

Ordinals and cardinals in HOL

Andrei Popescu & Dmitriy Traytel

Contents

1	Introduction	3
2	More on Injections, Bijections and Inverses	5
2.1	Purely functional properties	5
2.2	Properties involving finite and infinite sets	6
2.3	Properties involving Hilbert choice	6
2.4	Other facts	6
3	Basics on Order-Like Relations	7
3.1	The upper and lower bounds operators	7
3.2	Properties depending on more than one relation	19
4	More on Well-Founded Relations	19
4.1	Well-founded recursion via genuine fixpoints	19
5	Well-Order Relations	20
5.1	Auxiliaries	20
5.2	Well-founded induction and recursion adapted to non-strict well-order relations	20
5.2.1	Properties of max2	21
5.2.2	Properties of minim	21
5.2.3	Properties of sup	22
5.2.4	Properties of successor	23
5.2.5	Properties of order filters	25
5.2.6	Other properties	26
6	Well-Order Embeddings	27
6.1	Auxiliaries	27
6.2	(Well-order) embeddings, strict embeddings, isomorphisms and order-compatible functions	27
6.3	Uniqueness of embeddings	28
7	Order Union	29

8	Constructions on Wellorders	33
8.1	Order filters versus restrictions and embeddings	33
8.2	Ordering the well-orders by existence of embeddings	34
8.3	Copy via direct images	35
8.4	The maxim among a finite set of ordinals	37
8.5	Limit and succesor ordinals	39
8.5.1	Successor and limit elements of an ordinal	41
8.5.2	Well-order recursion with (zero), succesor, and limit	44
8.5.3	Well-order succ-lim induction	45
8.6	Projections of wellorders	46
9	Ordinal Arithmetic	49
10	Cardinal-Order Relations	87
10.1	Cardinal of a set	88
10.2	Cardinals versus set operations on arbitrary sets	89
10.3	Cardinals versus set operations involving infinite sets	98
10.4	Cardinals versus lists	98
10.5	Cardinals versus the finite powerset operator	101
10.6	The cardinal ω and the finite cardinals	103
10.6.1	First as well-orders	103
10.6.2	Then as cardinals	105
10.6.3	"Backward compatibility" with the numeric cardinal operator for finite sets	107
10.7	The successor of a cardinal	108
10.8	Others	110
10.9	Infinite cardinals are limit ordinals	114
10.10	Regular vs. stable cardinals	116
11	Cardinal Arithmetic	118
11.1	Binary sum	118
11.2	Product	119
11.3	Exponentiation	119
11.4	Powerset	122
11.5	Inverse image	123
11.6	Maximum	123
12	Extending Well-founded Relations to Wellorders	125
12.1	Extending Well-founded Relations to Wellorders	125
13	Theory of Ordinals and Cardinals	129

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-order 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:

¹The “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 sets³ 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.

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

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

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

⁵This 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 Fun-More
  imports Main
begin
```

2.1 Purely functional properties

```
lemma bij-betw-diff-singl:
  assumes BIJ: bij-betw f A A' and IN: a ∈ 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 *blast*
 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 *simp*
qed

2.2 Properties involving finite and infinite sets

lemma *bij-betw-inv-into-RIGHT*:
 assumes *BIJ*: *bij-betw* f A A' and *SUB*: $B' \leq A'$
 shows $f\ ' (inv-into\ A\ f)\ ' B' = B'$
 by (*metis BIJ SUB bij-betw-imp-surj-on image-inv-into-cancel*)

lemma *bij-betw-inv-into-RIGHT-LEFT*:
 assumes *BIJ*: *bij-betw* f A A' and *SUB*: $B' \leq A'$ and
 IM : $(inv-into\ A\ f)\ ' B' = B$
 shows $f\ ' B = B'$
 by (*metis BIJ IM SUB bij-betw-inv-into-RIGHT*)

lemma *bij-betw-inv-into-twice*:
 assumes *bij-betw* f A A'
 shows $\forall a \in A. inv-into\ A'\ (inv-into\ A\ f)\ a = f\ a$
 by (*simp add: assms inv-into-inv-into-eq*)

2.3 Properties involving Hilbert choice

lemma *bij-betw-inv-into-LEFT*:
 assumes *BIJ*: *bij-betw* f A A' and *SUB*: $B \leq A$
 shows $(inv-into\ A\ f)\ '(f\ ' B) = B$
 using *assms unfolding* *bij-betw-def* using *inv-into-image-cancel* by *force*

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 $(inv-into\ A\ f)\ ' B' = B$
 using *assms bij-betw-inv-into-LEFT*[*of f A A' B*] by *fast*

2.4 Other facts

lemma *atLeastLessThan-injective*:
 assumes $\{0 \ ..< m::nat\} = \{0 \ ..< n\}$
 shows $m = n$
 using *assms atLeast0LessThan* by *force*

```

lemma atLeastLessThan-injective2:
  bij-betw  $f \{0 \dots m::nat\} \{0 \dots n\} \implies m = n$ 
  using bij-betw-same-card by fastforce

lemma atLeastLessThan-less-eq:
   $(\{0 \dots m\} \leq \{0 \dots n\}) = ((m::nat) \leq n)$ 
  by auto

lemma atLeastLessThan-less-eq2:
  assumes inj-on  $f \{0 \dots (m::nat)\} f ' \{0 \dots m\} \leq \{0 \dots n\}$ 
  shows  $m \leq n$ 
  by (metis assms card-inj-on-le card-lessThan finite-lessThan lessThan-atLeast0)

lemma atLeastLessThan-less:
   $(\{0 \dots m\} < \{0 \dots n\}) = ((m::nat) < n)$ 
  by auto

end

```

3 Basics on Order-Like Relations

```

theory Order-Relation-More
  imports Main
begin

```

3.1 The upper and lower bounds operators

```

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

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

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

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

lemma Refl-above-in:  $\llbracket Refl \ r; a \in Field \ r \rrbracket \implies a \in above \ r \ a$ 
  by(auto simp add: refl-on-def above-def)

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

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

```

lemma *in-UnderS-aboveS*: $a \in \text{Field } r \implies a \in \text{UnderS } r \ (\text{aboveS } r \ a)$
by(*auto simp add: UnderS-def aboveS-def*)

lemma *UnderS-subset-Under*: $\text{UnderS } r \ A \leq \text{Under } r \ A$
by(*auto simp add: UnderS-def Under-def*)

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

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

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

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

lemma *Under-Above-Galois*:
 $\llbracket B \leq \text{Field } r; C \leq \text{Field } r \rrbracket \implies (B \leq \text{Above } r \ C) = (C \leq \text{Under } r \ B)$
by(*unfold Above-def Under-def, blast*)

lemma *UnderS-AboveS-Galois*:
 $\llbracket B \leq \text{Field } r; C \leq \text{Field } r \rrbracket \implies (B \leq \text{AboveS } r \ C) = (C \leq \text{UnderS } r \ B)$
by(*unfold AboveS-def UnderS-def, blast*)

lemma *Refl-above-aboveS*:
assumes *REFL*: *Refl* r **and** *IN*: $a \in \text{Field } r$
shows $\text{above } r \ a = \text{aboveS } r \ a \cup \{a\}$
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*: $a \in \text{Field } r$
shows $\text{Field } r = \text{under } r \ a \cup \text{aboveS } r \ a$
proof(*unfold under-def aboveS-def, auto*)
assume $a \in \text{Field } r \ (a, a) \notin r$
with *LIN IN order-on-defs*[*of - r*] *refl-on-def*[*of - r*] **show** *False* **by** *auto*
next
fix b **assume** $b \in \text{Field } r \ (b, a) \notin r$
with *LIN IN order-on-defs*[*of Field r r*] *total-on-def*[*of Field r r*]
have $(a, b) \in r \vee a = b$ **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 $b \in \text{Field } r$

