuc-isCardSuc* **unfolding** *isCardSuc-def* **by** *blast*

But from this we can infer general minimality:

lemma *cardSuc-least*:
assumes *CARD*: *Card-order r* **and** *CARD'*: *Card-order r'* **and** *LESS*: *r <o r'*
shows *cardSuc r* $\leq$*o r'*
proof−
  let *?p* = *cardSuc r*
  have *0*: *Well-order ?p* $\wedge$ *Well-order r'*
  using *assms cardSuc-Card-order* **unfolding** *card-order-on-def* **by** *blast*
  {**assume** *r' <o ?p*
   then obtain *r''* **where** *1*: *Field r''* < *Field ?p* **and** *2*: *r'* =*o r''* $\wedge$ *r'' <o ?p*
   using *internalize-ordLess*[*of r' ?p*] **by** *blast*

   have *Card-order r''* **using** *CARD' Card-order-ordIso2 2* **by** *blast*
   moreover have *r <o r''* **using** *LESS 2 ordLess-ordIso-trans* **by** *blast*
   ultimately have *?p* $\leq$*o r''* **using** *cardSuc-least-aux CARD* **by** *blast*
   hence *False* **using** *2 not-ordLess-ordLeq* **by** *blast*
  }
  thus *?thesis* **using** *0 ordLess-or-ordLeq* **by** *blast*

qed

lemma *Field-cardSuc-not-empty*:
assumes *Card-order r*
shows *Field* (*cardSuc r*) $\neq$ {}
proof
  assume *Field*(*cardSuc r*) = {}
  hence |*Field*(*cardSuc r*)| $\leq$*o r* **using** *assms Card-order-empty*[*of r*] **by** *auto*
  hence *cardSuc r* $\leq$*o r* **using** *assms card-of-Field-ordIso*
  cardSuc-Card-order ordIso-symmetric ordIso-ordLeq-trans **by** *blast*
  thus *False* **using** *cardSuc-greater not-ordLess-ordLeq assms* **by** *blast*
qed

lemma *cardSuc-ordLess-ordLeq*:
assumes *CARD*: *Card-order r* **and** *CARD'*: *Card-order r'*
shows (*r* <*o r'*) = (*cardSuc r* $\leq$*o r'*)
proof(*auto simp add*: *assms cardSuc-least*)
  assume *cardSuc r* $\leq$*o r'*
  thus *r* <*o r'* **using** *assms cardSuc-greater ordLess-ordLeq-trans* **by** *blast*
qed

lemma *cardSuc-ordLeq-ordLess*[*simp*]:
assumes *CARD*: *Card-order r* **and** *CARD'*: *Card-order r'*
shows (*r'* <*o cardSuc r*) = (*r'* $\leq$*o r*)
proof−
  have *Well-order r* $\wedge$ *Well-order r'*
  using *assms* **unfolding** *card-order-on-def* **by** *auto*
  moreover have *Well-order*(*cardSuc r*)
  using *assms cardSuc-Card-order card-order-on-def* **by** *blast*
  ultimately show *?thesis*
  using *assms cardSuc-ordLess-ordLeq*[*of r r'*]
  not-ordLeq-iff-ordLess[*of r r'*] *not-ordLeq-iff-ordLess*[*of r' cardSuc r*] **by** *blast*
qed

lemma *cardSuc-mono-ordLeq*[*simp*]:
assumes *CARD*: *Card-order r* **and** *CARD'*: *Card-order r'*
shows (*cardSuc r* $\leq$*o cardSuc r'*) = (*r* $\leq$*o r'*)
using *assms cardSuc-ordLeq-ordLess cardSuc-ordLess-ordLeq cardSuc-Card-order*
by *blast*

lemma *cardSuc-mono-ordLess*[*simp*]:
assumes *CARD*: *Card-order r* **and** *CARD'*: *Card-order r'*
shows (*cardSuc r* <*o cardSuc r'*) = (*r* <*o r'*)
proof−

have *0*: *Well-order r* $\wedge$ *Well-order r'* $\wedge$ *Well-order*(*cardSuc r*) $\wedge$ *Well-order*(*cardSuc r'*)
using *assms* **by** *auto*
thus *?thesis*
using *not-ordLeq-iff-ordLess not-ordLeq-iff-ordLess*[*of r r'*]
using *cardSuc-mono-ordLeq*[*of r' r*] *assms* **by** *blast*
qed

lemma *cardSuc-invar-ordIso*[*simp*]:
assumes *CARD*: *Card-order r* **and** *CARD'*: *Card-order r'*
shows (*cardSuc r =o cardSuc r'*) = (*r =o r'*)
proof−
have *0*: *Well-order r* $\wedge$ *Well-order r'* $\wedge$ *Well-order*(*cardSuc r*) $\wedge$ *Well-order*(*cardSuc r'*)
using *assms* **by** *auto*
thus *?thesis*
using *ordIso-iff-ordLeq*[*of r r'*] *ordIso-iff-ordLeq*
using *cardSuc-mono-ordLeq*[*of r r'*] *cardSuc-mono-ordLeq*[*of r' r*] *assms* **by** *blast*
qed

lemma *embed-implies-ordIso-Restr*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and** *EMB*: *embed r' r f*
shows *r' =o Restr r* (*f ‘* (*Field r'*))
using *assms embed-implies-iso-Restr Well-order-Restr* **unfolding** *ordIso-def* **by** *blast*

lemma *cardSuc-natLeq-on-Suc*:
cardSuc(*natLeq-on n*) *=o natLeq-on*(*Suc n*)
proof−
obtain *r r' p* **where** *r-def*: *r* = *natLeq-on n* **and**
r'-def: *r'* = *cardSuc*(*natLeq-on n*) **and**
p-def: *p* = *natLeq-on*(*Suc n*) **by** *blast*

have *CARD*: *Card-order r* $\wedge$ *Card-order r'* $\wedge$ *Card-order p* **unfolding** *r-def r'-def p-def*
using *cardSuc-ordLess-ordLeq natLeq-on-Card-order cardSuc-Card-order* **by** *blast*
hence *WELL*: *Well-order r* $\wedge$ *Well-order r'* $\wedge$ *Well-order p*
unfolding *card-order-on-def* **by** *force*
have *FIELD*: *Field r* = $\{0..<n\}$ $\wedge$ *Field p* = $\{0..<(Suc\ n)\}$
unfolding *r-def p-def Field-natLeq-on* **by** *simp*
hence *FIN*: *finite* (*Field r*) **by** *force*
have *r <o r'* **using** *CARD* **unfolding** *r-def r'-def* **using** *cardSuc-greater* **by** *blast*
hence |*Field r*| *<o r'* **using** *CARD card-of-Field-ordIso ordIso-ordLess-trans* **by** *blast*

hence *LESS*: $|Field\ r| <o\ |Field\ r'|$
using *CARD card-of-Field-ordIso ordLess-ordIso-trans ordIso-symmetric* **by** *blast*

have $r' \leq o\ p$ **using** *CARD* **unfolding** *r-def r'-def p-def*
using *natLeq-on-ordLeq-less cardSuc-ordLess-ordLeq* **by** *blast*
moreover have $p \leq o\ r'$
proof−
{**assume** $r' <o\ p$
then obtain f **where** *0*: *embedS r' p f* **unfolding** *ordLess-def* **by** *force*
let *?q* = *Restr p* (*f* ' *Field r'*)
have *1*: *embed r' p f* **using** *0* **unfolding** *embedS-def* **by** *force*
hence *2*: *f* ' *Field r'* < {*0*..<(*Suc n*)}
using *WELL FIELD 0* **by** (*auto simp add*: *embedS-iff*)
have *ofilter p* (*f* ' *Field r'*) **using** *embed-Field-ofilter 1 WELL* **by** *blast*
then obtain m **where** $m \leq$ *Suc n* **and** *3*: *f* ' (*Field r'*) = {*0*..<*m*}
unfolding *p-def* **by** (*auto simp add*: *natLeq-on-ofilter-iff*)
hence *4*: $m \leq n$ **using** *2* **by** *force*

have *bij-betw f* (*Field r'*) (*f* ' (*Field r'*))
using *1 WELL embed-inj-on* **unfolding** *bij-betw-def* **by** *force*
moreover have *finite*(*f* ' (*Field r'*)) **using** *3 finite-atLeastLessThan*[*of 0 m*]
by *force*
ultimately have *5*: *finite* (*Field r'*) ∧ *card*(*Field r'*) = *card* (*f* ' (*Field r'*))
using *bij-betw-imp-card bij-betw-finite* **by** *blast*
hence *card*(*Field r'*) ≤ *card*(*Field r*) **using** *3 4 FIELD* **by** *force*
hence $|Field\ r'| \leq o\ |Field\ r|$ **using** *FIN 5 finite-card-of-iff-card2* **by** *blast*
hence *False* **using** *LESS not-ordLess-ordLeq* **by** *auto*
}
thus *?thesis* **using** *WELL CARD* **by** *fastforce*
qed
ultimately show *?thesis* **using** *ordIso-iff-ordLeq* **unfolding** *r'-def p-def* **by**
blast
qed

lemma *card-of-cardSuc-finite*[*simp*]:
finite(*Field*(*cardSuc* |*A*|)) = *finite A*
proof
assume ∗: *finite* (*Field* (*cardSuc* |*A*|))
have *0*: |*Field*(*cardSuc* |*A*|)| =*o cardSuc* |*A*|
using *card-of-Card-order cardSuc-Card-order card-of-Field-ordIso* **by** *blast*
hence |*A*| ≤*o* |*Field*(*cardSuc* |*A*|)|
using *card-of-Card-order*[*of A*] *cardSuc-ordLeq*[*of* |*A*|] *ordIso-symmetric*
ordLeq-ordIso-trans **by** *blast*
thus *finite A* **using** ∗ *card-of-ordLeq-finite* **by** *blast*
next
assume *finite A*
then obtain n **where** |*A*| =*o natLeq-on n* **using** *finite-iff-card-of-natLeq-on* **by**
blast

hence *cardSuc* $|A|$ $=o$ *cardSuc*(*natLeq-on* n)
using *card-of-Card-order cardSuc-invar-ordIso natLeq-on-Card-order* **by** *blast*
hence *cardSuc* $|A|$ $=o$ *natLeq-on*(*Suc* n)
using *cardSuc-natLeq-on-Suc ordIso-transitive* **by** *blast*
hence *cardSuc* $|A|$ $=o$ $|\{0..<(Suc\ n)\}|$ **using** *card-of-less ordIso-equivalence* **by** *blast*
moreover have $|Field\ (cardSuc\ |A|\ )\ |$ $=o$ *cardSuc* $|A|$
using *card-of-Field-ordIso cardSuc-Card-order card-of-Card-order* **by** *blast*
ultimately have $|Field\ (cardSuc\ |A|\ )\ |$ $=o$ $|\{0..<(Suc\ n)\}|$
using *ordIso-equivalence* **by** *blast*
thus *finite* (*Field* (*cardSuc* $|A|$))
using *card-of-ordIso-finite finite-atLeastLessThan* **by** *blast*
qed

lemma *cardSuc-finite*[*simp*]:
assumes *Card-order r*
shows *finite* (*Field* (*cardSuc r*)) = *finite* (*Field r*)
proof−
let *?A* = *Field r*
have $|?A|$ $=o$ r **using** *assms* **by** *simp*
hence *cardSuc* $|?A|$ $=o$ *cardSuc r* **using** *assms* **by** *simp*
moreover have $|Field\ (cardSuc\ |?A|\ )\ |$ $=o$ *cardSuc* $|?A|$
using *card-of-Field-ordIso* **by** *simp*
moreover
{**have** $|Field\ (cardSuc\ r)\ |$ $=o$ *cardSuc r*
using *assms card-of-Field-ordIso* **by** *simp*
hence *cardSuc r* $=o$ $|Field\ (cardSuc\ r)\ |$
using *ordIso-symmetric* **by** *blast*
}
ultimately have $|Field\ (cardSuc\ |?A|\ )\ |$ $=o$ $|Field\ (cardSuc\ r)\ |$
using *ordIso-transitive* **by** *blast*
hence *finite* (*Field* (*cardSuc* $|?A|$)) = *finite* (*Field* (*cardSuc r*))
using *card-of-ordIso-finite* **by** *blast*
thus *?thesis* **by** *simp*
qed

lemma *card-of-Plus-ordLeq-infinite*[*simp*]:
assumes *C*: *infinite C* **and** *A*: $|A|$ $\leq o$ $|C|$ **and** *B*: $|B|$ $\leq o$ $|C|$
shows $|A <+> B|$ $\leq o$ $|C|$
proof−
let *?r* = *cardSuc* $|C|$
have *Card-order ?r* $\wedge$ *infinite* (*Field ?r*) **using** *assms* **by** *simp*
moreover have $|A|$ $<o$ *?r* **and** $|B|$ $<o$ *?r* **using** *A B* **by** *auto*
ultimately have $|A <+> B|$ $<o$ *?r*
using *card-of-Plus-ordLess-infinite-Field* **by** *blast*
thus *?thesis* **using** *C* **by** *simp*
qed

lemma *card-of-Plus-ordLeq-infinite-Field*[*simp*]:
assumes *r*: *infinite* (*Field r*) **and** *A*: $|A| \le o\ r$ **and** *B*: $|B| \le o\ r$
and *c*: *Card-order r*
shows $|A <+> B| \le o\ r$
proof−
  let *?r′* = *cardSuc r*
  have *Card-order ?r′* $\wedge$ *infinite* (*Field ?r′*) **using** *assms* **by** *simp*
  moreover have $|A| < o$ *?r′* **and** $|B| < o$ *?r′* **using** *A B c* **by** *auto*
  ultimately have $|A <+> B| < o$ *?r′*
  using *card-of-Plus-ordLess-infinite-Field* **by** *blast*
  thus *?thesis* **using** *c r* **by** *simp*
qed

lemma *card-of-Un-ordLeq-infinite*[*simp*]:
assumes *C*: *infinite C* **and** *A*: $|A| \le o\ |C|$ **and** *B*: $|B| \le o\ |C|$
shows $|A\ Un\ B| \le o\ |C|$
using *assms card-of-Plus-ordLeq-infinite card-of-Un-Plus-ordLeq ordLeq-transitive* **by** *blast*

lemma *card-of-Un-ordLeq-infinite-Field*[*simp*]:
assumes *C*: *infinite* (*Field r*) **and** *A*: $|A| \le o\ r$ **and** *B*: $|B| \le o\ r$
and *Card-order r*
shows $|A\ Un\ B| \le o\ r$
using *assms card-of-Plus-ordLeq-infinite-Field card-of-Un-Plus-ordLeq ordLeq-transitive* **by** *blast*