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

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

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

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

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

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

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

lemma *Refl-under-Under*:

```

    assumes REFL: Refl r and NE:  $A \neq \{\}$ 
    shows  $\text{Under } r \ A = (\bigcap a \in A. \text{under } r \ a)$ 
  proof
    show  $\text{Under } r \ A \subseteq (\bigcap a \in A. \text{under } r \ a)$ 
      by(unfold Under-def under-def, auto)
  next
    show  $(\bigcap a \in A. \text{under } r \ a) \subseteq \text{Under } r \ A$ 
    proof(auto)
      fix x
      assume *:  $\forall xa \in A. x \in \text{under } r \ xa$ 
      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  $x \in \text{under } r \ a$  by simp
       hence  $x \in \text{Field } r$ 
        using under-Field[of r a] by auto
      }
      ultimately show  $x \in \text{Under } r \ A$ 
        unfolding Under-def by auto
    qed
  qed

```

```

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

```

```

lemma Refl-above-Above:
  assumes REFL: Refl r and NE:  $A \neq \{\}$ 

```

```

shows Above r A = ( $\bigcap a \in A. \text{above } r a$ )
proof
  show Above r A  $\subseteq$  ( $\bigcap a \in A. \text{above } r a$ )
    by(unfold Above-def above-def, auto)
next
  show ( $\bigcap a \in A. \text{above } r a$ )  $\subseteq$  Above r A
  proof(auto)
    fix x
    assume *:  $\forall xa \in A. x \in \text{above } r xa$ 
    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  $x \in \text{above } r a$  by simp
     hence  $x \in \text{Field } r$ 
       using above-Field[of r a] by auto
    }
    ultimately show  $x \in \text{Above } r A$ 
      unfolding Above-def by auto
  qed
qed

```

```

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

```

```

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

```

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

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

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

lemma *Under-decr*: $A \leq B \implies \text{Under } r \ B \leq \text{Under } r \ A$
by(*unfold Under-def, auto*)

lemma *UnderS-decr*: $A \leq B \implies \text{UnderS } r \ B \leq \text{UnderS } r \ A$
by(*unfold UnderS-def, auto*)

lemma *Above-decr*: $A \leq B \implies \text{Above } r \ B \leq \text{Above } r \ A$
by(*unfold Above-def, auto*)

lemma *AboveS-decr*: $A \leq B \implies \text{AboveS } r \ B \leq \text{AboveS } r \ A$
by(*unfold AboveS-def, auto*)

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

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

lemma *aboveS-decr*:
assumes *TRANS*: *trans* *r* **and** *ANTISYM*: *antisym* *r* **and**
REL: $(a,b) \in r$

```

  shows  $\text{aboveS } r \ b \leq \text{aboveS } r \ a$ 
proof(unfold aboveS-def, auto)
  assume *:  $a \neq b$  and **:  $(b, a) \in r$ 
  with ANTISYM antisym-def[of r] REL
  show False by auto
next
  fix x assume  $x \neq b$   $(b, x) \in r$ 
  with REL TRANS trans-def[of r]
  show  $(a, x) \in r$  by blast
qed

lemma under-trans:
  assumes TRANS:  $\text{trans } r$  and
    IN1:  $a \in \text{under } r \ b$  and IN2:  $b \in \text{under } r \ c$ 
  shows  $a \in \text{under } r \ c$ 
proof-
  have  $(a, b) \in r \wedge (b, c) \in r$ 
  using IN1 IN2 under-def by fastforce
  hence  $(a, c) \in r$ 
  using TRANS trans-def[of r] by blast
  thus ?thesis unfolding under-def by simp
qed

lemma under-underS-trans:
  assumes TRANS:  $\text{trans } r$  and ANTISYM:  $\text{antisym } r$  and
    IN1:  $a \in \text{under } r \ b$  and IN2:  $b \in \text{underS } r \ c$ 
  shows  $a \in \text{underS } r \ c$ 
proof-
  have 0:  $(a, b) \in r \wedge (b, c) \in r$ 
  using IN1 IN2 under-def underS-def by fastforce
  hence 1:  $(a, c) \in r$ 
  using TRANS trans-def[of r] by blast
  have 2:  $b \neq c$  using IN2 underS-def by force
  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-under-trans:
  assumes TRANS:  $\text{trans } r$  and ANTISYM:  $\text{antisym } r$  and
    IN1:  $a \in \text{underS } r \ b$  and IN2:  $b \in \text{under } r \ c$ 
  shows  $a \in \text{underS } r \ c$ 
proof-
  have 0:  $(a, b) \in r \wedge (b, c) \in r$ 
  using IN1 IN2 under-def underS-def by fast
  hence 1:  $(a, c) \in r$ 

```

```

    using TRANS trans-def[of r] by fast
    have 2:  $a \neq b$  using IN1 underS-def by force
    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:  $a \in \text{underS } r \ b$  and IN2:  $b \in \text{underS } r \ c$ 
  shows  $a \in \text{underS } r \ c$ 
  proof-
    have  $a \in \text{under } r \ b$ 
      using IN1 underS-subset-under by fast
    with assms under-underS-trans show ?thesis by fast
  qed

lemma above-trans:
  assumes TRANS: trans r and
    IN1:  $b \in \text{above } r \ a$  and IN2:  $c \in \text{above } r \ b$ 
  shows  $c \in \text{above } r \ a$ 
  proof-
    have  $(a,b) \in r \wedge (b,c) \in r$ 
      using IN1 IN2 above-def by fast
    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:  $b \in \text{above } r \ a$  and IN2:  $c \in \text{aboveS } r \ b$ 
  shows  $c \in \text{aboveS } r \ a$ 
  proof-
    have 0:  $(a,b) \in r \wedge (b,c) \in r$ 
      using IN1 IN2 above-def aboveS-def by fast
    hence 1:  $(a,c) \in r$ 
      using TRANS trans-def[of r] by blast
    have 2:  $b \neq c$  using IN2 aboveS-def by force
    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: $b \in \text{aboveS } r \ a$ **and** *IN2*: $c \in \text{above } r \ b$
shows $c \in \text{aboveS } r \ a$
proof–
have 0 : $(a,b) \in r \wedge (b,c) \in r$
using *IN1 IN2 above-def aboveS-def* **by** *fast*
hence 1 : $(a,c) \in r$
using *TRANS trans-def[of r]* **by** *blast*
have 2 : $a \neq b$ **using** *IN1 aboveS-def* **by** *force*
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: $b \in \text{aboveS } r \ a$ **and** *IN2*: $c \in \text{aboveS } r \ b$
shows $c \in \text{aboveS } r \ a$
proof–
have $b \in \text{above } r \ a$
using *IN1 aboveS-subset-above* **by** *fast*
with *assms above-aboveS-trans* **show** *?thesis* **by** *fast*
qed

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

lemma *underS-Under-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
IN1: $a \in \text{underS } r \ b$ **and** *IN2*: $b \in \text{Under } r \ C$

shows $a \in \text{UnderS } r \ C$
proof–
 from $IN1$ have $a \in \text{under } r \ b$
 using $\text{underS-subset-under}[of \ r \ b]$ by *fast*
 with *assms under-Under-trans*
 have $a \in \text{Under } r \ C$ by *fast*

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

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

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

lemma above-Above-trans :
 assumes $\text{TRANS: trans } r$ and
 $IN1: a \in \text{above } r \ b$ and $IN2: b \in \text{Above } r \ C$
 shows $a \in \text{Above } r \ C$
proof–
 have $(b, a) \in r \wedge (\forall c \in C. (c, b) \in r)$
 using $IN1[\text{unfolded above-def}] \ IN2[\text{unfolded Above-def}]$ by *simp*
 hence $\forall c \in C. (c, a) \in r$
 using $\text{TRANS trans-def}[of \ r]$ by *blast*
 moreover
 have $a \in \text{Field } r$ using $IN1[\text{unfolded above-def}] \ \text{Field-def}$ by *fast*
 ultimately
 show *?thesis* unfolding Above-def by *auto*

qed

lemma *aboveS-Above-trans*:

assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**

IN1: $a \in \text{aboveS } r \ b$ **and** *IN2*: $b \in \text{Above } r \ C$

shows $a \in \text{AboveS } r \ C$

proof–

from *IN1* **have** $a \in \text{above } r \ b$

using *aboveS-subset-above*[*of r b*] **by** *blast*

with *assms above-Above-trans*

have $a \in \text{Above } r \ C$ **by** *fast*

moreover

have $a \notin C$

proof

assume *: $a \in C$

have 1: $b \neq a \wedge (b, a) \in r$

using *IN1 aboveS-def*[*of r b*] **by** *auto*

have $\forall c \in C. (c, b) \in r$

using *IN2 Above-def*[*of r C*] **by** *auto*

with * **have** $(a, b) \in r$ **by** *simp*

with 1 *ANTISYM antisym-def*[*of r*]

show *False* **by** *blast*

qed

ultimately

show *?thesis unfolding AboveS-def*

using *Above-def* **by** *force*

qed

lemma *above-AboveS-trans*:

assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**

IN1: $a \in \text{above } r \ b$ **and** *IN2*: $b \in \text{AboveS } r \ C$

shows $a \in \text{AboveS } r \ C$

proof–

from *IN2* **have** $b \in \text{Above } r \ C$

using *AboveS-subset-Above*[*of r C*] **by** *blast*

with *assms above-Above-trans*

have $a \in \text{Above } r \ C$ **by** *force*

moreover

have $a \notin C$

proof

assume *: $a \in C$

have 1: $(b, a) \in r$

using *IN1 above-def*[*of r b*] **by** *auto*

have $\forall c \in C. b \neq c \wedge (c, b) \in r$

using *IN2 AboveS-def*[*of r 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 force
qed

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

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

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

  ultimately
  show ?thesis unfolding UnderS-def Under-def by fast
qed