8.9 Regular cardinals

definition *cofinal* **where**
cofinal A r $\equiv$
 $ALL\ a : Field\ r.\ EX\ b : A.\ a \neq b \wedge (a,b) : r$

definition *regular* **where**
regular r $\equiv$
 $ALL\ K.\ K \le Field\ r \wedge cofinal\ K\ r \longrightarrow |K| =o\ r$

definition *relChain* **where**
relChain r As $\equiv$
 $ALL\ i\ j.\ (i,j) \in r \longrightarrow As\ i \le As\ j$

lemma *regular-UNION*:
assumes *r*: *Card-order r*   *regular r*
and *As*: *relChain r As*

and *Bsub*: $B \leq (UN\ i : Field\ r.\ As\ i)$
and *cardB*: $|B| <o\ r$
shows $EX\ i : Field\ r.\ B \leq As\ i$
proof−
  let $?phi = \%b\ j.\ j : Field\ r \wedge b : As\ j$
  have $ALL\ b : B.\ EX\ j.\ ?phi\ b\ j$ **using** *Bsub* **by** *blast*
  then obtain f **where** f: !! $b.\ b : B \Longrightarrow ?phi\ b\ (f\ b)$
  using *bchoice*[*of B ?phi*] **by** *blast*
  let $?K = f\ `\ B$
  {**assume** *1*: !! $i.\ i : Field\ r \Longrightarrow \sim\ B \leq As\ i$
   have *2*: *cofinal ?K r*
   unfolding *cofinal-def* **proof** *auto*
     fix i **assume** i: $i : Field\ r$
     with *1* **obtain** b **where** b: $b : B \wedge b \notin As\ i$ **by** *blast*
     hence $i \neq f\ b \wedge \sim\ (f\ b,i) : r$
     using *As f* **unfolding** *relChain-def* **by** *auto*
     hence $i \neq f\ b \wedge (i,\ f\ b) : r$ **using** r
     unfolding *card-order-on-def well-order-on-def linear-order-on-def*
     total-on-def **using** *i f b* **by** *auto*
     with b **show** $\exists\ b{\in}B.\ i \neq f\ b \wedge (i,\ f\ b) \in r$ **by** *blast*
   qed
   moreover have $?K \leq Field\ r$ **using** f **by** *blast*
   ultimately have $|?K| =o\ r$ **using** *2 r* **unfolding** *regular-def* **by** *blast*
   moreover
   {
    have $|?K| <=o\ |B|$ **using** *card-of-image* **.**
    hence $|?K| <o\ r$ **using** *cardB ordLeq-ordLess-trans* **by** *blast*
   }
   ultimately have *False* **using** *not-ordLess-ordIso* **by** *blast*
  }
  thus *?thesis* **by** *blast*
qed

lemma *infinite-cardSuc-regular*:
assumes *r-inf*: *infinite* (*Field r*) **and** *r-card*: *Card-order r*
shows *regular* (*cardSuc r*)
proof−
  let $?r' = cardSuc\ r$
  have r': *Card-order* $?r'$
  !! $p.\ Card\text{-}order\ p \longrightarrow (p \leq o\ r) = (p <o\ ?r')$
  using *r-card* **by** *auto*
  show *?thesis*
  unfolding *regular-def* **proof** *auto*
    fix K **assume** *1*: $K \leq Field\ ?r'$ **and** *2*: *cofinal* $K\ ?r'$
    hence $|K| \leq o\ |Field\ ?r'|$ **by** *simp*
    also have *22*: $|Field\ ?r'| =o\ ?r'$
    using r' **by** (*simp add*: *card-of-Field-ordIso*[*of* $?r'$])
    finally have $|K| \leq o\ ?r'$ **.**

moreover
{**let** *?L* = *UN j* : *K*. *underS ?r′ j*
let *?J* = *Field r*
have *rJ*: *r* =*o* |*?J*|
using *r-card card-of-Field-ordIso ordIso-symmetric* **by** *blast*
assume |*K*| <*o ?r′*
hence |*K*| <=*o r* **using** *r′ card-of-Card-order*[*of K*] **by** *blast*
hence |*K*| ≤*o* |*?J*| **using** *rJ ordLeq-ordIso-trans* **by** *blast*
moreover
{**have** *ALL j* : *K*. |*underS ?r′ j*| <*o ?r′*
using *r′ 1* **by** *auto*
hence *ALL j* : *K*. |*underS ?r′ j*| ≤*o r*
using *r′ card-of-Card-order* **by** *blast*
hence *ALL j* : *K*. |*underS ?r′ j*| ≤*o* |*?J*|
using *rJ ordLeq-ordIso-trans* **by** *blast*
}
ultimately have |*?L*| ≤*o* |*?J*|
using *r-inf card-of-UNION-ordLeq-infinite* **by** *blast*
hence |*?L*| ≤*o r* **using** *rJ ordIso-symmetric ordLeq-ordIso-trans* **by** *blast*
hence |*?L*| <*o ?r′* **using** *r′ card-of-Card-order* **by** *blast*
moreover
{
have *Field ?r′* ≤ *?L*
using *2* **unfolding** *rel.underS-def cofinal-def* **by** *auto*
hence |*Field ?r′*| ≤*o* |*?L*| **by** *simp*
hence *?r′* ≤*o* |*?L*|
using *22 ordIso-ordLeq-trans ordIso-symmetric* **by** *blast*
}
ultimately have |*?L*| <*o* |*?L*| **using** *ordLess-ordLeq-trans* **by** *blast*
hence *False* **using** *ordLess-irreflexive* **by** *blast*
}
ultimately show |*K*| =*o ?r′*
unfolding *ordLeq-iff-ordLess-or-ordIso* **by** *blast*
qed
qed

lemma *cardSuc-UNION*:
assumes *r*: *Card-order r* **and** *infinite* (*Field r*)
and *As*: *relChain* (*cardSuc r*) *As*
and *Bsub*: *B* ≤ (*UN i* : *Field* (*cardSuc r*). *As i*)
and *cardB*: |*B*| <=*o r*
shows *EX i* : *Field* (*cardSuc r*). *B* ≤ *As i*
proof−
let *?r′* = *cardSuc r*
have *Card-order ?r′* ∧ |*B*| <*o ?r′*
using *r cardB cardSuc-ordLeq-ordLess cardSuc-Card-order*
card-of-Card-order **by** *blast*
moreover have *regular ?r′*
using *assms* **by**(*simp add*: *infinite-cardSuc-regular*)

ultimately show *?thesis*
using *As Bsub cardB regular-UNION* **by** *blast*
qed

8.10 Others

lemma *under-mono*[*simp*]:
assumes *Well-order r* **and** $(i,j) \in r$
shows *under r i* $\subseteq$ *under r j*
using *assms* **unfolding** *rel.under-def order-on-defs*
trans-def **by** *blast*

lemma *underS-under*:
assumes $i \in$ *Field r*
shows *underS r i* = *under r i* $-$ $\{i\}$
using *assms* **unfolding** *rel.underS-def rel.under-def* **by** *auto*

lemma *relChain-under*:
assumes *Well-order r*
shows *relChain r* (λ *i. under r i*)
using *assms* **unfolding** *relChain-def* **by** *auto*

lemma *card-of-infinite-diff-finitte*:
assumes *infinite A* **and** *finite B*
shows $|A - B|$ =*o* $|A|$
by (*metis assms card-of-Un-diff-infinite finite-ordLess-infinite2*)

definition *Bpow* **where**
Bpow r A $\equiv$ $\{X \; . \; X \subseteq A \wedge |X| \leq$*o r*$\}$

lemma *Bpow-empty*[*simp*]:
assumes *Card-order r*
shows *Bpow r* $\{\}$ = $\{\{\}\}$
using *assms* **unfolding** *Bpow-def* **by** *auto*

lemma *singl-in-Bpow*:
assumes *rc*: *Card-order r*
and *r*: *Field r* $\neq \{\}$ **and** *a*: $a \in A$
shows $\{a\} \in$ *Bpow r A*
proof−
  have $|\{a\}| \leq$*o r* **using** *r rc* **by** *auto*
  thus *?thesis* **unfolding** *Bpow-def* **using** *a* **by** *auto*
qed

lemma *ordLeq-card-Bpow*:
assumes *rc*: *Card-order r* **and** *r*: *Field r* $\neq \{\}$
shows $|A| \leq$*o* |*Bpow r A*|
proof−

have *inj-on* (λ *a*. {*a*}) *A* **unfolding** *inj-on-def* **by** *auto*
moreover have (λ *a*. {*a*}) ' *A* $\subseteq$ *Bpow r A*
using *singl-in-Bpow*[*OF assms*] **by** *auto*
ultimately show *?thesis* **unfolding** *card-of-ordLeq*[*symmetric*] **by** *blast*
qed

lemma *infinite-Bpow*:
assumes *rc*: *Card-order r* **and** *r*: *Field r* $\neq$ {}
and *A*: *infinite A*
shows *infinite* (*Bpow r A*)
using *ordLeq-card-Bpow*[*OF rc r*]
by (*metis A card-of-ordLeq-infinite*)

definition *Func* **where**
Func A B $\equiv$
{*f*. ($\forall$ *a*. *f a* $\neq$ *None* $\longleftrightarrow$ *a* $\in$ *A*) $\wedge$ ($\forall$ *a* $\in$ *A*. *case f a of Some b* $\Rightarrow$ *b* $\in$ *B* |*None* $\Rightarrow$ *True*)}

lemma *Func-empty*[*simp*]:
Func {} *B* = {*empty*}
unfolding *Func-def* **by** *auto*

lemma *Func-elim*:
assumes *g* $\in$ *Func A B* **and** *a* $\in$ *A*
shows $\exists$ *b*. *b* $\in$ *B* $\wedge$ *g a* = *Some b*
using *assms* **unfolding** *Func-def* **by** (*cases g a*) *force+*

lemma *Bpow-ordLeq-Func-Field*:
assumes *rc*: *Card-order r* **and** *r*: *Field r* $\neq$ {} **and** *A*: *infinite A*
shows |*Bpow r A*| $\leq$*o* |*Func* (*Field r*) *A*|
proof−
let *?F* = λ *f*. {*x* | *x a*. *f a* = *Some x*}
{**fix** *X* **assume** *X* $\in$ *Bpow r A* − {{}}
hence *XA*: *X* $\subseteq$ *A* **and** |*X*| $\leq$*o r*
and *X*: *X* $\neq$ {} **unfolding** *Bpow-def* **by** *auto*
hence |*X*| $\leq$*o* |*Field r*| **by** (*metis Field-card-of card-of-mono2*)
then obtain *F* **where** *1*: *X* = *F* ' (*Field r*)
using *card-of-ordLeq2*[*OF X*] **by** *metis*
def *f* $\equiv$ λ *i*. *if i* $\in$ *Field r then Some* (*F i*) *else None*
have $\exists$ *f* $\in$ *Func* (*Field r*) *A*. *X* = *?F f*
apply (*intro bexI*[*of - f*]) **using** *1 XA* **unfolding** *Func-def f-def* **by** *auto*
}
hence *Bpow r A* − {{}} $\subseteq$ *?F* ' (*Func* (*Field r*) *A*) **by** *auto*
hence |*Bpow r A* − {{}}| $\leq$*o* |*Func* (*Field r*) *A*|
by (*rule surj-imp-ordLeq*)
moreover
{**have** *2*: *infinite* (*Bpow r A*) **using** *infinite-Bpow*[*OF rc r A*] **.**
have |*Bpow r A*| =*o* |*Bpow r A* − {{}}|

```
   using card-of-infinite-diff-finitte
   by (metis Pow-empty 2 finite-Pow-iff infinite-imp-nonempty ordIso-symmetric)
  }
  ultimately show ?thesis by (metis ordIso-ordLeq-trans)
qed

definition curr where
curr A f ≡ λ a. if a ∈ A then Some (λ b. f (a,b)) else None

lemma curr-in[intro, simp]:
assumes f: f ∈ Func (A <*> B) C
shows curr A f ∈ Func A (Func B C)
using assms unfolding curr-def Func-def by auto

lemma curr-inj:
assumes f1 ∈ Func (A <*> B) C and f2 ∈ Func (A <*> B) C
shows curr A f1 = curr A f2 ⟷ f1 = f2
proof safe
  assume c: curr A f1 = curr A f2
  show f1 = f2
  proof (clarify intro!: ext)
    fix a b show f1 (a, b) = f2 (a, b)
    proof (cases (a,b) ∈ A <*> B)
      case False
      thus ?thesis using assms unfolding Func-def
      apply(cases f1 (a,b)) apply(cases f2 (a,b), fastforce, fastforce)
      apply(cases f2 (a,b)) by auto
    next
      case True hence a: a ∈ A and b: b ∈ B by auto
      thus ?thesis
      using c unfolding curr-def fun-eq-iff
      apply(elim allE[of - a]) apply simp unfolding fun-eq-iff by auto
    qed
  qed
qed

lemma curr-surj:
assumes g ∈ Func A (Func B C)
shows ∃ f ∈ Func (A <*> B) C. curr A f = g
proof
  let ?f = λ ab. case g (fst ab) of None ⇒ None | Some g1 ⇒ g1 (snd ab)
  show curr A ?f = g
  proof (rule ext)
    fix a show curr A ?f a = g a
    proof (cases a ∈ A)
      case False
      hence g a = None using assms unfolding Func-def by auto
      thus ?thesis unfolding curr-def using False by simp
    next
```

case *True*
obtain *g1* **where** *g1* ∈ *Func B C* **and** *g a* = *Some g1*
using *assms* **using** *Func-elim*[*OF assms True*] **by** *blast*
thus *?thesis* **using** *True* **unfolding** *curr-def* **by** *auto*
qed
qed
show *?f* ∈ *Func* (*A* <∗> *B*) *C*
unfolding *Func-def mem-Collect-eq* **proof**(*intro conjI allI ballI*)
fix *ab* **show** *?f ab* ≠ *None* ⟷ *ab* ∈ *A* × *B*
proof(*cases g* (*fst ab*))
case *None*
hence *fst ab* ∉ *A* **using** *assms* **unfolding** *Func-def* **by** *force*
thus *?thesis* **using** *None* **by** *auto*
next
case (*Some g1*)
hence *fst*: *fst ab* ∈ *A* **and** *g1*: *g1* ∈ *Func B C*
using *assms* **unfolding** *Func-def*[*of A*] **by** *force*+
hence *?f ab* ≠ *None* ⟷ *g1* (*snd ab*) ≠ *None* **using** *Some* **by** *auto*
also have ... ⟷ *snd ab* ∈ *B* **using** *g1* **unfolding** *Func-def* **by** *auto*
also have ... ⟷ *ab* ∈ *A* × *B* **using** *fst* **by** (*cases ab*, *auto*)
finally show *?thesis* **.**
qed
next
fix *ab* **assume** *ab*: *ab* ∈ *A* × *B*
hence *fst ab* ∈ *A* **and** *snd ab* ∈ *B* **by**(*cases ab*, *auto*)
then obtain *g1* **where** *g1* ∈ *Func B C* **and** *g* (*fst ab*) = *Some g1*
using *assms* **using** *Func-elim*[*OF assms*] **by** *blast*
thus *case ?f ab of Some c* ⇒ *c* ∈ *C* |*None* ⇒ *True*
unfolding *Func-def* **by** *auto*
qed
qed

lemma *bij-betwe-curr*:
bij-betw (*curr A*) (*Func* (*A* <∗> *B*) *C*) (*Func A* (*Func B C*))
unfolding *bij-betw-def inj-on-def image-def*
using *curr-in curr-inj curr-surj* **by** *blast*

lemma *card-of-Func-Times*:
|*Func* (*A* <∗> *B*) *C*| =*o* |*Func A* (*Func B C*)|
unfolding *card-of-ordIso*[*symmetric*]
using *bij-betwe-curr* **by** *blast*

definition *Func-map* **where**
Func-map B2 f1 f2 g b2 ≡
if b2 ∈ *B2 then case g* (*f2 b2*) *of None* ⇒ *None* | *Some a1* ⇒ *Some* (*f1 a1*)
else None

lemma *Func-map*:
assumes *g*: *g* ∈ *Func A2 A1* **and** *f1*: *f1* ‘ *A1* ⊆ *B1* **and** *f2*: *f2* ‘ *B2* ⊆ *A2*

shows *Func-map B2 f1 f2 g* $\in$ *Func B2 B1*
unfolding *Func-def mem-Collect-eq* **proof**(*intro conjI allI ballI*)
  fix *b2* **show** *Func-map B2 f1 f2 g b2* $\neq$ *None* $\longleftrightarrow$ *b2* $\in$ *B2*
  proof(*cases b2* $\in$ *B2*)
    case *True*
    hence *f2 b2* $\in$ *A2* **using** *f2* **by** *auto*
    then obtain *a1* **where** *g* (*f2 b2*) = *Some a1* **and** *a1* $\in$ *A1*
    using *g* **unfolding** *Func-def* **by**(*cases g* (*f2 b2*), *fastforce+*)
    thus *?thesis* **unfolding** *Func-map-def* **using** *True* **by** *auto*
  qed(*unfold Func-map-def*, *auto*)
next
  fix *b2* **assume** *b2*: *b2* $\in$ *B2*
  hence *f2 b2* $\in$ *A2* **using** *f2* **by** *auto*
  then obtain *a1* **where** *g* (*f2 b2*) = *Some a1* **and** *a1* $\in$ *A1*
  using *g* **unfolding** *Func-def* **by**(*cases g* (*f2 b2*), *fastforce+*)
  thus *case Func-map B2 f1 f2 g b2 of None* $\Rightarrow$ *True* | *Some b1* $\Rightarrow$ *b1* $\in$ *B1*
  unfolding *Func-map-def* **using** *b2 f1* **by** *auto*
qed

lemma *Func-map-empty*[*simp*]:
Func-map B2 f1 f2 empty = *empty*
unfolding *Func-map-def-raw* **by** (*rule ext*, *auto*)

lemma *Func-emp-empty*[*simp*]:
Func {} *B* = {*empty*}
unfolding *Func-def* **by** *auto*

lemma *Func-non-emp*:
assumes *B* $\neq$ {}
shows *Func A B* $\neq$ {}
proof−
  obtain *b* **where** *b*: *b* $\in$ *B* **using** *assms* **by** *auto*
  hence (λ *a*. *if a* $\in$ *A then Some b else None*) $\in$ *Func A B*
  unfolding *Func-def* **by** *auto*
  thus *?thesis* **by** *blast*
qed

lemma *Func-is-emp*[*simp*]:
Func A B = {} $\longleftrightarrow$ *A* $\neq$ {} $\wedge$ *B* = {} (**is** *?L* $\longleftrightarrow$ *?R*)
proof
  assume *L*: *?L*
  moreover {**assume** *A* = {} **hence** *False* **using** *L* **by** *auto*}
  moreover {**assume** *B* $\neq$ {} **hence** *False* **using** *L Func-non-emp* **by** *metis*}
  ultimately show *?R* **by** *blast*
next
  assume *R*: *?R*
  moreover
  {**fix** *f* **assume** *f* $\in$ *Func A B*
   moreover obtain *a* **where** *a* $\in$ *A* **using** *R* **by** *blast*

```
   ultimately obtain b where b ∈ B unfolding Func-def by(cases f a, force+)
   with R have False by auto
  }
  thus ?L by blast
qed

lemma Func-emp2[simp]: A ≠ {} ⟹ Func A {} = {} by auto

lemma empty-in-Func[simp]:
B ≠ {} ⟹ empty ∈ Func {} B
unfolding Func-def by auto

lemma Func-map-surj:
assumes B1: f1 ‘ A1 = B1 and A2: inj-on f2 B2 f2 ‘ B2 ⊆ A2
and B2A2: B2 = {} ⟹ A2 = {}
shows Func B2 B1 = Func-map B2 f1 f2 ‘ Func A2 A1
proof(cases B2 = {})
  case True
  thus ?thesis using B2A2 by auto
next
  case False note B2 = False
  show ?thesis
proof safe
  fix h assume h: h ∈ Func B2 B1
  def j1 ≡ inv-into A1 f1
  have ∀ a2 ∈ f2 ‘ B2. ∃ b2. b2 ∈ B2 ∧ f2 b2 = a2 by blast
  then obtain k where k: ∀ a2 ∈ f2 ‘ B2. k a2 ∈ B2 ∧ f2 (k a2) = a2 by metis
  {fix b2 assume b2: b2 ∈ B2
   hence f2 (k (f2 b2)) = f2 b2 using k A2(2) by auto
   moreover have k (f2 b2) ∈ B2 using b2 A2(2) k by auto
    ultimately have k (f2 b2) = b2 using b2 A2(1) unfolding inj-on-def by
blast
  } note kk = this
  obtain b22 where b22: b22 ∈ B2 using B2 by auto
  def j2 ≡ λ a2. if a2 ∈ f2 ‘ B2 then k a2 else b22
  have j2A2: j2 ‘ A2 ⊆ B2 unfolding j2-def using k b22 by auto
  have j2: ⋀ b2. b2 ∈ B2 ⟹ j2 (f2 b2) = b2
  using kk unfolding j2-def by auto
  def g ≡ Func-map A2 j1 j2 h
  have Func-map B2 f1 f2 g = h
  proof (rule ext)
    fix b2 show Func-map B2 f1 f2 g b2 = h b2
    proof(cases b2 ∈ B2)
      case True
      show ?thesis
      proof (cases h b2)
        case (Some b1)
        hence b1: b1 ∈ f1 ‘ A1 using True h unfolding B1 Func-def by auto
        show ?thesis
```

using *Some True A2 f-inv-into-f*[*OF b1*]
unfolding *g-def Func-map-def j1-def j2*[*OF True*] **by** *auto*
qed(*insert A2 True j2*[*OF True*], *unfold g-def Func-map-def*, *auto*)
qed(*insert h*, *unfold Func-def Func-map-def*, *auto*)
qed
moreover have $g \in$ *Func A2 A1* **unfolding** *g-def* **apply**(*rule Func-map*[*OF h*])
using *inv-into-into j2A2 B1 A2 inv-into-into*
unfolding *j1-def image-def* **by**(*force*, *force*)
ultimately show $h \in$ *Func-map B2 f1 f2 ' Func A2 A1*
unfolding *Func-map-def-raw* **unfolding** *image-def* **by** *auto*
qed(*insert B1 Func-map*[*OF - - A2*(*2*)], *auto*)
qed

definition *Pfunc* **where**
Pfunc A B $\equiv$
$\{f.\ (\forall\, a.\ f\ a \neq None \longrightarrow a \in A) \wedge$
$(\forall\, a.\ case\ f\ a\ of\ None \Rightarrow True \mid Some\ b \Rightarrow b \in B)\}$

lemma *Func-mono*[*simp*]:
assumes $B1 \subseteq B2$
shows *Func A B1* $\subseteq$ *Func A B2*
using *assms* **unfolding** *Func-def* **by** *force*

lemma *Pfunc-mono*[*simp*]:
assumes $A1 \subseteq A2$ **and** $B1 \subseteq B2$
shows *Pfunc A B1* $\subseteq$ *Pfunc A B2*
using *assms in-mono* **unfolding** *Pfunc-def* **apply** *safe*
apply(*case-tac x a*, *auto*)
by (*metis in-mono option.simps*(*5*))

lemma *Func-Pfunc*:
Func A B $\subseteq$ *Pfunc A B*
unfolding *Func-def Pfunc-def* **by** *auto*

lemma *Pfunc-Func*:
Pfunc A B $= (\bigcup\ A' \in Pow\ A.\ Func\ A'\ B)$
proof *safe*
fix *f* **assume** *f*: $f \in$ *Pfunc A B*
show $f \in (\bigcup A' \in Pow\ A.\ Func\ A'\ B)$
proof (*intro UN-I*)
let $?A' = \{a.\ f\ a \neq None\}$
show $?A' \in Pow\ A$ **using** *f* **unfolding** *Pow-def Pfunc-def* **by** *auto*
show $f \in Func\ ?A'\ B$ **using** *f* **unfolding** *Func-def Pfunc-def* **by** *auto*
qed
next
fix $f\ A'$ **assume** $f \in Func\ A'\ B$ **and** $A' \subseteq A$
thus $f \in$ *Pfunc A B* **unfolding** *Func-def Pfunc-def* **by** *auto*

qed

lemma *card-of-Pow-Func*:
$|Pow\ A| =o\ |Func\ A\ (UNIV::bool\ set)|$
proof−
def $F \equiv \lambda\ A'\ a.$ *if* $a \in A$ *then* (*if* $a \in A'$ *then Some True else Some False*)
else None
have *bij-betw* F (*Pow* A) (*Func* A (*UNIV*::*bool set*))
unfolding *bij-betw-def inj-on-def* **proof** (*intro ballI impI conjI*)
fix *A1 A2* **assume** *A1*: $A1 \in Pow\ A$ **and** *A2*: $A2 \in Pow\ A$ **and** *eq*: $F\ A1 = F\ A2$
show $A1 = A2$
proof−
{**fix** a
have $a \in A1 \longleftrightarrow F\ A1\ a = Some\ True$ **using** *A1* **unfolding** *F-def Pow-def* **by** *auto*
also have ... $\longleftrightarrow F\ A2\ a = Some\ True$ **unfolding** *eq* **..**
also have ... $\longleftrightarrow a \in A2$ **using** *A2* **unfolding** *F-def Pow-def* **by** *auto*
finally have $a \in A1 \longleftrightarrow a \in A2$ **.**
}
thus *?thesis* **by** *auto*
qed
next
show F ‘ *Pow* A = *Func* A *UNIV*
proof *safe*
fix f **assume** f: $f \in Func\ A\ (UNIV::bool\ set)$
show $f \in F$ ‘ *Pow* A **unfolding** *image-def mem-Collect-eq* **proof**(*intro bexI*)
let $?A1 = \{a \in A.\ f\ a = Some\ True\}$
show $f = F\ ?A1$ **unfolding** *F-def* **apply**(*rule ext*)
using f **unfolding** *Func-def mem-Collect-eq* **by** (*auto*,*force*)
qed *auto*
qed(*unfold Func-def mem-Collect-eq F-def*, *auto*)
qed
thus *?thesis* **unfolding** *card-of-ordIso*[*symmetric*] **by** *blast*
qed

lemma *card-of-Func-mono*:
fixes *A1 A2* :: *'a set* **and** B :: *'b set*
assumes *A12*: $A1 \subseteq A2$ **and** B: $B \neq \{\}$
shows $|Func\ A1\ B| \leq o\ |Func\ A2\ B|$
proof−
obtain *bb* **where** *bb*: $bb \in B$ **using** B **by** *auto*
def $F \equiv \lambda\ (f1::'a \Rightarrow 'b\ option)\ a.$ *if* $a \in A2$ *then* (*if* $a \in A1$ *then* $f1\ a$ *else Some* bb)
else None
show *?thesis* **unfolding** *card-of-ordLeq*[*symmetric*] **proof**(*intro exI*[*of - F*] *conjI*)
show *inj-on* F (*Func A1 B*) **unfolding** *inj-on-def* **proof** *safe*
fix $f\ g$ **assume** f: $f \in Func\ A1\ B$ **and** g: $g \in Func\ A1\ B$ **and** *eq*: $F\ f = F\ g$
show $f = g$

proof(*rule ext*)
fix *a* **show** *f a* = *g a*
proof(*cases a* ∈ *A1*)
case *True*
thus *?thesis* **using** *eq A12* **unfolding** *F-def fun-eq-iff*
by (*elim allE*[*of - a*]) *auto*
qed(*insert f g*, *unfold Func-def*, *fastforce*)
qed
qed
qed(*insert bb*, *unfold Func-def F-def*, *force*)
qed

lemma *card-of-Pfunc-Pow-Func*:
assumes $B \neq \{\}$
shows $|Pfunc\ A\ B| \leq o\ |Pow\ A <*> Func\ A\ B|$
proof−
have $|Pfunc\ A\ B| =o\ |\bigcup\ A' \in Pow\ A.\ Func\ A'\ B|$ (**is** - =*o ?K*)
unfolding *Pfunc-Func* **by**(*rule card-of-refl*)
also have $?K \leq o\ |Sigma\ (Pow\ A)\ (\lambda\ A'.\ Func\ A'\ B)|$ **using** *card-of-UNION-Sigma*
.
also have $|Sigma\ (Pow\ A)\ (\lambda\ A'.\ Func\ A'\ B)| \leq o\ |Pow\ A <*> Func\ A\ B|$
apply(*rule card-of-Sigma-mono1*) **using** *card-of-Func-mono*[*OF - assms*] **by** *auto*
finally show *?thesis* **.**
qed

lemma *ordLeq-Func*:
assumes $\{b1,b2\} \subseteq B\ b1 \neq b2$
shows $|A| \leq o\ |Func\ A\ B|$
unfolding *card-of-ordLeq*[*symmetric*] **proof**(*intro exI conjI*)
let $?F = \lambda\ aa\ a.$ *if a* ∈ *A then* (*if a* = *aa then Some b1 else Some b2*)
else None
show *inj-on ?F A* **using** *assms* **unfolding** *inj-on-def fun-eq-iff* **by** *auto*
show *?F ‘ A* ⊆ *Func A B* **using** *assms* **unfolding** *Func-def* **apply** *auto*
by (*metis option.simps*(*3*))
qed

lemma *infinite-Func*:
assumes *A*: *infinite A* **and** *B*: $\{b1,b2\} \subseteq B\ b1 \neq b2$
shows *infinite* (*Func A B*)
using *ordLeq-Func*[*OF B*] **by** (*metis A card-of-ordLeq-finite*)

definition *Ffunc* **where**
$Ffunc\ A\ B \equiv \{f\ .\ (\forall\ a \in A.\ f\ a \in B) \wedge (\forall\ a.\ a \notin A \longrightarrow f\ a = undefined)\}$

lemma *card-of-Func-Ffunc*:
$|Ffunc\ A\ B| =o\ |Func\ A\ B|$
unfolding *card-of-ordIso*[*symmetric*] **proof**
let $?F = \lambda\ f\ a.$ *if a* ∈ *A then Some* (*f a*) *else None*

```
  show bij-betw ?F (Ffunc A B) (Func A B)
  unfolding bij-betw-def unfolding inj-on-def proof(intro conjI ballI impI)
    fix f g assume f: f ∈ Ffunc A B and g: g ∈ Ffunc A B and eq: ?F f = ?F g
    show f = g
    proof(rule ext)
      fix a
      show f a = g a
      proof(cases a ∈ A)
        case True
        have Some (f a) = ?F f a using True by auto
        also have ... = ?F g a using eq unfolding fun-eq-iff by(rule allE)
        also have ... = Some (g a) using True by auto
        finally have Some (f a) = Some (g a) .
        thus ?thesis by simp
      qed(insert f g, unfold Ffunc-def Ffunc-def, auto)
    qed
  next
    show ?F ‘ Ffunc A B = Func A B
    proof safe
      fix f assume f: f ∈ Func A B
      def g ≡ λ a. case f a of Some b ⇒ b | None ⇒ undefined
      have g ∈ Ffunc A B
      using f unfolding g-def Func-def Ffunc-def by force+
      moreover have f = ?F g
      proof(rule ext)
        fix a show f a = ?F g a
        using f unfolding Func-def g-def by (cases a ∈ A) force+
      qed
      ultimately show f ∈ ?F ‘ (Ffunc A B) by blast
    qed(unfold Ffunc-def Func-def, auto)
  qed
qed


end
```