```

3.2 Properties depending on more than one relation

lemma *under-incr2*:

$r \leq r' \implies \text{under } r \ a \leq \text{under } r' \ a$

unfolding *under-def* **by** *blast*

lemma *underS-incr2*:

$r \leq r' \implies \text{underS } r \ a \leq \text{underS } r' \ a$

unfolding *underS-def* **by** *blast*

lemma *above-incr2*:

$r \leq r' \implies \text{above } r \ a \leq \text{above } r' \ a$

unfolding *above-def* **by** *blast*

lemma *aboveS-incr2*:

$r \leq r' \implies \text{aboveS } r \ a \leq \text{aboveS } r' \ a$

unfolding *aboveS-def* **by** *blast*

end

4 More on Well-Founded Relations

theory *Wellfounded-More*

imports *Main Order-Relation-More*

begin

4.1 Well-founded recursion via genuine fixpoints

lemma *adm-wf-unique-fixpoint*:

fixes $r :: ('a * 'a) \text{ set}$ **and**

$H :: ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$ **and**

$f :: 'a \Rightarrow 'b$ **and** $g :: 'a \Rightarrow 'b$

assumes *WF*: $\text{wf } r$ **and** *ADM*: $\text{adm-wf } r \ H$ **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 (λx. 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 $r :: ('a * 'a) \text{ set}$ **and**
 $H :: ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$ **and**
 $f :: 'a \Rightarrow 'b$
assumes $WF: wf\ r$ **and** $ADM: adm\text{-}wf\ r\ H$ **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
end

5 Well-Order Relations

theory *Wellorder-Relation*
imports *Wellfounded-More*
begin

context *wo-rel*
begin

5.1 Auxiliaries

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 *cases-Total2*:
 $\bigwedge\ phi\ a\ b. \llbracket \{a,b\} \leq Field\ r; ((a,b) \in r - Id \implies phi\ a\ b);$
 $((b,a) \in r - Id \implies phi\ a\ b); (a = b \implies phi\ a\ b) \rrbracket$
 $\implies phi\ a\ b$
using *TOTALS* **by** *auto*

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

lemma *worec-unique-fixpoint*:
assumes $ADM: adm\text{-}wo\ H$ **and** $fp: f = H\ f$
shows $f = worec\ H$
proof–
have $adm\text{-}wf\ (r - Id)\ H$

unfolding *adm-wf-def*
using *ADM adm-wo-def[of H] underS-def[of r]* **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

5.2.1 Properties of max2

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

5.2.2 Properties of minim

lemma *minim-Under*:
 $\llbracket B \leq Field\ r; B \neq \{\} \rrbracket \implies minim\ B \in Under\ B$
by(*auto simp add: Under-def minim-inField minim-least*)

lemma *equals-minim-Under*:
 $\llbracket B \leq Field\ r; a \in B; a \in Under\ B \rrbracket$
 $\implies a = minim\ B$
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 $a \in B \wedge a \in Under\ B$
using *assms minim-in minim-Under* **by** *simp*
next
assume $a \in B \wedge a \in Under\ B$
thus $a = minim\ B$
using *assms equals-minim-Under* **by** *simp*
qed

lemma *minim-Under-under*:
assumes *NE*: $A \neq \{\}$ **and** *SUB*: $A \leq Field\ r$
shows $Under\ A = under\ (minim\ A)$
proof–

have $\text{minim } A \in A$
using *assms minim-in* **by** *auto*
then have $\text{Under } A \leq \text{under } (\text{minim } A)$
by (*simp add: Under-decr under-Under-singl*)
moreover have $\text{under } (\text{minim } A) \leq \text{Under } A$
by (*meson NE SUB TRANS minim-Under subset-eq under-Under-trans*)
ultimately show *?thesis* **by** *blast*
qed

lemma *minim-UnderS-underS*:
assumes *NE: $A \neq \{\}$ and SUB: $A \leq \text{Field } r$*
shows $\text{UnderS } A = \text{underS } (\text{minim } A)$
proof –
have $\text{minim } A \in A$
using *assms minim-in* **by** *auto*
then have $\text{UnderS } A \leq \text{underS } (\text{minim } A)$
by (*simp add: UnderS-decr underS-UnderS-singl*)
moreover have $\text{underS } (\text{minim } A) \leq \text{UnderS } A$
by (*meson ANTISYM NE SUB TRANS minim-Under subset-eq underS-Under-trans*)
ultimately show *?thesis* **by** *blast*
qed

5.2.3 Properties of supr

lemma *supr-Above*:
assumes *Above $B \neq \{\}$*
shows $\text{supr } B \in \text{Above } B$
by (*simp add: assms Above-Field minim-in supr-def*)

lemma *supr-greater*:
assumes *Above $B \neq \{\}$ $b \in B$*
shows $(b, \text{supr } B) \in r$
using *assms Above-def supr-Above* **by** *fastforce*

lemma *supr-least-Above*:
assumes *$a \in \text{Above } B$*
shows $(\text{supr } B, a) \in r$
by (*simp add: assms Above-Field minim-least supr-def*)

lemma *supr-least*:
 $\llbracket B \leq \text{Field } r; a \in \text{Field } r; (\bigwedge b. b \in B \implies (b, a) \in r) \rrbracket$
 $\implies (\text{supr } B, a) \in r$
by(*auto simp add: supr-least-Above Above-def*)

lemma *equals-supr-Above*:
assumes *$a \in \text{Above } B \wedge a'. a' \in \text{Above } B \implies (a, a') \in r$*
shows $a = \text{supr } B$
by (*simp add: assms Above-Field equals-minim supr-def*)

lemma *equals-supr*:
 assumes *SUB*: $B \leq \text{Field } r$ and *IN*: $a \in \text{Field } r$ and
 ABV: $\bigwedge b. b \in B \implies (b, a) \in r$ and
 MINIM: $\bigwedge a'. \llbracket a' \in \text{Field } r; \bigwedge b. b \in B \implies (b, a') \in r \rrbracket \implies (a, a') \in r$
 shows $a = \text{supr } B$
proof–
 have $a \in \text{Above } B$
 unfolding *Above-def* using *ABV IN* by *simp*
 moreover
 have $\bigwedge a'. a' \in \text{Above } B \implies (a, a') \in r$
 unfolding *Above-def* using *MINIM* by *simp*
 ultimately show *?thesis*
 using *equals-supr-Above SUB* by *auto*
qed

lemma *supr-inField*:
 assumes *Above* $B \neq \{\}$
 shows $\text{supr } B \in \text{Field } r$
 by (*simp add: Above-Field assms minim-inField supr-def*)

lemma *supr-above-Above*:
 assumes *SUB*: $B \leq \text{Field } r$ and *ABOVE*: *Above* $B \neq \{\}$
 shows *Above* $B = \text{above } (\text{supr } B)$
 apply (*clarsimp simp: Above-def above-def set-eq-iff*)
 by (*meson ABOVE FieldI2 SUB TRANS supr-greater supr-least transD*)

lemma *supr-under*:
 assumes $a \in \text{Field } r$
 shows $a = \text{supr } (\text{under } a)$
proof–
 have $\text{under } a \leq \text{Field } r$
 using *under-Field[of r]* by *auto*
 moreover
 have $\text{under } a \neq \{\}$
 using *assms Refl-under-in[of r] REFL* by *auto*
 moreover
 have $a \in \text{Above } (\text{under } a)$
 using *in-Above-under[of - r] assms* by *auto*
 moreover
 have $\forall a' \in \text{Above } (\text{under } a). (a, a') \in r$
 by (*auto simp: Above-def above-def REFL Refl-under-in assms*)
 ultimately show *?thesis*
 using *equals-supr-Above* by *auto*
qed

5.2.4 Properties of successor

lemma *suc-least*:
 $\llbracket B \leq \text{Field } r; a \in \text{Field } r; (\bigwedge b. b \in B \implies a \neq b \wedge (b, a) \in r) \rrbracket$

$\implies (suc\ B, a) \in r$
by(*auto simp add: suc-least-AboveS AboveS-def*)

lemma *equals-suc*:

assumes *SUB*: $B \leq Field\ r$ **and** *IN*: $a \in Field\ r$ **and**

ABVS: $\bigwedge b. b \in B \implies a \neq b \wedge (b, a) \in r$ **and**

MINIM: $\bigwedge a'. \llbracket a' \in Field\ r; \bigwedge b. b \in B \implies a' \neq b \wedge (b, a') \in r \rrbracket \implies (a, a') \in$

r

shows $a = suc\ B$

proof–

have $a \in AboveS\ B$

unfolding *AboveS-def* **using** *ABVS IN* **by** *simp*

moreover

have $\bigwedge a'. a' \in AboveS\ B \implies (a, a') \in r$

unfolding *AboveS-def* **using** *MINIM* **by** *simp*

ultimately show *?thesis*

using *equals-suc-AboveS SUB* **by** *auto*

qed

lemma *suc-above-AboveS*:

assumes *SUB*: $B \leq Field\ r$ **and**

ABOVE: $AboveS\ B \neq \{\}$

shows $AboveS\ B = above\ (suc\ B)$

using *assms*

proof(*unfold AboveS-def above-def, auto*)

fix *a* **assume** $a \in Field\ r \ \forall b \in B. a \neq b \wedge (b, a) \in r$

with *suc-least assms*

show $(suc\ B, a) \in r$ **by** *auto*

next

fix *b* **assume** $(suc\ B, b) \in r$

thus $b \in Field\ r$

using *REFL refl-on-def[of - r]* **by** *auto*

next

fix *a b*

assume *1*: $(suc\ B, b) \in r$ **and** *2*: $a \in B$

with *assms suc-greater[of B a]*

have $(a, suc\ B) \in r$ **by** *auto*

thus $(a, b) \in r$

using *1 TRANS trans-def[of r]* **by** *blast*

next

fix *a*

assume $(suc\ B, a) \in r$ **and** *2*: $a \in B$

thus *False*

by (*metis ABOVE ANTISYM SUB antisymD suc-greater*)

qed

lemma *suc-singl-pred*:

assumes *IN*: $a \in Field\ r$ **and** *ABOVE-NE*: $aboveS\ a \neq \{\}$ **and**

REL: $(a', suc\ \{a\}) \in r$ **and** *DIFF*: $a' \neq suc\ \{a\}$

shows $a' = a \vee (a', a) \in r$
proof –
 have *: $\text{succ } \{a\} \in \text{Field } r \wedge a' \in \text{Field } r$
 using *WELL REL well-order-on-domain* **by** *metis*
 {assume **: $a' \neq a$
 hence $(a, a') \in r \vee (a', a) \in r$
 using *TOTAL IN ** **by** (*auto simp add: total-on-def*)
 moreover
 {assume $(a, a') \in r$
 with ** * *assms WELL succ-least*[*of* $\{a\}$ a']
 have $(\text{succ } \{a\}, a') \in r$ **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*: $a \in \text{Field } r$ **and** *ABV*: $\text{aboveS } a \neq \{\}$
 shows $\text{underS } (\text{succ } \{a\}) = \text{under } a$
proof –
 have $\text{AboveS } \{a\} \neq \{\}$
 using *ABV aboveS-AboveS-singl*[*of* r] **by** *auto*
 then have 2: $a \neq \text{succ } \{a\} \wedge (a, \text{succ } \{a\}) \in r$
 using *suc-greater*[*of* $\{a\}$ a] *IN* **by** *auto*
 have $\text{underS } (\text{succ } \{a\}) \leq \text{under } a$
 using *ABV IN REFL Reft-under-in underS-E under-def wo-rel.suc-singl-pred*
wo-rel-axioms **by** *fastforce*
 moreover
 have $\text{under } a \leq \text{underS } (\text{succ } \{a\})$
 by (*simp add: 2 ANTISYM TRANS subsetI underS-I under-underS-trans*)
 ultimately show ?thesis **by** *blast*
qed

5.2.5 Properties of order filters

lemma *ofilter-Under*[*simp*]:
 assumes $A \leq \text{Field } r$
 shows *ofilter*(*Under* A)
by (*clarsimp simp: ofilter-def*) (*meson TRANS Under-Field subset-eq under-Under-trans*)

lemma *ofilter-UnderS*[*simp*]:
 assumes $A \leq \text{Field } r$
 shows *ofilter*(*UnderS* A)
by (*clarsimp simp: ofilter-def*) (*meson ANTISYM TRANS UnderS-Field subset-eq under-UnderS-trans*)

lemma *ofilter-Int[simp]*: $\llbracket \text{ofilter } A; \text{ ofilter } B \rrbracket \implies \text{ofilter}(A \text{ Int } B)$
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-Un[simp]*: $\llbracket \text{ofilter } A; \text{ ofilter } B \rrbracket \implies \text{ofilter}(A \cup B)$
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-INTER*:
 $\llbracket I \neq \{\}; \bigwedge i. i \in I \implies \text{ofilter}(A \ i) \rrbracket \implies \text{ofilter}(\bigcap i \in I. A \ i)$
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-Inter*:
 $\llbracket S \neq \{\}; \bigwedge A. A \in S \implies \text{ofilter } A \rrbracket \implies \text{ofilter}(\bigcap S)$
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-Union*:
 $(\bigwedge A. A \in S \implies \text{ofilter } A) \implies \text{ofilter}(\bigcup S)$
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-under-Union*:
 $\text{ofilter } A \implies A = \bigcup \{\text{under } a \mid a. a \in A\}$
using *ofilter-under-UNION* [of *A*] **by** *auto*