9 Cardinal Arithmetic

theory *Cardinal-Arithmetic*
imports *Cardinal-Order-Relation*
begin

The following collection of lemmas should be seen as an user interface to the HOL Theory of cardinals. It is not expected to be complete in any sense, since it's development was driven by demand arising from the development of the (co)datatype package.

lemma *ordIso-refl*: *Card-order* $r \Longrightarrow r =o\ r$
by (*rule card-order-on-ordIso*) *assumption*

lemma *ordLeq-refl*: *Card-order* $r \Longrightarrow r \leq o\ r$
by (*rule ordIso-imp-ordLeq*, *rule card-order-on-ordIso*) *assumption*

lemma *card-of-refl-ordLeq*: $|A| \leq o\ |A|$
by *simp*

lemma *card-of-ordIso-subst*: $A = B \Longrightarrow |A| =o\ |B|$
by (*simp only*: *ordIso-refl card-of-Card-order*)

lemma *card-of-Ball*: $|\{x \in A.\ P\ x\}| \leq o\ |A|$ (**is** $|?L| \leq o\ |?R|$)
proof −
  have *inj-on id ?L* **unfolding** *inj-on-def* **by** *simp*
  moreover have $id\ `\ ?L \subseteq\ ?R$ **by** *auto*
  ultimately show *?thesis* **unfolding** *card-of-ordLeq*[*symmetric*] **by** *blast*
qed

lemma *ordIso-bij*: $|A| =o\ |B| \Longrightarrow \exists f.$ *bij-betw f A B*
unfolding *ordIso-def iso-def-raw* **by** *auto*

lemma *card-of-Times-Plus-distrib*:
  $|A <*> (B <+> C)| =o\ |A <*> B <+> A <*> C|$ (**is** $|?RHS| =o\ |?LHS|$)
proof −
  let $?f = \lambda(a,\ bc).$ *case bc of Inl b* $\Rightarrow$ *Inl* $(a,\ b)$ | *Inr c* $\Rightarrow$ *Inr* $(a,\ c)$
  have *bij-betw ?f ?RHS ?LHS* **unfolding** *bij-betw-def inj-on-def* **by** *force*
  thus *?thesis* **using** *card-of-ordIso* **by** *blast*
qed

9.1 Zero

definition *czero* **where**
  czero = *card-of* {}

lemma *empty-czero*[*simp*]:
  $|\{\}| = czero$
unfolding *czero-def* **by** *simp*

lemma *czero-ordIso*[*simp*]:
  czero =*o czero*
using *card-of-empty-ordIso* **by** *simp*

lemma *empty-czero-ordIso*:
  $|\{\}| =o$ *czero*
using *card-of-empty-ordIso* **by** *simp*

lemma *czero-ordLeq*[*simp*]:
  Card-order $r \Longrightarrow$ *czero* $\leq o$ r
unfolding *czero-def* **using** *Card-order-empty* **.**

lemma *card-of-czero-iff-empty*:
  $|A| =$ *czero* $\longleftrightarrow A = \{\}$
proof
  assume $|A| =$ *czero*
  hence *Field* $|A| =$ *Field czero* **by** *auto*
  thus $A = \{\}$ **unfolding** *czero-def* **by** (*simp only*: *Field-card-of*)
qed *simp*

lemma *card-of-ordIso-czero-iff-empty*:
  $|A| =o$ (*czero* :: *'a rel*) $\longleftrightarrow A = (\{\}$ :: *'a set*)
unfolding *czero-def* **by** (*rule iffI*[*OF card-of-empty2*]) *auto*

abbreviation *Cnotzero* **where**
  Cnotzero (r :: *'a rel*) $\equiv \neg(r =o$ (*czero* :: *'a rel*)) $\wedge$ *Card-order* r

lemma *Cnotzero-imp-not-empty*:
  assumes *Cnotzero* r
  shows *Field* $r \neq \{\}$
proof (*rule ccontr*)
  assume $\neg$ (*Field* $r \neq \{\}$)
  hence $|$*Field* $r| =o$ *czero* **by** *simp*
  with *assms*(*1*) **have** $r =o$ *czero* **by** (*blast intro*: *card-of-unique ordIso-transitive*)
  with *assms* **show** *False* **by** *blast*
qed

lemma *czeroI*:
  $[\![$*Card-order* r; *Field* $r = \{\}]\!] \Longrightarrow r =o$ *czero*
using *Cnotzero-imp-not-empty ordIso-transitive*[*OF - czero-ordIso*] **by** *blast*

lemma *czeroE*:
  $r =o$ *czero* $\Longrightarrow$ *Field* $r = \{\}$
unfolding *czero-def*
by (*drule card-of-cong*) (*simp only*: *Field-card-of card-of-empty2*)

lemma *Cnotzero-mono*:
  $[\![$*Cnotzero* r; *Card-order* q; $r \leq o$ $q]\!] \Longrightarrow$ *Cnotzero* q
apply (*rule ccontr*)
apply *auto*

apply (*drule czeroE*)
apply (*erule notE*)
apply (*erule czeroI*)
apply (*drule card-of-mono2*)
apply (*simp only*: *card-of-empty3*)
done

9.2 Infinite cardinals

definition *cinfinite* **where**
  cinfinite r = *infinite* (*Field r*)

abbreviation *Cinfinite* **where**
  Cinfinite r ≡ *cinfinite r* ∧ *Card-order r*

lemma *natLeq-ordLeq-cinfinite*:
  assumes *inf*: *Cinfinite r*
  shows *natLeq* ≤*o* *r*
proof −
  from *inf* **have** *natLeq* ≤*o* |*Field r*| **by** (*simp add*: *cinfinite-def infinite-iff-natLeq-ordLeq*)
  also from *inf* **have** |*Field r*| =*o* *r* **by** *simp*
  finally show *?thesis* **.**
qed

lemma *cinfinite-not-czero*[*simp*]:
  cinfinite r ⟹ ¬ (*r* =*o* (*czero* :: ′*a rel*))
proof
  assume *cinfinite r r* =*o* (*czero* :: ′*a rel*)
  then obtain *f* **where** *bij-betw f* (*Field r*) (*Field* (*czero* :: ′*a rel*))
    unfolding *ordIso-def iso-def-raw* **by** *auto*
  hence *bij-betw f* (*Field r*) {} **unfolding** *czero-def* **by** (*simp only*: *Field-card-of*)
  hence *Field r* = {} **using** *Fun.bij-betw-empty2* **by** *blast*
  with ‹*cinfinite r*› **show** *False* **unfolding** *cinfinite-def* **by** *simp*
qed

lemma *Cinfinite-Cnotzero*[*simp*]:
  Cinfinite r ⟹ *Cnotzero r*
by *simp*

lemma *Cinfinite-cong*:
  assumes *r1* =*o* *r2 Cinfinite r1*
  shows *Cinfinite r2*
proof (*unfold cinfinite-def*, *rule ccontr*)
  assume ¬ (*infinite* (*Field r2*) ∧ *Card-order r2*)
  hence *finite* (*Field r2*) ∨ ¬ *Card-order r2* **by** *simp*
  with *assms* **have** *finite* (*Field r2*) **using** *Card-order-ordIso2* **by** *blast*
  hence *finite* (*Field r1*)
    using *card-of-ordIso-finite*[*OF card-of-cong*[*OF assms*(*1*)]] **by** *simp*
  with *assms*(*2*) **show** *False* **unfolding** *cinfinite-def* **by** *simp*

qed

lemma *cinfinite-mono*:
assumes $r1 \leq o\ r2$ *cinfinite r1*
shows *cinfinite r2*
proof (*unfold cinfinite-def*, *rule ccontr*)
assume ¬ *infinite* (*Field r2*)
hence *finite* (*Field r2*) **by** *simp*
hence *finite* (*Field r1*)
using *card-of-ordLeq-finite*[*OF card-of-mono2*[*OF assms*(*1*)]] **by** *simp*
with *assms*(*2*) **show** *False* **unfolding** *cinfinite-def* **by** *simp*
qed

lemma *card-of-Cinfinite-mono*:
$[\![r \leq o\ |A|;\ Cinfinite\ r]\!] \Longrightarrow Cinfinite\ |A|$
using *cinfinite-mono card-of-Card-order* **by** (*rule conjI*) *auto*

lemma *Cinfinite-mono*:
$[\![r1 \leq o\ r2;\ Cinfinite\ r1;\ Card\text{-}order\ r2]\!] \Longrightarrow Cinfinite\ r2$
using *cinfinite-mono* **by** (*rule conjI*) *auto*

9.3 Binary sum

definition *csum* (**infixr** *+c 65*) **where**
$r1 +c\ r2 \equiv |Field\ r1\ $ <+> $Field\ r2|$

lemma *Card-order-csum*[*simp*]:
Card-order (*r1 +c r2*)
unfolding *csum-def* **by** *simp*

lemma *csum-not-czero*[*simp*]:
$A \neq \{\} \vee B \neq \{\} \Longrightarrow \neg\ (\ |A| +c\ |B| =o\ czero)$
unfolding *czero-def csum-def*
using *Field-card-of Plus-eq-empty-conv card-of-empty2* **by** *auto*

lemma *csum-Cnotzero*:
$A \neq \{\} \vee B \neq \{\} \Longrightarrow Cnotzero\ (\ |A| +c\ |B|\ )$
unfolding *czero-def csum-def*
using *Field-card-of Plus-eq-empty-conv card-of-empty2* **by** *auto*

lemma *csum-not-czero1*[*simp*]:
assumes *Cnotzero r1*
shows ¬ (*r1 +c r2 =o czero*)
proof −
from *assms*(*1*) **have** ∗: $|Field\ r1| =o\ r1$ **by** *simp*
moreover
from *assms* **have** $Field\ r1 \neq \{\}$ **using** *Cnotzero-imp-not-empty*[*of r1*] **by** *simp*
hence $\neg\ (\ |Field\ r1| +c\ |Field\ r2| =o\ czero)$ **by** *simp*
ultimately show *?thesis* **by** (*simp add*: *csum-def*)

qed

lemma *csum-Cnotzero1*:
  Cnotzero r1 $\Longrightarrow$ *Cnotzero* (*r1* +*c* *r2*)
by *simp*

lemma *csum-not-czero2*[*simp*]:
  assumes *Cnotzero r2*
  shows ¬ (*r1* +*c* *r2* =*o* *czero*)
proof −
  from *assms*(*1*) **have** ∗: |*Field r2*| =*o* *r2* **by** *auto*
  moreover
  from *assms* **have** *Field r2* ≠ {} **using** *Cnotzero-imp-not-empty*[*of r2*] **by** *auto*
  hence ¬ (|*Field r1*| +*c* |*Field r2*| =*o* *czero*) **by** *simp*
  ultimately show *?thesis* **by** (*simp add*: *csum-def*)
qed

lemma *csum-Cnotzero2*:
  Cnotzero r2 $\Longrightarrow$ *Cnotzero* (*r1* +*c* *r2*)
by *simp*

lemma *csum-not-czero'*[*simp*]:
  assumes *Cnotzero r1* ∨ *Cnotzero r2*
  shows ¬ (*r1* +*c* *r2* =*o* *czero*)
proof −
  from *assms* **have** *Field r1* ≠ {} ∨ *Field r2* ≠ {}
    using *Cnotzero-imp-not-empty*[*of r1*] *Cnotzero-imp-not-empty*[*of r2*] **by** *blast*
  hence ¬ (|*Field r1*| +*c* |*Field r2*| =*o* *czero*) **by** (*rule csum-not-czero*)
  thus *?thesis* **by** (*simp add*: *csum-def*)
qed

lemma *csum-Cnotzero'*:
  Cnotzero r1 ∨ *Cnotzero r2* $\Longrightarrow$ *Cnotzero* (*r1* +*c* *r2*)
by *simp*

lemma *card-order-csum*[*simp*]:
  assumes *card-order r1 card-order r2*
  shows *card-order* (*r1* +*c* *r2*)
proof −
  have *Field r1* = *UNIV Field r2* = *UNIV* **using** *assms card-order-on-Card-order*
by *auto*
  thus *?thesis* **unfolding** *csum-def* **by** *auto*
qed

lemma *cinfinite-csum*[*simp*]:
  cinfinite r1 ∨ *cinfinite r2* $\Longrightarrow$ *cinfinite* (*r1* +*c* *r2*)
unfolding *cinfinite-def csum-def* **by** *auto*

lemma *Cinfinite-csum*[*simp*]:

Cinfinite r1 ∨ *Cinfinite r2* ⟹ *Cinfinite* (*r1* +*c r2*)
unfolding *cinfinite-def csum-def* **by** *auto*

lemma *csum-cong*[*simp*]:
assumes *p1* =*o r1* **and** *p2* =*o r2*
shows *p1* +*c p2* =*o r1* +*c r2*
unfolding *csum-def* **by** (*simp only*: *assms ordIso-Plus-cong*)

lemma *csum-cong1*[*simp*]:
assumes *p1* =*o r1*
shows *p1* +*c q* =*o r1* +*c q*
unfolding *csum-def* **by** (*simp only*: *assms ordIso-Plus-cong1*)

lemma *csum-cong2*[*simp*]:
assumes *p2* =*o r2*
shows *q* +*c p2* =*o q* +*c r2*
unfolding *csum-def* **by** (*simp only*: *assms ordIso-Plus-cong2*)

lemma *csum-mono*[*simp*]:
assumes *p1* ≤*o r1* **and** *p2* ≤*o r2*
shows *p1* +*c p2* ≤*o r1* +*c r2*
unfolding *csum-def* **by** (*simp only*: *assms ordLeq-Plus-mono*)

lemma *csum-mono1*[*simp*]:
assumes *p1* ≤*o r1*
shows *p1* +*c q* ≤*o r1* +*c q*
unfolding *csum-def* **by** (*simp only*: *assms ordLeq-Plus-mono1*)

lemma *csum-mono2*[*simp*]:
assumes *p2* ≤*o r2*
shows *q* +*c p2* ≤*o q* +*c r2*
unfolding *csum-def* **by** (*simp only*: *assms ordLeq-Plus-mono2*)

lemma *ordLeq-csum1*:
assumes *Card-order p1*
shows *p1* ≤*o p1* +*c p2*
unfolding *csum-def* **by** (*simp only*: *assms Card-order-Plus1*)

lemma *ordLeq-csum2*:
assumes *Card-order p2*
shows *p2* ≤*o p1* +*c p2*
unfolding *csum-def* **by** (*simp only*: *assms Card-order-Plus2*)

lemma *csum-com*:
p1 +*c p2* =*o p2* +*c p1*
unfolding *csum-def* **by** (*simp only*: *card-of-Plus-commute*)

lemma *csum-assoc*:
(*p1* +*c p2*) +*c p3* =*o p1* +*c p2* +*c p3*

unfolding *csum-def* **by** (*simp only*: *Field-card-of card-of-Plus-assoc*)

lemma *Plus-csum*[*simp*]:
$|A <+> B| =o\ |A| +c\ |B|$
unfolding *csum-def* **by** (*simp only*: *Field-card-of card-of-refl*)

lemma *czero-csum*[*simp*]:
assumes *Card-order r*
shows *czero +c r =o r*
proof −
have $|\{\} <+> Field\ r| =o\ r$ **by** (*simp only*: *Card-order-Plus-empty2 assms ordIso-symmetric*)
thus *?thesis* **unfolding** *csum-def czero-def* **by** (*simp only*: *Field-card-of*)
qed

lemma *csum-czero*[*simp*]:
assumes *Card-order r*
shows *r +c czero =o r*
proof −
have $|Field\ r <+> \{\}| =o\ r$ **by** (*simp only*: *Card-order-Plus-empty1 assms ordIso-symmetric*)
thus *?thesis* **unfolding** *csum-def czero-def* **by** (*simp only*: *Field-card-of*)
qed

lemma *Un-csum*[*simp*]:
$|A \cup B| \leq o\ |A| +c\ |B|$
using *ordLeq-ordIso-trans*[*OF card-of-Un-Plus-ordLeq Plus-csum*] **by** *blast*

lemmas *Un-csum3*[*simp*] =
ordLeq-transitive[*OF Un-csum csum-mono1*[*OF Un-csum*]]
ordLeq-transitive[*OF Un-csum csum-mono2*[*OF Un-csum*]]

9.4 One

definition *cone* **where**
cone = *card-of* $\{()\}$

lemma *card-order-cone*[*simp*]:
card-order cone
unfolding *cone-def* **using** *UNIV-unit* **by** *simp*

lemma *Card-order-cone*[*simp*]:
Card-order cone
unfolding *cone-def* **by** *simp*

lemma *single-cone*:
$|\{x\}| =o\ cone$
proof −
let *?f* = $\lambda x.\ ()$

have *bij-betw ?f* $\{x\}$ $\{()\}$ **unfolding** *bij-betw-def* **by** *auto*
thus *?thesis* **unfolding** *cone-def* **using** *card-of-ordIso* **by** *blast*
qed

lemma *czero-not-cone*:
$\neg$ (*czero =o cone*)
using *card-of-empty3*[*of* $\{()\}$] *ordIso-iff-ordLeq* **by** (*auto simp*: *cone-def*)

lemma *cone-not-czero*:
$\neg$ (*cone =o czero*)
using *card-of-empty3*[*of* $\{()\}$] *ordIso-iff-ordLeq* **by** (*auto simp*: *cone-def*)

lemma *cone-Cnotzero*:
Cnotzero cone
by (*simp add*: *cone-not-czero*)

lemma *cone-ordLeq-Cnotzero*:
assumes *r*: *Cnotzero r* (**is** $\neg$ (- *=o ?zero*) $\wedge$ -)
shows *cone* $\leq$*o r*
proof −
from *r* **have** *Field r* $\neq \{\}$ **using** *czeroI* **by** *blast*
then obtain *x* **where** $x \in$ *Field r* **by** *blast*
hence *cone* $\leq$*o* |*Field r*|
unfolding *cone-def inj-on-def card-of-ordLeq*[*symmetric*]
using *exI*[*of* - λ-. *x*] **by** *auto*
with *r* **show** *?thesis* **by** (*simp add*: *ordLeq-ordIso-trans*)
qed

9.5 Two

definition *ctwo* **where**
ctwo = |*UNIV* :: *bool set*|

lemma *Card-order-ctwo*[*simp*]:
Card-order ctwo
unfolding *ctwo-def* **by** *simp*

lemma *ordLeq-ctwo*:
|$\{a,b\}$| $\leq$*o ctwo*
proof −
have *ab*: $\{a,b\} \neq \{\}$ **by** *simp*
show *?thesis* **unfolding** *ctwo-def card-of-ordLeq2*[*OF ab*, *symmetric*]
by (*rule exI*[*of* - λ *c*. *if c then a else b*]) *auto*
qed

lemma *ordIso-ctwo*:
$a \neq b \Longrightarrow$ |$\{a,b\}$| *=o ctwo*
unfolding *ctwo-def card-of-ordIso*[*symmetric*] *bij-betw-def*
by (*rule exI*[*of* - $\lambda c.\ c = a$]) *auto*

lemma *cone-ordLeq-ctwo*:
cone $\leq o$ *ctwo*
unfolding *cone-def ctwo-def card-of-ordLeq*[*symmetric*] **by** *auto*

lemma *cone-ordLess-ctwo*:
cone $<o$ *ctwo*
proof−
{ assume *ctwo* $\leq o$ *cone*
hence $|UNIV::bool\ set| \leq o\ |UNIV::unit\ set|$
unfolding *ctwo-def cone-def* **by** (*auto intro*: *card-of-UNIV ordLeq-transitive*)
then obtain $f::bool \Rightarrow unit$ **where** *inj f*
unfolding *card-of-ordLeq*[*symmetric*] **by** *auto*
hence *f True* $\neq$ *f False* **unfolding** *inj-on-def* **by** *auto*
hence *False* **by** *auto*
}
thus *?thesis* **using** *cone-ordLeq-ctwo*
using *ordIso-iff-ordLeq ordLeq-iff-ordLess-or-ordIso* **by** *auto*
qed

lemma *czero-not-ctwo*:
$\neg$ (*czero* $=o$ *ctwo*)
using *card-of-empty3*[*of UNIV* :: *bool set*] *ordIso-iff-ordLeq* **by** (*auto simp*: *ctwo-def*)

lemma *ctwo-not-czero*:
$\neg$ (*ctwo* $=o$ *czero*)
using *card-of-empty3*[*of UNIV* :: *bool set*] *ordIso-iff-ordLeq* **by** (*auto simp*: *ctwo-def*)

lemma *ctwo-Cnotzero*:
Cnotzero ctwo
by (*simp add*: *ctwo-not-czero*)

9.6 Family sum

definition *Csum* **where**
Csum r rs $\equiv$ $|SIGMA\ i : Field\ r.\ Field\ (rs\ i)|$

syntax *-Csum* ::
pttrn => ('*a* ∗ '*a*) *set* => '*b* ∗ '*b set* => (('*a* ∗ '*b*) ∗ ('*a* ∗ '*b*)) *set*
((*3CSUM* -:-. -) [*0*, *51*, *10*] *10*)

translations
CSUM i:*r*. *rs* == *CONST Csum r* (%*i*. *rs*)

lemma *card-of-UNION-csum*[*simp*]:
$|\bigcup i\in I.\ A\ i| \leq o$ (*CSUM i* : $|I|$. $|A\ i|$)
unfolding *Csum-def* **by** (*simp add*: *card-of-UNION-Sigma*)

lemma *SIGMA-CSUM*[*simp*]:
$|SIGMA\ i : I.\ As\ i| = (CSUM\ i : |I|.\ |As\ i|\ )$
unfolding *Csum-def* **by** *simp*

9.7 Product

definition *cprod* (**infixr** $*c$ *80*) **where**
$r1 *c\ r2 = |Field\ r1 <*> Field\ r2|$

lemma *Times-cprod*: $|A \times B| =o\ |A| *c\ |B|$
unfolding *cprod-def* **by** (*simp only*: *Field-card-of card-of-refl*)

lemma *card-order-cprod*[*simp*]:
assumes *card-order r1 card-order r2*
shows *card-order* ($r1 *c\ r2$)
proof −
have *Field r1* = *UNIV Field r2* = *UNIV* **using** *assms card-order-on-Card-order*
by *auto*
thus *?thesis* **unfolding** *cprod-def* **by** *auto*
qed

lemma *Card-order-cprod*[*simp*]:
Card-order ($r1 *c\ r2$)
unfolding *cprod-def* **by** (*simp only*: *Field-card-of card-of-card-order-on*)

lemma *cprod-cong*[*simp*]:
assumes $p1 =o\ r1$ **and** $p2 =o\ r2$
shows $p1 *c\ p2 =o\ r1 *c\ r2$
unfolding *cprod-def* **by** (*simp only*: *assms ordIso-Times-cong*)

lemma *cprod-cong1*[*simp*]:
assumes $p1 =o\ r1$
shows $p1 *c\ q =o\ r1 *c\ q$
unfolding *cprod-def* **by** (*simp only*: *assms ordIso-Times-cong1*)

lemma *cprod-cong2*[*simp*]:
assumes $p2 =o\ r2$
shows $q *c\ p2 =o\ q *c\ r2$
unfolding *cprod-def* **by** (*simp only*: *assms ordIso-Times-cong2*)

lemma *cprod-mono*[*simp*]:
assumes $p1 \leq o\ r1$ **and** $p2 \leq o\ r2$
shows $p1 *c\ p2 \leq o\ r1 *c\ r2$
unfolding *cprod-def* **by** (*simp only*: *assms ordLeq-Times-mono*)

lemma *cprod-mono1*[*simp*]:
assumes $p1 \leq o\ r1$
shows $p1 *c\ q \leq o\ r1 *c\ q$
unfolding *cprod-def* **by** (*simp only*: *assms ordLeq-Times-mono1*)

lemma *cprod-mono2*[*simp*]:
  assumes *p2* ≤*o* *r2*
  shows *q* ∗*c* *p2* ≤*o* *q* ∗*c* *r2*
unfolding *cprod-def* **by** (*simp only*: *assms ordLeq-Times-mono2*)

lemma *ordLeq-cprod1*[*simp*]:
  assumes *Card-order p1 Cnotzero p2*
  shows *p1* ≤*o* *p1* ∗*c* *p2*
proof −
  from *assms*(*2*) **have** ∗: |*Field p2*| =*o* *p2* **by** *simp*
  { **assume** |*Field p2*| = *czero*
    with ∗ **have** *czero* =*o* *p2* **using** *czero-ordIso ordIso-transitive* **by** *fastforce*
    hence *p2* =*o* *czero* **using** *ordIso-symmetric* **by** *blast*
    with *assms*(*2*) **have** *False* **by** *blast*
  }
  hence |*Field p2*| ≠ *czero* **by** *blast*
  hence *Field p2* ≠ {} **by** (*simp add*: *card-of-czero-iff-empty*)
  with *assms*(*1*) **show** *?thesis* **unfolding** *cprod-def* **by** (*simp add*: *Card-order-Times1 del*: *SIGMA-CSUM*)
qed

lemma *ordLeq-cprod1'*[*simp*]:
  assumes *Card-order r A* ≠ {}
  shows *r* ≤*o* *r* ∗*c* |*A*|
proof −
  from *assms*(*2*) **have** *Card-order* |*A*| ¬ (|*A*| =*o* *czero*) **by** (*auto simp add*: *card-of-empty2*)+
  with *assms*(*1*) **show** *?thesis* **using** *ordLeq-cprod1* **by** *blast*
qed

lemma *ordLeq-cprod2*[*simp*]:
  assumes *Cnotzero p1 Card-order p2*
  shows *p2* ≤*o* *p1* ∗*c* *p2*
proof −
  from *assms*(*1*) **have** ∗: |*Field p1*| =*o* *p1* **by** *simp*
  { **assume** |*Field p1*| = *czero*
    with ∗ **have** *czero* =*o* *p1* **using** *czero-ordIso ordIso-transitive* **by** *fastforce*
    hence *p1* =*o* *czero* **using** *ordIso-symmetric* **by** *blast*
    with *assms*(*1*) **have** *False* **by** *blast*
  }
  hence |*Field p1*| ≠ *czero* **by** *blast*
  hence *Field p1* ≠ {} **by** (*simp add*: *card-of-czero-iff-empty*)
  with *assms*(*2*) **show** *?thesis* **unfolding** *cprod-def* **by** (*simp add*: *Card-order-Times2 del*: *SIGMA-CSUM*)
qed

lemma *ordLeq-cprod2'*[*simp*]:
  assumes *Card-order r A* ≠ {}

shows *r* $\leq$*o* |*A*| $*$*c* *r*
proof −
from *assms*(*2*) **have** *Card-order* |*A*| ¬ (|*A*| =*o* *czero*) **by** (*auto simp add*: *card-of-empty2*)+
with *assms*(*1*) **show** *?thesis* **using** *ordLeq-cprod2* **by** *blast*
qed

lemma *cinfinite-cprod*[*simp*]:
⟦*cinfinite r1*; *cinfinite r2*⟧ $\Longrightarrow$ *cinfinite* (*r1* $*$*c* *r2*)
unfolding *cinfinite-def cprod-def* **by** (*simp del*: *SIGMA-CSUM*)

lemma *Cinfinite-cprod*[*simp*]:
⟦*Cinfinite r1*; *Cinfinite r2*⟧ $\Longrightarrow$ *Cinfinite* (*r1* $*$*c* *r2*)
unfolding *cinfinite-def cprod-def* **by** (*simp del*: *SIGMA-CSUM*)

lemma *cinfinite-cprod1*:
assumes *Cinfinite r1 Cnotzero r2*
shows *cinfinite* (*r1* $*$*c* *r2*)
proof −
from *assms* **have** *r1* $\leq$*o* *r1* $*$*c* *r2* **by** (*auto intro*: *ordLeq-cprod1*)
with *assms*(*1*) **show** *?thesis* **unfolding** *cinfinite-def cprod-def*
using *card-of-mono2 card-of-ordLeq-infinite* **by** *auto*
qed

lemma *Cinfinite-cprod1*:
⟦*Cinfinite r1*; *Cnotzero r2*⟧ $\Longrightarrow$ *Cinfinite* (*r1* $*$*c* *r2*)
by (*blast intro*: *cinfinite-cprod1 Card-order-cprod*)

lemma *cinfinite-cprod2*:
assumes *Cnotzero r1 Cinfinite r2*
shows *cinfinite* (*r1* $*$*c* *r2*)
proof −
from *assms* **have** *r2* $\leq$*o* *r1* $*$*c* *r2* **by** (*auto intro*: *ordLeq-cprod2*)
with *assms*(*2*) **show** *?thesis* **unfolding** *cinfinite-def cprod-def*
using *card-of-mono2 card-of-ordLeq-infinite* **by** *auto*
qed

lemma *Cinfinite-cprod2*:
⟦*Cnotzero r1*; *Cinfinite r2*⟧ $\Longrightarrow$ *Cinfinite* (*r1* $*$*c* *r2*)
by (*blast intro*: *cinfinite-cprod2 Card-order-cprod*)

lemma *Cnotzero-cprod*:
assumes *r1*: *Cnotzero* (*r1* :: ′*a rel*) **and** *r2*: *Cnotzero* (*r2* :: ′*b rel*)
shows *Cnotzero* (*r1* $*$*c* *r2*)
proof (*intro conjI*, *rule ccontr*)
assume ¬ (*r1* $*$*c* *r2*, *czero* :: (′*a* × ′*b*) *rel*) ∉ *ordIso*
hence *r1* $*$*c* *r2* =*o* (*czero* :: (′*a* × ′*b*) *rel*) **by** *blast*
hence *Field r1* <$*$> *Field r2* = ({} :: (′*a* × ′*b*) *set*) **unfolding** *cprod-def*
using *iffD1*[*OF card-of-ordIso-czero-iff-empty*] **by** *blast*

hence *Field r1* = {} ∨ *Field r2* = {} **by** *auto*
thus *False*
using *Cnotzero-imp-not-empty*[*OF r1*] *Cnotzero-imp-not-empty*[*OF r2*] **by** *simp*
qed *simp*

lemma *cprod-com*:
p1 ∗*c p2* =*o p2* ∗*c p1*
unfolding *cprod-def* **by** (*simp only*: *card-of-Times-commute*)

lemma *cprod-assoc*:
(*p1* ∗*c p2*) ∗*c p3* =*o p1* ∗*c p2* ∗*c p3*
unfolding *cprod-def* **by** (*simp only*: *Field-card-of card-of-Times-assoc*)

lemma *Prod-cprod*[*simp*]:
|*A* <∗> *B*| =*o* |*A*| ∗*c* |*B*|
unfolding *cprod-def* **by**(*simp only*: *Field-card-of card-of-refl*)

lemma *czero-cprod*[*simp*]:
czero ∗*c r* =*o czero*
unfolding *cprod-def czero-def* **by** (*simp del*: *empty-czero add*: *card-of-empty-ordIso*)

lemma *cprod-czero*[*simp*]:
r ∗*c czero* =*o czero*
unfolding *cprod-def czero-def* **by** (*simp del*: *empty-czero add*: *card-of-empty-ordIso*)

lemma *cone-cprod*[*simp*]:
assumes *Card-order r*
shows *cone* ∗*c r* =*o r*
proof −
have |{()} <∗> *Field r*| =*o r* **by** (*simp only*: *Card-order-Times-singl2 assms ordIso-symmetric*)
thus *?thesis* **unfolding** *cprod-def cone-def* **by** (*simp only*: *Field-card-of*)
qed

lemma *cprod-cone*[*simp*]:
assumes *Card-order r*
shows *r* ∗*c cone* =*o r*
proof −
have |*Field r* <∗> {()}| =*o r* **by** (*simp only*: *Card-order-Times-singl1 assms ordIso-symmetric*)
thus *?thesis* **unfolding** *cprod-def cone-def* **by** (*simp only*: *Field-card-of*)
qed

lemma *card-of-Csum-Times*:
$\forall i \in I$. |*A i*| ≤*o* |*B*| $\Longrightarrow$ (*CSUM i* : |*I*|. |*A i*|) ≤*o* |*I*| ∗*c* |*B*|
unfolding *Csum-def cprod-def* **using** *card-of-Sigma-Times* **by** *simp*

lemma *card-of-Csum-Times'*:
assumes *Card-order r* $\forall i \in I$. |*A i*| ≤*o r*

shows $(CSUM\ i : |I|.\ |A\ i|\ ) \leq o\ |I|\ {*c}\ r$
proof −
from *assms*(*1*) **have** ∗: $r =o\ |Field\ r|$ **by** *simp*
with *assms*(*2*) **have** $\forall\, i \in I.\ |A\ i| \leq o\ |Field\ r|$ **by** (*blast intro*: *ordLeq-ordIso-trans*)
hence $(CSUM\ i : |I|.\ |A\ i|\ ) \leq o\ |I|\ {*c}\ |Field\ r|$ **by** (*simp only*: *card-of-Csum-Times*)
also from ∗ **have** $|I|\ {*c}\ |Field\ r| \leq o\ |I|\ {*c}\ r$ **by** *simp*
finally show *?thesis* **.**
qed

lemma *cprod-csum-distrib1*:
$r1\ {*c}\ r2\ {+c}\ r1\ {*c}\ r3 =o\ r1\ {*c}\ (r2\ {+c}\ r3)$
unfolding *csum-def cprod-def*
by (*simp add*: *card-of-Times-Plus-distrib ordIso-symmetric del*: *SIGMA-CSUM*)

lemma *cprod-csum-distrib2*:
$r1\ {*c}\ r3\ {+c}\ r2\ {*c}\ r3 =o\ (r1\ {+c}\ r2)\ {*c}\ r3$
by (*blast intro*: *cprod-com cprod-csum-distrib1 csum-cong ordIso-transitive*)

lemma *csum-cprod-ordLeq*:
assumes *Cnotzero r1 Card-order r2*
shows $r1\ {+c}\ r2 \leq o\ r1\ {*c}\ (cone\ {+c}\ r2)$
proof −
from *assms*(*1*) **have** $r1 \leq o\ r1\ {*c}\ cone$
using *Card-order-cone cone-Cnotzero* **by** (*blast intro*: *ordLeq-cprod1*)
moreover from *assms* **have** $r2 \leq o\ r1\ {*c}\ r2$ **by** (*blast intro*: *ordLeq-cprod2*)
ultimately have $r1\ {+c}\ r2 \leq o\ r1\ {*c}\ cone\ {+c}\ r1\ {*c}\ r2$ **by** (*rule csum-mono*)
also have $r1\ {*c}\ cone\ {+c}\ r1\ {*c}\ r2 =o\ r1\ {*c}\ (cone\ {+c}\ r2)$ **by** (*rule cprod-csum-distrib1*)
finally show *?thesis* **.**
qed

lemma *csum-absorb2'*:
assumes *card*: *Card-order r2*
and *r12*: $r1 \leq o\ r2$ **and** *cr12*: $cinfinite\ r1 \lor cinfinite\ r2$
shows $r1\ {+c}\ r2 =o\ r2$
proof −
have *infinite* (*Field r2*)
using *assms card-of-mono2 card-of-ordLeq-infinite*
unfolding *csum-def cinfinite-def* **by** *auto*
hence $r1\ {+c}\ r2 =o\ |Field\ r2|$
using *card-of-Plus-infinite2 card-of-mono2 r12* **unfolding** *csum-def* **by** *auto*
thus *?thesis* **using** *card card-of-Field-ordIso ordIso-transitive* **by** *auto*
qed

lemma *csum-absorb2*:
$[\![Cinfinite\ r2;\ r1 \leq o\ r2]\!] \Longrightarrow r1\ {+c}\ r2 =o\ r2$
by (*rule csum-absorb2'*) *auto*

lemma *csum-absorb1'*:
assumes *card*: *Card-order r2*

and *r12*: *r1* $\leq o$ *r2* **and** *cr12*: *cinfinite r1* $\vee$ *cinfinite r2*
shows *r2* $+c$ *r1* $=o$ *r2*
by (*rule ordIso-transitive*, *rule csum-com*, *rule csum-absorb2'*, (*simp only*: *assms*)+)

lemma *csum-absorb1*:
⟦*Cinfinite r2*; *r1* $\leq o$ *r2*⟧ $\Longrightarrow$ *r2* $+c$ *r1* $=o$ *r2*
by (*rule csum-absorb1'*) *auto*

lemma *cprod-infinite1*:
⟦*Cinfinite r*; $A \neq \{\}$; $|A| \leq o\ r$⟧ $\Longrightarrow$ $r *c\ |A| =o\ r$
unfolding *cinfinite-def cprod-def*
by (*rule Card-order-Times-infinite*[*THEN conjunct1*]) *auto*

lemma *cprod-infinite1'*:
⟦*Cinfinite r*; *Cnotzero p*; $p \leq o\ r$⟧ $\Longrightarrow$ $r *c\ p =o\ r$
unfolding *cinfinite-def cprod-def*
by (*rule Card-order-Times-infinite*[*THEN conjunct1*]) (*blast intro*: *czeroI*)+

lemma *cprod-infinite1-natLeq*:
Cinfinite r $\Longrightarrow$ $r *c$ *natLeq* $=o\ r$
unfolding *cprod-def*
by (*rule Card-order-Times-infinite*[*THEN conjunct1*])
(*auto simp add*: *natLeq-ordLeq-cinfinite cinfinite-def Field-natLeq*)

lemma *cprod-infinite2*:
⟦*Cinfinite r*; $A \neq \{\}$; $|A| \leq o\ r$⟧ $\Longrightarrow$ $|A| *c\ r =o\ r$
unfolding *cinfinite-def cprod-def*
by (*rule Card-order-Times-infinite*[*THEN conjunct2*]) *auto*

lemma *cprod-infinite2'*:
⟦*Cinfinite r*; *Cnotzero p*; $p \leq o\ r$⟧ $\Longrightarrow$ $p *c\ r =o\ r$
unfolding *cinfinite-def cprod-def*
by (*rule Card-order-Times-infinite*[*THEN conjunct2*]) (*blast intro*: *czeroI*)+

lemma *cprod-infinite2-natLeq*:
Cinfinite r $\Longrightarrow$ *natLeq* $*c\ r =o\ r$
unfolding *cprod-def*
by (*rule Card-order-Times-infinite*[*THEN conjunct2*])
(*auto simp add*: *natLeq-ordLeq-cinfinite cinfinite-def Field-natLeq*)

9.8 Exponentiation

definition *cexp* (**infixr** *^c 80*) **where**
r1 ^c r2 $\equiv$ |*Func* (*Field r2*) (*Field r1*)|

definition *ccexp* (**infixr** *^^c 80*) **where**
r1 ^^c r2 $\equiv$ |*Pfunc* (*Field r2*) (*Field r1*)|

lemma *cexp-ordLeq-ccexp*:

$r1$ ^c $r2$ $\leq o$ $r1$ ^^c $r2$
unfolding *cexp-def ccexp-def* **by** (*rule card-of-mono1*) (*rule Func-Pfunc*)

lemma *card-order-ccexp*:
assumes *card-order r1 card-order r2*
shows *card-order* (*r1* ^^c *r2*)
proof −
have *Field r1* = *UNIV Field r2* = *UNIV* **using** *assms card-order-on-Card-order* **by** *auto*
thus *?thesis* **unfolding** *ccexp-def Pfunc-def* **by** (*auto split*: *option.split*)
qed

lemma *Card-order-cexp*: *Card-order* (*r1* ^c *r2*)
unfolding *cexp-def* **by** *simp*

lemma *Card-order-ccexp*: *Card-order* (*r1* ^^c *r2*)
unfolding *ccexp-def* **by** *simp*

lemma *cexp-mono'*:
assumes *1*: $p1 \leq o$ $r1$ **and** *2*: $p2 \leq o$ $r2$
and *n1*: *Field p1* $\neq \{\} \vee$ *cone* $\leq o$ *r1* ^c *r2*
and *n2*: *Field p2* $= \{\} \Longrightarrow$ *Field r2* $= \{\}$
shows *p1* ^c *p2* $\leq o$ *r1* ^c *r2*
proof(*cases Field p1* = {})
case *True*
hence |*Field* (*p1* ^c *p2*)| $\leq o$ *cone*
unfolding *cone-def cexp-def Field-card-of* **by** (*cases Field p2* = {}) *auto*
hence *p1* ^c *p2* $\leq o$ *cone* **by** (*simp add*: *cexp-def*)
thus *?thesis* **using** *True n1 ordLeq-transitive* **by** *auto*
next
case *False*
have *1*: |*Field p1*| $\leq o$ |*Field r1*| **and** *2*: |*Field p2*| $\leq o$ |*Field r2*|
using *1 2* **by** *auto*
obtain *f1* **where** *f1*: *f1* ‘ *Field r1* = *Field p1*
using *1* **unfolding** *card-of-ordLeq2*[*OF False*, *symmetric*] **by** *auto*
obtain *f2* **where** *f2*: *inj-on f2* (*Field p2*) *f2* ‘ *Field p2* $\subseteq$ *Field r2*
using *2* **unfolding** *card-of-ordLeq*[*symmetric*] **by** *blast*
have *0*: *Func-map* (*Field p2*) *f1 f2* ‘ (*Field* (*r1* ^c *r2*)) = *Field* (*p1* ^c *p2*)
unfolding *cexp-def Field-card-of* **using** *Func-map-surj*[*OF f1 f2 n2*, *symmetric*]
.
have *00*: *Field* (*p1* ^c *p2*) $\neq$ {} **unfolding** *cexp-def Field-card-of Func-is-emp*
using *False* **by** *simp*
show *?thesis*
using *0 card-of-ordLeq2*[*OF 00*] **unfolding** *cexp-def Field-card-of* **by** *blast*
qed

lemma *cexp-mono*[*simp*]:
assumes *1*: *p1* $\leq o$ *r1* **and** *2*: *p2* $\leq o$ *r2*
and *n1*: *Cnotzero p1* $\vee$ *cone* $\leq o$ *r1* ^c *r2*