5.2.6 Other properties

lemma *Trans-Under-regressive*:
assumes *NE*: $A \neq \{\}$ **and** *SUB*: $A \leq \text{Field } r$
shows $\text{Under}(\text{Under } A) \leq \text{Under } A$
by (*metis* *INT-E* *NE* *REFL* *Refl-under-Under* *SUB* *empty-iff minim-Under minim-Under-under subsetI*)

lemma *ofilter-suc-Field*:
assumes *OF*: $\text{ofilter } A$ **and** *NE*: $A \neq \text{Field } r$
shows $\text{ofilter}(A \cup \{\text{suc } A\})$
by (*metis* *NE* *OF* *REFL* *Refl-under-underS* *ofilter-underS-Field* *suc-underS under-ofilter*)

declare
minim-in[simp]
minim-inField[simp]
minim-least[simp]
under-ofilter[simp]
underS-ofilter[simp]
Field-ofilter[simp]

end

abbreviation *worec* \equiv *wo-rel.worec*

abbreviation *adm-wo* \equiv *wo-rel.adm-wo*
abbreviation *isMinim* \equiv *wo-rel.isMinim*
abbreviation *minim* \equiv *wo-rel.minim*
abbreviation *max2* \equiv *wo-rel.max2*
abbreviation *supr* \equiv *wo-rel.supr*
abbreviation *suc* \equiv *wo-rel.suc*

end

6 Well-Order Embeddings

theory *Wellorder-Embedding*
imports *Fun-More Wellorder-Relation*
begin

6.1 Auxiliaries

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

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

lemma *embed-halfcong*:
assumes $\bigwedge a. a \in \text{Field } r \implies f \ a = g \ a$ **and** *embed r r' f*
shows *embed r r' g*
by (*smt (verit, del-insts) assms bij-betw-cong embed-def in-mono under-Field*)

lemma *embed-cong[fundef-cong]*:
assumes $\bigwedge a. a \in \text{Field } r \implies f \ a = g \ a$
shows $\text{embed } r \ r' \ f = \text{embed } r \ r' \ g$
by (*metis assms embed-halfcong*)

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

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

lemma *comp-compat*:
 $\llbracket \text{compat } r\ r'\ f; \text{compat } r'\ r''\ f \rrbracket \implies \text{compat } r\ r''\ (f' \circ f)$
by(*auto simp add: comp-def compat-def*)

corollary *one-set-greater*:
 $(\exists f::'a \Rightarrow 'a'. f\ 'A \leq A' \wedge \text{inj-on } f\ A) \vee (\exists g::'a' \Rightarrow 'a. g\ 'A' \leq A \wedge \text{inj-on } g\ A')$

proof –

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

qed

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

6.3 Uniqueness of embeddings

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* $r\ r''\ (f' \circ f)$

using *EMB EMB' WELL WELL' embedS-comp-embed embedS-def* **by** *blast*

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

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

lemma *not-embedS-iso*:
 $\neg (embedS\ r\ r'\ f \wedge iso\ r\ r'\ f)$
unfolding *embedS-def iso-def* **by** *blast*

lemma *embed-embedS-iff-not-iso*:
assumes *embed r r' f*
shows $embedS\ r\ r'\ f = (\neg iso\ r\ r'\ f)$
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 $iso\ r'\ r\ (inv\ into\ (Field\ r)\ f)$
by (*meson ISO WELL bij-betw-inv-into inv-into-Field-embed-bij-betw iso-def iso-iff iso-iff2*)

lemma *embedS-or-iso*:
assumes *WELL: Well-order r and WELL': Well-order r'*
shows $(\exists g. embedS\ r\ r'\ g) \vee (\exists h. embedS\ r'\ r\ h) \vee (\exists f. iso\ r\ r'\ f)$
by (*metis WELL WELL' embed-embedS-iso embed-embedS-iso iso-iff4 wellorders-totally-ordered*)

end

7 Order Union

theory *Order-Union*

imports *Main*

begin

definition *Osum* :: $'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel$ (**infix** $\langle Osum \rangle$ 60) **where**
 $r\ Osum\ r' = r \cup r' \cup \{(a, a').\ a \in Field\ r \wedge a' \in Field\ r'\}$

notation *Osum* (**infix** $\langle \cup o \rangle$ 60)

lemma *Field-Osum*: $Field\ (r\ \cup o\ r') = Field\ r \cup Field\ r'$
unfolding *Osum-def Field-def* **by** *blast*

lemma *Osum-wf*:

assumes *FLD*: $\text{Field } r \text{ Int Field } r' = \{\}$ **and**
 WF : $\text{wf } r$ **and** WF' : $\text{wf } r'$
shows $\text{wf } (r \text{ Osum } r')$
unfolding *wf-eq-minimal2* **unfolding** *Field-Osum*
proof(*intro allI impI, elim conjE*)
fix A **assume** *: $A \subseteq \text{Field } r \cup \text{Field } r'$ **and** **: $A \neq \{\}$
obtain B **where** $B\text{-def}$: $B = A \text{ Int Field } r$ **by** *blast*
show $\exists a \in A. \forall a' \in A. (a', a) \notin r \cup r'$
proof(*cases* $B = \{\}$)
assume Case1 : $B \neq \{\}$
hence $B \neq \{\} \wedge B \leq \text{Field } r$ **using** $B\text{-def}$ **by** *auto*
then obtain a **where** 1 : $a \in B$ **and** 2 : $\forall a1 \in B. (a1, a) \notin r$
using WF **unfolding** *wf-eq-minimal2* **by** *blast*
hence 3 : $a \in \text{Field } r \wedge a \notin \text{Field } r'$ **using** $B\text{-def}$ *FLD* **by** *auto*

have $\forall a1 \in A. (a1, a) \notin r \text{ Osum } r'$
proof(*intro ballI*)
fix $a1$ **assume** **: $a1 \in A$
{assume Case11 : $a1 \in \text{Field } r$
hence $(a1, a) \notin r$ **using** $B\text{-def}$ ** 2 **by** *auto*
moreover
have $(a1, a) \notin r'$ **using** 3 **by** (*auto simp add: Field-def*)
ultimately have $(a1, a) \notin r \text{ Osum } r'$
using 3 **unfolding** *Osum-def* **by** *auto*
}
moreover
{assume Case12 : $a1 \notin \text{Field } r$
hence $(a1, a) \notin r$ **unfolding** *Field-def* **by** *auto*
moreover
have $(a1, a) \notin r'$ **using** 3 **unfolding** *Field-def* **by** *auto*
ultimately have $(a1, a) \notin r \text{ Osum } r'$
using 3 **unfolding** *Osum-def* **by** *auto*
}
ultimately show $(a1, a) \notin r \text{ Osum } r'$ **by** *blast*
qed
thus *?thesis* **using** 1 $B\text{-def}$ **by** *auto*
next
assume Case2 : $B = \{\}$
hence 1 : $A \neq \{\} \wedge A \leq \text{Field } r'$ **using** * ** $B\text{-def}$ **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 : $a' \in \text{Field } r' \wedge a' \notin \text{Field } r$ **using** 1 *FLD* **by** *blast*

have $\forall a1' \in A. (a1', a') \notin r \text{ Osum } r'$
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** $a1' \in \text{Field } r$

thus *False* using *Case2 B-def* by *auto*
 qed
 thus *?thesis* using 2 by *blast*
 qed
 qed