and *n2*: *p2* =*o* *czero* ⟹ *r2* =*o* *czero* **and** *card*: *Card-order p2*
shows *p1* ^*c* *p2* ≤*o* *r1* ^*c* *r2*
proof (*rule cexp-mono'*[*OF 1 2*])
show *Field p1* ≠ {} ∨ *cone* ≤*o* *r1* ^*c* *r2*
proof (*cases Cnotzero p1*)
case *True* **show** *?thesis* **using** *Cnotzero-imp-not-empty*[*OF True*] **by** (*rule disjI1*)
next
case *False* **with** *n1* **show** *?thesis* **by** *blast*
qed
qed (*rule czeroI*[*OF card*, *THEN n2*, *THEN czeroE*])

lemma *cexp-mono1'*:
assumes *1*: *p1* ≤*o* *r1*
and *n1*: *Field p1* ≠ {} ∨ *cone* ≤*o* *r1* ^*c* *q* **and** *q*: *Card-order q*
shows *p1* ^*c* *q* ≤*o* *r1* ^*c* *q*
using *ordLeq-refl*[*OF q*] **by** (*rule cexp-mono'*[*OF 1 - n1*]) *auto*

lemma *cexp-mono1*[*simp*]:
assumes *1*: *p1* ≤*o* *r1*
and *n1*: *Cnotzero p1* ∨ *cone* ≤*o* *r1* ^*c* *q* **and** *q*: *Card-order q*
shows *p1* ^*c* *q* ≤*o* *r1* ^*c* *q*
using *ordLeq-refl*[*OF q*] **by** (*rule cexp-mono*[*OF 1 - n1*]) (*auto simp*: *q*)

lemma *cexp-mono2'*:
assumes *2*: *p2* ≤*o* *r2* **and** *q*: *Card-order q*
and *n1*: *Field q* ≠ {} ∨ *cone* ≤*o* *q* ^*c* *r2*
and *n2*: *Field p2* = {} ⟹ *Field r2* = {}
shows *q* ^*c* *p2* ≤*o* *q* ^*c* *r2*
using *ordLeq-refl*[*OF q*] **by** (*rule cexp-mono'*[*OF - 2 n1 n2*]) *auto*

lemma *cexp-mono2*[*simp*]:
assumes *2*: *p2* ≤*o* *r2* **and** *q*: *Card-order q*
and *n1*: *Cnotzero q* ∨ *cone* ≤*o* *q* ^*c* *r2*
and *n2*: *p2* =*o* *czero* ⟹ *r2* =*o* *czero* **and** *card*: *Card-order p2*
shows *q* ^*c* *p2* ≤*o* *q* ^*c* *r2*
using *ordLeq-refl*[*OF q*] **by** (*rule cexp-mono*[*OF - 2 n1 n2 card*]) *auto*

lemma *cexp-cong'*:
assumes *1*: *p1* =*o* *r1* **and** *2*: *p2* =*o* *r2*
and *p1*: *Field p1* ≠ {} ∨ *cone* ≤*o* *r1* ^*c* *r2*
and *r1*: *Field r1* ≠ {} ∨ *cone* ≤*o* *p1* ^*c* *p2*
shows *p1* ^*c* *p2* =*o* *r1* ^*c* *r2*
proof −
obtain *f* **where** *bij-betw f* (*Field p2*) (*Field r2*)
using *2 card-of-ordIso*[*of Field p2 Field r2*] *card-of-cong* **by** *auto*
hence *0*: *Field p2* = {} ⟷ *Field r2* = {} **unfolding** *bij-betw-def* **by** *auto*
show *?thesis* **using** *0 1 2 cexp-mono'*[*OF - - p1*] *cexp-mono'*[*OF - - r1*]
unfolding *ordIso-iff-ordLeq* **by** *auto*

qed

lemma *cexp-cong*[*simp*]:
  assumes *1*: *p1* =*o r1* **and** *2*: *p2* =*o r2*
  and *p1*: *Cnotzero p1* ∨ *cone* ≤*o r1* ^*c r2* **and** *Cr*: *Card-order r2*
  and *r1*: *Cnotzero r1* ∨ *cone* ≤*o p1* ^*c p2* **and** *Cp*: *Card-order p2*
  shows *p1* ^*c p2* =*o r1* ^*c r2*
proof −
  obtain *f* **where** *bij-betw f* (*Field p2*) (*Field r2*)
    using *2 card-of-ordIso*[*of Field p2 Field r2*] *card-of-cong* **by** *auto*
  hence *0*: *Field p2* = {} ⟷ *Field r2* = {} **unfolding** *bij-betw-def* **by** *auto*
  have *r*: *p2* =*o czero* ⟹ *r2* =*o czero*
    and *p*: *r2* =*o czero* ⟹ *p2* =*o czero*
     using *0 Cr Cp czeroE czeroI* **by** *auto*
  show *?thesis* **using** *0 1 2* **unfolding** *ordIso-iff-ordLeq*
    using *r p cexp-mono*[*OF - - p1 - Cp*] *cexp-mono*[*OF - - r1 - Cr*]
    by *blast*
qed

lemma *cexp-cong1'*:
  assumes *1*: *p1* =*o r1* **and** *q*: *Card-order q*
  and *p1*: *Field p1* ≠ {} ∨ *cone* ≤*o r1* ^*c q*
  and *r1*: *Field r1* ≠ {} ∨ *cone* ≤*o p1* ^*c q*
  shows *p1* ^*c q* =*o r1* ^*c q*
by (*rule cexp-cong'*[*OF 1 - p1 r1*]) (*rule ordIso-refl*[*OF q*])

lemma *cexp-cong1*[*simp*]:
  assumes *1*: *p1* =*o r1* **and** *q*: *Card-order q*
  and *p1*: *Cnotzero p1* ∨ *cone* ≤*o r1* ^*c q*
  and *r1*: *Cnotzero r1* ∨ *cone* ≤*o p1* ^*c q*
  shows *p1* ^*c q* =*o r1* ^*c q*
by (*rule cexp-cong*[*OF 1 - p1 q r1 q*]) (*rule ordIso-refl*[*OF q*])

lemma *cexp-cong2'*:
  assumes *2*: *p2* =*o r2* **and** *q*: *Card-order q*
  shows *Field q* ≠ {} ∨ (*cone* ≤*o q* ^*c p2* ∧ *cone* ≤*o q* ^*c r2*) ⟹
    q ^*c p2* =*o q* ^*c r2*
by (*rule cexp-cong'*[*OF - 2*]) (*auto simp only*: *ordIso-refl q*)

lemma *cexp-cong2*[*simp*]:
  assumes *2*: *p2* =*o r2* **and** *q*: *Card-order q*
  and *p*: *Card-order p2* **and** *r*: *Card-order r2*
  shows *Cnotzero q* ∨ (*cone* ≤*o q* ^*c p2* ∧ *cone* ≤*o q* ^*c r2*) ⟹
    q ^*c p2* =*o q* ^*c r2*
by (*rule cexp-cong*[*OF - 2*]) (*auto simp only*: *ordIso-refl q p r*)

lemma *cexp-czero*:
  r ^*c czero* =*o cone*
unfolding *cexp-def czero-def Field-card-of Func-emp-empty* **by** (*rule single-cone*)

lemma *czero-cexp*:
  Cnotzero r $\Longrightarrow$ *czero ^c r =o czero*
unfolding *cexp-def czero-def Field-card-of*
using *Func-emp2*[*of Field r*] *card-of-empty-ordIso card-of-unique*[*of Field r r*]
by *fastforce*

lemma *cexp-cone*:
  assumes *Card-order r*
  shows *r ^c cone =o r*
proof –
  have *r ^c cone =o* |*Field r*|
    unfolding *cexp-def cone-def Field-card-of Func-emp-empty*
      card-of-ordIso[*symmetric*] *bij-betw-def Func-def inj-on-def image-def*
    by (*rule exI*[*of - λf. case f* () *of Some a* ⇒ *a*]) *auto*
  also have |*Field r*| *=o r* **by** (*rule card-of-Field-ordIso*[*OF assms*])
  finally show *?thesis* **.**
qed

lemma *cexp-cprod*:
  assumes *r1*: *Cnotzero r1*
  shows (*r1 ^c r2*) *^c r3 =o r1 ^c* (*r2 ∗c r3*) (**is** *?L =o ?R*)
proof –
  have *?L =o r1 ^c* (*r3 ∗c r2*)
    unfolding *cprod-def cexp-def Field-card-of*
    using *card-of-Func-Times* **by**(*rule ordIso-symmetric*)
  also have *r1 ^c* (*r3 ∗c r2*) *=o ?R*
    apply(*rule cexp-cong2*) **using** *cprod-com r1* **by** *auto*
  finally show *?thesis* **.**
qed

lemma *cexp-cprod-ordLeq*:
  assumes *r1*: *Cnotzero r1* **and** *r2*: *Cinfinite r2*
  and *r3*: *Cnotzero r3 r3 ≤o r2*
  shows (*r1 ^c r2*) *^c r3 =o r1 ^c r2* (**is** *?L =o ?R*)
proof–
  have *?L =o r1 ^c* (*r2 ∗c r3*) **using** *cexp-cprod*[*OF r1*] **.**
  also have *r1 ^c* (*r2 ∗c r3*) *=o ?R*
  apply(*rule cexp-cong2*)
  apply(*rule cprod-infinite1'*[*OF r2 r3*]) **using** *r1 r2* **by** *fastforce+*
  finally show *?thesis* **.**
qed

lemma *cexp-cprod-natLeq*:
  assumes *r1*: *Cnotzero r1*
  and *r2*: *Cinfinite r2*
  shows (*r1 ^c r2*) *^c natLeq =o r1 ^c r2* (**is** *?L =o ?R*)
proof–
  have *?L =o r1 ^c* (*r2 ∗c natLeq*) **using** *cexp-cprod*[*OF r1*] **.**

also have *r1* ^*c* (*r2* ∗*c natLeq*) =*o ?R*
apply(*rule cexp-cong2*)
apply(*rule cprod-infinite1-natLeq*[*OF r2*]) **using** *r1 r2* **by** *fastforce*+
finally show *?thesis* **.**
qed

lemma *Cnotzero-UNIV*[*simp*]: *Cnotzero* |*UNIV*|
by (*simp*, *rule notI*, *drule czeroE*, *auto*)

lemma *card-of-Pfunc-Func*:
assumes *A*: *infinite A* **and** *B*: {*b1*,*b2*} ⊆ *B b1* ≠ *b2*
shows |*Pfunc A B*| =*o* |*Func A B*|
unfolding *ordIso-iff-ordLeq* **proof**
show |*Func A B*| ≤*o* |*Pfunc A B*| **using** *Func-Pfunc* **by** (*rule card-of-mono1*)
next
have *B'*: *B* ≠ {} **using** *B* **by** *auto*
have *B''*: |*UNIV*::*bool set*| ≤*o* |*B*|
using *ordIso-ordLeq-trans*[*OF card-of-bool*[*OF B*(*2*)] *card-of-mono1*[*OF B*(*1*)]]
.
have |*Pfunc A B*| ≤*o* |*Pow A* × *Func A B*| **by** (*rule card-of-Pfunc-Pow-Func*[*OF B'*])
also have |*Pow A* × *Func A B*| =*o* |*Func A* (*UNIV* :: *bool set*) × *Func A B*|
by (*rule card-of-Times-cong1*[*OF card-of-Pow-Func*])
also have |*Func A* (*UNIV* :: *bool set*) × *Func A B*| =*o* |*Func A B*|
proof (*rule card-of-Times-infinite*[*THEN conjunct2*, *OF infinite-Func*[*OF A B*]])
show *Func A* (*UNIV* :: *bool set*) ≠ {} **using** *A* **unfolding** *Func-is-emp* **by** *simp*
next
show |*Func A* (*UNIV* :: *bool set*)| ≤*o* |*Func A B*|
using *cexp-mono1*[*OF B''*, *of* |*A*|, *OF - card-of-Card-order*]
unfolding *cexp-def Field-card-of* **by** *auto*
qed
finally show |*Pfunc A B*| ≤*o* |*Func A B*| **.**
qed

lemma *Pow-cexp-ctwo*:
|*Pow A*| =*o ctwo* ^*c* |*A*|
unfolding *ctwo-def cexp-def Field-card-of* **by** (*rule card-of-Pow-Func*)

lemma *ordLess-ctwo-cexp*:
assumes *Card-order r*
shows *r* <*o ctwo* ^*c r*
proof −
have *r* <*o* |*Pow* (*Field r*)| **using** *assms* **by** *simp*
also have |*Pow* (*Field r*)| =*o ctwo* ^*c r*
unfolding *ctwo-def cexp-def Field-card-of* **by** (*rule card-of-Pow-Func*)
finally show *?thesis* **.**
qed

lemma *ordLeq-cexp1*:
  assumes *Cnotzero r Card-order q*
  shows $q \leq o\ q\ \hat{}c\ r$
proof (*cases* $q = o$ (*czero* :: $'a$ *rel*))
  case *True* **thus** *?thesis* **by** (*simp add*: *ordIso-ordLeq-trans Card-order-cexp*)
next
  case *False*
  thus *?thesis*
    apply −
    apply (*rule ordIso-ordLeq-trans*)
    apply (*rule ordIso-symmetric*)
    apply (*rule cexp-cone*)
    apply (*rule assms*(*2*))
    apply (*rule cexp-mono2*)
    apply (*rule cone-ordLeq-Cnotzero*)
    apply (*rule assms*(*1*))
    apply (*rule assms*(*2*))
    apply (*rule disjI1*)
    apply (*rule conjI*)
    apply (*rule notI*)
    apply (*erule notE*)
    apply (*rule ordIso-transitive*)
    apply *assumption*
    apply (*rule czero-ordIso*)
    apply (*rule assms*(*2*))
    apply (*rule notE*)
    apply (*rule cone-not-czero*)
    apply *assumption*
    apply (*rule Card-order-cone*)
  done
qed

lemma *ordLeq-cexp2*:
  assumes *ctwo* $\leq o$ *q Card-order r*
  shows $r \leq o\ q\ \hat{}c\ r$
proof (*cases* $r = o$ (*czero* :: $'a$ *rel*))
  case *True* **thus** *?thesis* **by** (*simp add*: *ordIso-ordLeq-trans Card-order-cexp*)
next
  case *False* **thus** *?thesis*
    apply −
    apply (*rule ordLess-imp-ordLeq*)
    apply (*rule ordLess-ordLeq-trans*)
    apply (*rule ordLess-ctwo-cexp*)
    apply (*rule assms*(*2*))
    apply (*rule cexp-mono1*)
    apply (*rule assms*(*1*))
    apply (*rule disjI1*)
    apply (*rule ctwo-Cnotzero*)
    apply (*rule assms*(*2*))