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')*
 using *assms* unfolding *Osum-def trans-def disjoint-iff Field-iff* by *blast*

lemma *Osum-Preorder*:
 $\llbracket \text{Field } r \text{ Int Field } r' = \{\}; \text{Preorder } r; \text{Preorder } r' \rrbracket \implies \text{Preorder } (r \text{ Osum } r')$
 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')*
 using *assms* by (*auto simp: disjoint-iff antisym-def Osum-def Field-def*)

lemma *Osum-Partial-order*:
 $\llbracket \text{Field } r \text{ Int Field } r' = \{\}; \text{Partial-order } r; \text{Partial-order } r' \rrbracket \implies$
 $\text{Partial-order } (r \text{ Osum } r')$
 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*:
 $\llbracket \text{Field } r \text{ Int Field } r' = \{\}; \text{Linear-order } r; \text{Linear-order } r' \rrbracket \implies \text{Linear-order } (r$
 $\text{Osum } r')$
 by (*simp add: Osum-Partial-order Osum-Total linear-order-on-def*)

lemma *Osum-minus-Id1*:
 assumes $r \leq \text{Id}$
 shows $(r \text{ Osum } r') - \text{Id} \leq (r' - \text{Id}) \cup (\text{Field } r \times \text{Field } r')$

using *assms* **by** (*force simp: Osum-def*)

lemma *Osum-minus-Id2*:
 assumes $r' \leq Id$
 shows $(r \text{ Osum } r') - Id \leq (r - Id) \cup (Field\ r \times Field\ r')$
using *assms* **by** (*force simp: Osum-def*)

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 $(r \text{ Osum } r') - Id \leq (r - Id) \text{ Osum } (r' - Id)$
using *assms* *Total-Id-Field* **by** (*force simp: Osum-def*)

lemma *wf-Int-Times*:
 assumes $A \text{ Int } B = \{\}$
 shows $wf(A \times B)$
unfolding *wf-def* **using** *assms* **by** *blast*

lemma *Osum-wf-Id*:
 assumes *TOT*: Total r and *TOT'*: Total r' and
FLD: $Field\ r \text{ Int } Field\ r' = \{\}$ and
WF: $wf(r - Id)$ and *WF'*: $wf(r' - Id)$
 shows $wf((r \text{ Osum } r') - Id)$
proof(*cases* $r \leq Id \vee r' \leq Id$)
 assume *Case1*: $\neg (r \leq Id \vee r' \leq Id)$
 have $Field(r - Id) \text{ Int } Field(r' - Id) = \{\}$
using *Case1 FLD TOT TOT' Total-Id-Field* **by** *blast*
 thus *?thesis*
 by (*meson Case1 Osum-minus-Id Osum-wf TOT TOT' WF WF' wf-subset*)
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 $(r \text{ Osum } r') - Id \leq (r' - Id) \cup (Field\ r \times Field\ r')$
using *Osum-minus-Id1* [*of* $r\ r'$] **by** *simp*
moreover
 {have $Domain(Field\ r \times Field\ r') \text{ Int } Range(r' - Id) = \{\}$
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* **using** *wf-subset* **by** *blast*
 }
moreover
 {assume *Case22*: $r' \leq Id$
 hence $(r \text{ Osum } r') - Id \leq (r - Id) \cup (Field\ r \times Field\ r')$


```

    using Osum-minus-Id2[of  $r' r$ ] by simp
  moreover
  {have Range(Field  $r \times \text{Field } r'$ ) Int Domain( $r - \text{Id}$ ) = {}
    using FLD unfolding Field-def by blast
    hence wf(( $r - \text{Id}$ )  $\cup$  (Field  $r \times \text{Field } r'$ ))
      using 1 WF wf-Un[of  $r - \text{Id Field } r \times \text{Field } r'$ ]
      by (auto simp add: Un-commute)
  }
  ultimately have ?thesis using wf-subset by blast
}
ultimately show ?thesis by blast
qed

```

```

lemma Osum-Well-order:
  assumes FLD: Field  $r \text{ Int Field } r' = \{\}$  and
    WELL: Well-order  $r$  and WELL': Well-order  $r'$ 
  shows Well-order ( $r \text{ Osum } r'$ )
proof-
  have Total  $r \wedge \text{Total } r'$  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
end

```

8 Constructions on Wellorders

```

theory Wellorder-Constructions
  imports
    Wellorder-Embedding Order-Union
begin

unbundle cardinal-syntax

```

```

declare
  ordLeq-Well-order-simp[simp]
  not-ordLeq-iff-ordLess[simp]
  not-ordLess-iff-ordLeq[simp]
  Func-empty[simp]
  Func-is-emp[simp]

```

8.1 Order filters versus restrictions and embeddings

```

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

```

8.2 Ordering the well-orders by existence of embeddings

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

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*)

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

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

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

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

corollary *ordIso-equiv*: *equiv* {*r*. *Well-order r*} *ordIso*
 using *ordIso-subset* *ordIso-refl-on* *ordIso-sym* *ordIso-trans* **by** (*intro equivI*)

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*

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*

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

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*

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 *ofilter-ordLeq*:
 assumes Well-order *r* and *ofilter* *r* *A*
 shows *Restr* *r* *A* $\leq_o r$
by (*metis assms inf.orderE ofilter-embed ofilter-subset-ordLeq refl-on-def wo-rel.Field-ofilter*
wo-rel.REFL wo-rel.intro)

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

8.3 Copy via direct images

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*
 by (*clarsimp simp: dir-image-def*) (*metis FieldI1 FieldI2 UnCI assms inj-on-def*)
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 *FLD*
 by (auto simp add: *wo-rel-def wo-rel.ofilter-def Osum-def under-def Field-iff disjoint-iff*)
 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*: $f \text{ ' } A \leq B$
 shows $\exists r'. \text{ well-order-on } B \text{ } r' \wedge r \leq_o r'$
proof–
 have *bij-betw* f A ($f \text{ ' } A$)
 using *INJ inj-on-imp-bij-betw* by *blast*
 then obtain r'' where *well-order-on* ($f \text{ ' } A$) r'' and 1: $r =_o r''$
 using *WELL Well-order-iso-copy* by *blast*
 hence 2: *Well-order* $r'' \wedge \text{Field } r'' = (f \text{ ' } A)$
 using *well-order-on-Well-order* by *blast*

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

8.4 The maxim among a finite set of ordinals

The correct phrasing 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* $\equiv r \in R \wedge (\forall r' \in R. r' \leq_o r)$

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 $\forall r \in R. \text{Well-order } r$
shows $\exists r. \text{isOmax } R \ r$

proof –

have *finite* *R* $\implies R \neq \{\} \longrightarrow (\forall r \in R. \text{Well-order } r) \longrightarrow (\exists r. \text{isOmax } R \ r)$
apply(*erule finite-induct*) apply(*simp add: isOmax-def*)

proof(*clarsimp*)

fix *r* :: ('a \times 'a) set and *R* assume *: *finite* *R* and **: $r \notin R$
and ***: *Well-order* *r* and ****: $\forall r \in R. \text{Well-order } r$
and *IH*: $R \neq \{\} \longrightarrow (\exists p. \text{isOmax } R \ p)$

let $?R' = \text{insert } r \ R$

show $\exists p'. (\text{isOmax } ?R' \ p')$

proof(*cases* $R = \{\}$)

case *True*

thus *?thesis*

by (*simp add: *** isOmax-def ordLeq-reflexive*)

next

case *False*

then obtain *p* where *p*: *isOmax* *R* *p* using *IH* by *auto*

hence *Well-order* *p* using **** unfolding *isOmax-def* by *simp*

then consider $r \leq_o p \mid p \leq_o r$

using *** *ordLeq-total* by *auto*

then show *?thesis*

proof *cases*

case 1

then show *?thesis*

using *p* unfolding *isOmax-def* by *auto*

next

case 2

then show *?thesis*

by (*metis *** insert-iff isOmax-def ordLeq-reflexive ordLeq-transitive p*)

qed

qed

qed

thus *?thesis* using *assms* by *auto*

qed

lemma *omax-isOmax*:

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

lemma *omax-in*:

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

lemma *Well-order-omax*:

assumes *finite R* and $R \neq \{\}$ and $\forall r \in R. \text{Well-order } r$
shows $\text{Well-order (omax } R)$
using *assms omax-in* by *blast*

lemma *omax-maxim*:

assumes *finite R* and $\forall r \in R. \text{Well-order } r$ and $r \in R$
shows $r \leq_o \text{omax } R$
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 $\text{omax } R \leq_o p$
by (*meson assms omax-in ordLeq-Well-order-simp*)

lemma *omax-ordLess*:

assumes *finite R* and $R \neq \{\}$ and $\forall r \in R. r <_o p$
shows $\text{omax } R <_o p$
by (*meson assms omax-in ordLess-Well-order-simp*)

lemma *omax-ordLeq-elim*:

assumes *finite R* and $\forall r \in R. \text{Well-order } r$
and $\text{omax } R \leq_o p$ and $r \in R$
shows $r \leq_o p$
by (*meson assms omax-maxim ordLeq-transitive*)

lemma *omax-ordLess-elim*:

assumes *finite R* and $\forall r \in R. \text{Well-order } r$
and $\text{omax } R <_o p$ and $r \in R$
shows $r <_o p$
by (*meson assms omax-maxim ordLeq-ordLess-trans*)

lemma *ordLeq-omax*:

assumes *finite R* and $\forall r \in R. \text{Well-order } r$
and $r \in R$ and $p \leq_o r$
shows $p \leq_o \text{omax } R$

by (*meson* *assms* *omax-maxim* *ordLeq-transitive*)

lemma *ordLess-omax*:

assumes *finite* *R* **and** $\forall r \in R.$ *Well-order* *r*

and $r \in R$ **and** $p <_o r$

shows $p <_o \text{omax } R$

by (*meson* *assms* *omax-maxim* *ordLess-ordLeq-trans*)

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 $\text{omax } P \leq_o \text{omax } R$

by (*meson* *LEQ* *NE-P* *P* *R* *Well-R* *omax-ordLeq* *ordLeq-omax*)

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 $\text{omax } P <_o \text{omax } R$

by (*meson* *LEQ* *NE-P* *P* *R* *Well-R* *omax-ordLess* *ordLess-omax*)

8.5 Limit and succesor ordinals

lemma *embed-underS2*:

assumes *r*: *Well-order* *r* **and** *g*: *embed* *r* *s* *g* **and** *a*: $a \in \text{Field } r$

shows $g \text{ 'underS } r \text{ } a = \text{underS } s \text{ } (g \text{ } a)$

by (*meson* *a* *bij-betw-def* *embed-underS* *g* *r*)

lemma *bij-betw-insert*:

assumes $b \notin A$ **and** $f \text{ } b \notin A'$ **and** *bij-betw* *f* *A* *A'*

shows *bij-betw* *f* (*insert* *b* *A*) (*insert* (*f* *b*) *A'*)

using *notIn-Un-bij-betw*[*OF* *assms*] **by** *auto*

context *wo-rel*

begin

lemma *underS-induct*:

assumes $\bigwedge a. (\bigwedge a1. a1 \in \text{underS } a \implies P \text{ } a1) \implies P \text{ } a$

shows $P \text{ } a$

by (*induct* *rule*: *well-order-induct*) (*rule* *assms*, *simp* *add*: *underS-def*)

lemma *suc-underS'*:

assumes *B*: $B \subseteq \text{Field } r$ **and** *A*: *AboveS* *B* $\neq \{\}$ **and** *b*: $b \in B$

shows $b \in \text{underS } (\text{suc } B)$

using *suc-AboveS*[*OF* *B* *A*] *b* **unfolding** *underS-def* *AboveS-def* **by** *auto*

lemma *underS-supr*:

assumes *bA*: $b \in \text{underS } (\text{supr } A)$ **and** *A*: $A \subseteq \text{Field } r$

shows $\exists a \in A. b \in \text{underS } a$
proof(rule ccontr, simp)
have $bb: b \in \text{Field } r$ **using** bA **unfolding** $\text{underS-def Field-def}$ **by** *auto*
assume $\forall a \in A. b \notin \text{underS } a$
hence $0: \forall a \in A. (a, b) \in r$ **using** A bA **unfolding** underS-def
by *simp (metis REFL in-mono max2-def max2-greater refl-on-domain)*
have $(\text{supr } A, b) \in r$
by (*simp add: 0 A bb supr-least*)
thus *False*
by (*metis antisymD bA underS-E wo-rel.ANTISYM wo-rel-axioms*)
qed

lemma *underS-suc*:
assumes $bA: b \in \text{underS } (\text{suc } A)$ **and** $A: A \subseteq \text{Field } r$
shows $\exists a \in A. b \in \text{under } a$
proof(rule ccontr, simp)
have $bb: b \in \text{Field } r$ **using** bA **unfolding** $\text{underS-def Field-def}$ **by** *auto*
assume $\forall a \in A. b \notin \text{under } a$
hence $0: \forall a \in A. a \in \text{underS } b$ **using** A bA
by (*metis bb in-mono max2-def max2-greater mem-Collect-eq underS-I under-def*)
have $(\text{suc } A, b) \in r$
by (*metis 0 A bb suc-least underS-E*)
thus *False*
by (*metis antisymD bA underS-E wo-rel.ANTISYM wo-rel-axioms*)
qed

lemma (*in wo-rel*) *in-underS-supr*:
assumes $j \in \text{underS } i$ **and** $i \in A$ **and** $A \subseteq \text{Field } r$ **and** *Above* $A \neq \{\}$
shows $j \in \text{underS } (\text{supr } A)$
by (*meson assms LIN in-mono supr-greater supr-inField underS-incl-iff*)

lemma *inj-on-Field*:
assumes $A: A \subseteq \text{Field } r$ **and** $f: \bigwedge a b. \llbracket a \in A; b \in A; a \in \text{underS } b \rrbracket \implies f a \neq f b$
shows *inj-on* f A
by (*smt (verit) A f in-notinI inj-on-def subsetD underS-I*)

lemma *ofilter-init-seg-of*:
assumes *ofilter* F
shows *Restr* r F *initial-segment-of* r
using *assms* **unfolding** *ofilter-def init-seg-of-def under-def* **by** *auto*

lemma *underS-init-seg-of-Collect*:
assumes $\bigwedge j1 j2. \llbracket j2 \in \text{underS } i; (j1, j2) \in r \rrbracket \implies R$ $j1$ *initial-segment-of* R $j2$
shows $\{R$ j $| j. j \in \text{underS } i\} \in \text{Chains init-seg-of}$
using *TOTALS assms*
by (*clarsimp simp: Chains-def (meson BNF-Least-Fixpoint.underS-Field)*)

lemma (in *wo-rel*) *Field-init-seg-of-Collect*:
assumes $\bigwedge j1\ j2. \llbracket j2 \in \text{Field } r; (j1, j2) \in r \rrbracket \implies R\ j1\ \text{initial-segment-of } R\ j2$
shows $\{R\ j \mid j. j \in \text{Field } r\} \in \text{Chains init-seg-of}$
using *TOTALS* *assms* **by** (*auto simp: Chains-def*)