done
qed

lemma *Cnotzero-cexp*:
assumes *Cnotzero q Card-order r*
shows *Cnotzero* (*q ^c r*)
proof (*cases r =o czero*)
case *False* **thus** *?thesis*
apply −
apply (*rule Cnotzero-mono*)
apply (*rule assms*(*1*))
apply (*rule Card-order-cexp*)
apply (*rule ordLeq-cexp1*)
apply (*rule conjI*)
apply (*rule notI*)
apply (*erule notE*)
apply (*rule ordIso-transitive*)
apply *assumption*
apply (*rule czero-ordIso*)
apply (*rule assms*(*2*))
apply (*rule conjunct2*)
apply (*rule assms*(*1*))
done
next
case *True* **thus** *?thesis*
apply −
apply (*rule Cnotzero-mono*)
apply (*rule cone-Cnotzero*)
apply (*rule Card-order-cexp*)
apply (*rule ordIso-imp-ordLeq*)
apply (*rule ordIso-symmetric*)
apply (*rule ordIso-transitive*)
apply (*rule cexp-cong2*)
apply *assumption*
apply (*rule conjunct2*)
apply (*rule assms*(*1*))
apply (*rule assms*(*2*))
apply (*simp only*: *card-of-Card-order czero-def*)
apply (*rule disjI1*)
apply (*rule assms*(*1*))
apply (*rule cexp-czero*)
done
qed

lemma *cinfinite-ctwo-cexp*:
cinfinite r $\Longrightarrow$ *cinfinite* (*ctwo ^c r*)
unfolding *ctwo-def cexp-def cinfinite-def Field-card-of*
by (*rule infinite-Func*) *auto*

lemma *Cinfinite-ctwo-cexp*:
  Cinfinite r $\Longrightarrow$ *Cinfinite* (*ctwo* ^*c r*)
unfolding *ctwo-def cexp-def cinfinite-def Field-card-of*
by (*rule conjI*, *rule infinite-Func*) *auto*

lemma *cinfinite-cexp*:
  assumes *q*: *ctwo* $\leq$*o q* **and** *r*: *Cinfinite r*
  shows *cinfinite* (*q* ^*c r*)
proof −
  have *ctwo* ^*c r* $\leq$*o q* ^*c r* **by** (*rule cexp-mono1*[*OF q*]) (*auto simp add*: *r ctwo-def*)
  thus *?thesis* **using** *Cinfinite-ctwo-cexp*[*OF r*]
  unfolding *cinfinite-def* **using** *card-of-ordLeq-infinite card-of-mono2* **by** *blast*
qed

lemma *cinfinite-ccexp*:
  [[*ctwo* $\leq$*o q*; *Cinfinite r*]] $\Longrightarrow$ *cinfinite* (*q* ^^*c r*)
using *cinfinite-mono*[*OF cexp-ordLeq-ccexp cinfinite-cexp*] **by** *auto*

lemma *Cinfinite-cexp*:
  [[*ctwo* $\leq$*o q*; *Cinfinite r*]] $\Longrightarrow$ *Cinfinite* (*q* ^*c r*)
by (*simp add*: *cinfinite-cexp Card-order-cexp*)

lemma *Cinfinite-ccexp*:
  [[*ctwo* $\leq$*o q*; *Cinfinite r*]] $\Longrightarrow$ *Cinfinite* (*q* ^^*c r*)
using *Cinfinite-mono*[*OF cexp-ordLeq-ccexp Cinfinite-cexp*]
by (*auto simp add*: *Card-order-ccexp*)

lemma *ctwo-ordLeq-natLeq*:
  ctwo $\leq$*o natLeq*
proof −
  have *ctwo* $\leq$*o* |*Field natLeq*|
    unfolding *Field-natLeq ctwo-def inj-on-def card-of-ordLeq*[*symmetric*]
    by (*rule exI*[*of* - λ *c. if c then 0 else Suc 0*]) *auto*
  thus *?thesis* **using** *card-of-Field-natLeq ordLeq-ordIso-trans* **by** *blast*
qed

lemma *ctwo-ordLess-natLeq*:
  ctwo <*o natLeq*
unfolding *ctwo-def* **using** *finite-iff-ordLess-natLeq finite-UNIV* **by** *fastforce*

lemma *ctwo-ordLess-Cinfinite*:
  assumes *r*: *Cinfinite r*
  shows *ctwo* <*o r*
proof −
  have *natLeq* $\leq$*o r* **using** *r* **by** (*rule natLeq-ordLeq-cinfinite*)
  thus *?thesis* **using** *ctwo-ordLess-natLeq ordLess-ordLeq-trans* **by** *blast*
qed

lemma *ctwo-ordLeq-Cinfinite*:

assumes *r*: *Cinfinite r*
shows *ctwo* $\leq$*o r*
by (*rule ordLess-imp-ordLeq*[*OF ctwo-ordLess-Cinfinite*[*OF r*]])

lemma *Cinfinite-ordLess-cexp*:
assumes *r*: *Cinfinite r*
shows *r* <*o r* ^*c r*
proof −
have *r* <*o ctwo* ^*c r* **using** *r* **by** (*simp only*: *ordLess-ctwo-cexp*)
also have *ctwo* ^*c r* $\leq$*o r* ^*c r*
by (*rule cexp-mono1*[*OF ctwo-ordLeq-Cinfinite*]) (*auto simp*: *r ctwo-not-czero*)
finally show *?thesis* **.**
qed

lemma *infinite-ordLeq-cexp*:
assumes *r*: *Cinfinite r*
shows *r* $\leq$*o r* ^*c r*
by (*rule ordLess-imp-ordLeq*[*OF Cinfinite-ordLess-cexp*[*OF r*]])

lemma *cone-ordLeq-iff-Field*:
assumes *cone* $\leq$*o r*
shows *Field r* $\neq$ {}
proof (*rule ccontr*)
assume ¬ *Field r* $\neq$ {}
hence *Field r* = {} **by** *simp*
thus *False* **using** *card-of-empty3*
card-of-mono2[*OF assms*] *Cnotzero-imp-not-empty*[*OF cone-Cnotzero*] **by** *auto*
qed

lemma *cone-ordLeq-cexp*[*simp*]:
assumes *cone* $\leq$*o r1*
shows *cone* $\leq$*o r1* ^*c r2*
proof −
have *Field r1* $\neq$ {} **using** *assms cone-ordLeq-iff-Field* **by** *auto*
thus *?thesis* **unfolding** *cone-def cexp-def* **by** *auto*
qed

lemma *Card-order-czero*[*simp*]: *Card-order czero*
by (*simp only*: *card-of-Card-order czero-def*)

lemma *cexp-mono2''*:
assumes *2*: *p2* $\leq$*o r2*
and *n1*: *Cnotzero q*
and *n2*: *Card-order p2*
shows *q* ^*c p2* $\leq$*o q* ^*c r2*
proof (*cases p2* =*o* (*czero* :: *'a rel*))
case *True*
hence *q* ^*c p2* =*o q* ^*c* (*czero* :: *'a rel*) **using** *n1 n2 cexp-cong2 Card-order-czero*
by *blast*

also have q ^c ($czero$ :: $'a$ rel) $=o$ $cone$ **using** $cexp$-$czero$ **by** *blast*
also have $cone \leq o$ q ^c $r2$ **using** *cone-ordLeq-cexp cone-ordLeq-Cnotzero n1* **by** *blast*
finally show *?thesis* **.**
next
case *False* **thus** *?thesis* **using** *assms cexp-mono2′ czeroI* **by** *metis*
qed

9.9 Infinite bounds

lemma *cone-ordLeq-cinfinite*: *Cinfinite r* $\Longrightarrow$ *cone* $\leq o$ *r*
unfolding *cinfinite-def cone-def*
by (*auto intro*: *Card-order-singl-ordLeq*)

lemma *Un-Cinfinite-bound*:
$[\![|A| \leq o\ r;\ |B| \leq o\ r;\ Cinfinite\ r]\!] \Longrightarrow |A \cup B| \leq o\ r$
by (*auto simp add*: *cinfinite-def*)

lemma *UNION-Cinfinite-bound*:
$[\![|I| \leq o\ r;\ \forall\, i \in I.\ |A\ i| \leq o\ r;\ Cinfinite\ r]\!] \Longrightarrow |\bigcup i \in I.\ A\ i| \leq o\ r$
by (*auto simp add*: *card-of-UNION-ordLeq-infinite-Field cinfinite-def*)

lemma *csum-cinfinite-bound*:
assumes $p \leq o\ r\ q \leq o\ r$ *Card-order p Card-order q Cinfinite r*
shows $p +c\ q \leq o\ r$
proof −
from *assms*(*1*−*4*) **have** $|Field\ p| \leq o\ r\ |Field\ q| \leq o\ r$
unfolding *card-order-on-def*
using *card-of-least ordLeq-transitive* **by** *blast+*
with *assms* **show** *?thesis* **unfolding** *cinfinite-def csum-def*
by (*blast intro*: *card-of-Plus-ordLeq-infinite-Field*)
qed

lemma *csum-cexp*: $[\![Cinfinite\ r1;\ Cinfinite\ r2;\ Card\text{-}order\ q;\ ctwo \leq o\ q]\!] \Longrightarrow$
q ^c $r1 +c\ q$ ^c $r2 \leq o\ q$ ^c $(r1 +c\ r2)$
apply (*rule csum-cinfinite-bound*)
apply (*rule cexp-mono2*)
apply (*rule ordLeq-csum1*)
apply (*erule conjunct2*)
apply *assumption*
apply (*rule disjI2*)
apply (*rule ordLeq-transitive*)
apply (*rule cone-ordLeq-ctwo*)
apply (*rule ordLeq-transitive*)
apply *assumption*
apply (*rule ordLeq-cexp1*)
apply (*rule Cinfinite-Cnotzero*)
apply (*rule Cinfinite-csum*)
apply (*rule disjI1*)

apply *assumption*
apply *assumption*
apply (*rule notE*)
apply (*rule cinfinite-not-czero*[*of r1*])
apply (*erule conjunct1*)
apply *assumption*
apply (*erule conjunct2*)
apply (*rule cexp-mono2*)
apply (*rule ordLeq-csum2*)
apply (*erule conjunct2*)
apply *assumption*
apply (*rule disjI2*)
apply (*rule ordLeq-transitive*)
apply (*rule cone-ordLeq-ctwo*)
apply (*rule ordLeq-transitive*)
apply *assumption*
apply (*rule ordLeq-cexp1*)
apply (*rule Cinfinite-Cnotzero*)
apply (*rule Cinfinite-csum*)
apply (*rule disjI1*)
apply *assumption*
apply *assumption*
apply (*rule notE*)
apply (*rule cinfinite-not-czero*[*of r2*])
apply (*erule conjunct1*)
apply *assumption*
apply (*erule conjunct2*)
apply (*rule Card-order-cexp*)
apply (*rule Card-order-cexp*)
apply (*rule Cinfinite-cexp*)
apply *assumption*
apply (*rule Cinfinite-csum*)
apply (*rule disjI1*)
apply *assumption*
done

lemma *csum-cexp'*: $[\![$*Cinfinite r*; *Card-order q*; *ctwo* $\leq o$ *q*$]\!] \Longrightarrow q +c\ r \leq o\ q\ \hat{}c\ r$
by (*rule csum-cinfinite-bound*) (*auto simp add*: *ordLeq-cexp1 ordLeq-cexp2 Cinfinite-cexp*)

lemma *cprod-cinfinite-bound*:
  assumes $p \leq o\ r\ q \leq o\ r$ *Card-order p Card-order q Cinfinite r*
  shows $p *c\ q \leq o\ r$
proof −
  from *assms*(*1*−*4*) **have** $|Field\ p| \leq o\ r\ |Field\ q| \leq o\ r$
    unfolding *card-order-on-def*
    using *card-of-least ordLeq-transitive* **by** *blast*+
  with *assms* **show** *?thesis* **unfolding** *cinfinite-def cprod-def*
    by (*blast intro*: *card-of-Times-ordLeq-infinite-Field*)
qed

lemma *cprod-csum-cexp*:
r1 $*c$ *r2* $\leq o$ (*r1* $+c$ *r2*) $\hat{}c$ *ctwo*
unfolding *cprod-def csum-def cexp-def ctwo-def Field-card-of*
proof −
let *?f* = $\lambda(a, b)$. %*x*. *if x then Some* (*Inl a*) *else Some* (*Inr b*)
have *inj-on ?f* (*Field r1* × *Field r2*) (**is** *inj-on - ?LHS*)
unfolding *inj-on-def fun-eq-iff* **by** (*auto split*: *bool.split*)
moreover
have *?f* ‘ *?LHS* ⊆ *Func* (*UNIV* :: *bool set*) (*Field r1* <+> *Field r2*) (**is** - ⊆ *?RHS*)
unfolding *Func-def* **by** *auto*
ultimately show |*?LHS*| $\leq o$ |*?RHS*| **using** *card-of-ordLeq* **by** *blast*
qed

lemma *Cinfinite-cardSuc*: *Cinfinite r* ⟹ *Cinfinite* (*cardSuc r*)
by (*rule Cinfinite-mono*) *auto*

lemma *Cnotzero-cardSuc*: *Card-order r* ⟹ *Cnotzero* (*cardSuc r*)
using *Field-cardSuc-not-empty czeroE* **by** *auto*

9.10 Powerset

definition *cpow* **where** *cpow r* = |*Pow* (*Field r*)|

lemma *card-order-cpow*[*simp*]:
assumes *card-order r*
shows *card-order* (*cpow r*)
proof −
have *Field r* = *UNIV* **using** *assms card-order-on-Card-order* **by** *auto*
thus *?thesis* **unfolding** *cpow-def* **by** *auto*
qed

lemma *cpow-greater*[*simp*]: *Card-order r* ⟹ *r* <*o cpow r*
unfolding *cpow-def* **by** *auto*

lemma *cpow-greater-eq*[*simp*]: *Card-order r* ⟹ *r* $\leq o$ *cpow r*
by (*rule ordLess-imp-ordLeq*) (*erule cpow-greater*)

lemma *Card-order-cpow*[*simp*]: *Card-order* (*cpow r*)
unfolding *cpow-def* **by** *simp*

lemma *Cinfinite-cpow*: *Cinfinite r* ⟹ *Cinfinite* (*cpow r*)
by (*rule Cinfinite-mono*) *auto*

lemma *Cnotzero-cpow*: *Card-order r* ⟹ *Cnotzero* (*cpow r*)

unfolding *cpow-def* **by** (*auto* , *drule czeroE*, *auto*)

lemma *cardSuc-ordLeq-cpow*: *Card-order r* $\Longrightarrow$ *cardSuc r* $\leq$*o cpow r*
by (*rule cardSuc-least*) *auto*

lemma *cpow-cexp-ctwo*:
cpow r =o ctwo ^c r
unfolding *cpow-def ctwo-def cexp-def Field-card-of* **by** (*rule card-of-Pow-Func*)

end