8.5.1 Successor and limit elements of an ordinal

definition $\text{succ } i \equiv \text{succ } \{i\}$

definition $\text{isSucc } i \equiv \exists j. \text{aboveS } j \neq \{\} \wedge i = \text{succ } j$

definition $\text{zero} = \text{minim } (\text{Field } r)$

definition $\text{isLim } i \equiv \neg \text{isSucc } i$

lemma *zero-smallest[simp]*:
assumes $j \in \text{Field } r$ **shows** $(\text{zero}, j) \in r$
by (*simp add: assms wo-rel.ofilter-linord wo-rel-axioms zero-def*)

lemma *zero-in-Field*: **assumes** $\text{Field } r \neq \{\}$ **shows** $\text{zero} \in \text{Field } r$
using *assms unfolding zero-def by (metis Field-ofilter minim-in ofilter-def)*

lemma *leq-zero-imp[simp]*:
 $(x, \text{zero}) \in r \implies x = \text{zero}$
by (*metis ANTISYM WELL antisymD well-order-on-domain zero-smallest*)

lemma *leq-zero[simp]*:
assumes $\text{Field } r \neq \{\}$ **shows** $(x, \text{zero}) \in r \longleftrightarrow x = \text{zero}$
using *zero-in-Field[OF assms] in-notinI[of x zero] by auto*

lemma *under-zero[simp]*:
assumes $\text{Field } r \neq \{\}$ **shows** $\text{under } \text{zero} = \{\text{zero}\}$
using *assms unfolding under-def by auto*

lemma *underS-zero[simp,intro]*: $\text{underS } \text{zero} = \{\}$
unfolding *underS-def* **by** *auto*

lemma *isSucc-succ*: $\text{aboveS } i \neq \{\} \implies \text{isSucc } (\text{succ } i)$
unfolding *isSucc-def succ-def* **by** *auto*

lemma *succ-in-diff*:
assumes $\text{aboveS } i \neq \{\}$ **shows** $(i, \text{succ } i) \in r \wedge \text{succ } i \neq i$
using *assms suc-greater[of {i}] unfolding succ-def AboveS-def aboveS-def Field-def*
by *auto*

lemmas $\text{succ-in}[simp] = \text{succ-in-diff}[THEN\ conjunct1]$
lemmas $\text{succ-diff}[simp] = \text{succ-in-diff}[THEN\ conjunct2]$

lemma *succ-in-Field[simp]*:

```

assumes  $\text{aboveS } i \neq \{\}$  shows  $\text{succ } i \in \text{Field } r$ 
using  $\text{succ-in}[OF \text{ assms}]$  unfolding  $\text{Field-def}$  by auto

lemma succ-not-in:
  assumes  $\text{aboveS } i \neq \{\}$  shows  $(\text{succ } i, i) \notin r$ 
  by (metis  $\text{FieldI2}$  assms  $\text{max2-equals1}$   $\text{max2-equals2}$   $\text{succ-diff}$   $\text{succ-in}$ )

lemma not-isSucc-zero:  $\neg \text{isSucc zero}$ 
  by (metis  $\text{isSucc-def}$   $\text{leq-zero-imp}$   $\text{succ-in-diff}$ )

lemma isLim-zero[simp]:  $\text{isLim zero}$ 
  by (metis  $\text{isLim-def}$  not-isSucc-zero)

lemma succ-smallest:
  assumes  $(i, j) \in r$  and  $i \neq j$ 
  shows  $(\text{succ } i, j) \in r$ 
  by (metis  $\text{Field-iff}$  assms  $\text{empty-subsetI}$   $\text{insert-subset}$   $\text{singletonD}$   $\text{suc-least}$   $\text{succ-def}$ )

lemma isLim-supr:
  assumes  $f: i \in \text{Field } r$  and  $l: \text{isLim } i$ 
  shows  $i = \text{supr } (\text{underS } i)$ 
proof(rule equals-supr)
  fix  $j$  assume  $j: j \in \text{Field } r$  and  $1: \bigwedge j'. j' \in \text{underS } i \implies (j', j) \in r$ 
  show  $(i, j) \in r$ 
  proof(intro in-notinI[OF - f j], safe)
    assume  $ji: (j, i) \in r$   $j \neq i$ 
    hence  $a: \text{aboveS } j \neq \{\}$  unfolding  $\text{aboveS-def}$  by auto
    hence  $i \neq \text{succ } j$  using  $l$  unfolding  $\text{isLim-def}$   $\text{isSucc-def}$  by auto
    moreover have  $(\text{succ } j, i) \in r$  using  $\text{succ-smallest}[OF ji]$  by auto
    ultimately have  $\text{succ } j \in \text{underS } i$  unfolding  $\text{underS-def}$  by auto
    hence  $(\text{succ } j, j) \in r$  using  $1$  by auto
    thus False using  $\text{succ-not-in}[OF a]$  by simp
  qed
qed(use f underS-def Field-def in fastforce)+

definition pred  $i \equiv \text{SOME } j. j \in \text{Field } r \wedge \text{aboveS } j \neq \{\} \wedge \text{succ } j = i$ 

lemma pred-Field-succ:
  assumes  $\text{isSucc } i$  shows  $\text{pred } i \in \text{Field } r \wedge \text{aboveS } (\text{pred } i) \neq \{\} \wedge \text{succ } (\text{pred } i) = i$ 
proof–
  obtain  $j$  where  $j: \text{aboveS } j \neq \{\}$   $i = \text{succ } j$ 
  using assms unfolding  $\text{isSucc-def}$  by auto
  then obtain  $j \in \text{Field } r$   $j \neq i$ 
  by (metis  $\text{FieldI1}$   $\text{succ-diff}$   $\text{succ-in}$ )
  then show ?thesis unfolding  $\text{pred-def}$ 
  by (metis (mono-tags, lifting)  $j$  someI-ex)
qed

```

lemmas $\text{pred-Field}[simp] = \text{pred-Field-succ}[THEN\ conjunct1]$
lemmas $\text{aboveS-pred}[simp] = \text{pred-Field-succ}[THEN\ conjunct2, THEN\ conjunct1]$
lemmas $\text{succ-pred}[simp] = \text{pred-Field-succ}[THEN\ conjunct2, THEN\ conjunct2]$

lemma isSucc-pred-in :
assumes $\text{isSucc } i$ **shows** $(\text{pred } i, i) \in r$
by $(metis\ assms\ \text{pred-Field-succ}\ \text{succ-in})$

lemma isSucc-pred-diff :
assumes $\text{isSucc } i$ **shows** $\text{pred } i \neq i$
by $(metis\ \text{aboveS-pred}\ assms\ \text{succ-diff}\ \text{succ-pred})$

lemma $\text{succ-inj}[simp]$:
assumes $\text{aboveS } i \neq \{\}$ **and** $\text{aboveS } j \neq \{\}$
shows $\text{succ } i = \text{succ } j \longleftrightarrow i = j$
by $(metis\ \text{FieldI1}\ assms\ \text{succ-def}\ \text{succ-in}\ \text{supr-under}\ \text{under-underS-suc})$

lemma $\text{pred-succ}[simp]$:
assumes $\text{aboveS } j \neq \{\}$ **shows** $\text{pred } (\text{succ } j) = j$
using $assms\ \text{isSucc-def}\ \text{pred-Field-succ}\ \text{succ-inj}$ **by** blast

lemma $\text{less-succ}[simp]$:
assumes $\text{aboveS } i \neq \{\}$
shows $(j, \text{succ } i) \in r \longleftrightarrow (j, i) \in r \vee j = \text{succ } i$
by $(metis\ \text{FieldI1}\ assms\ \text{in-notinI}\ \text{max2-equals1}\ \text{max2-equals2}\ \text{max2-iff}\ \text{succ-in}\ \text{succ-smallest})$

lemma $\text{underS-succ}[simp]$:
assumes $\text{aboveS } i \neq \{\}$
shows $\text{underS } (\text{succ } i) = \text{under } i$
unfolding $\text{underS-def}\ \text{under-def}$ **by** $(auto\ simp:\ assms\ \text{succ-not-in})$

lemma succ-mono :
assumes $\text{aboveS } j \neq \{\}$ **and** $(i, j) \in r$
shows $(\text{succ } i, \text{succ } j) \in r$
by $(metis\ (\text{full-types})\ assms\ \text{less-succ}\ \text{succ-smallest})$

lemma $\text{under-succ}[simp]$:
assumes $\text{aboveS } i \neq \{\}$
shows $\text{under } (\text{succ } i) = \text{insert } (\text{succ } i) (\text{under } i)$
using $\text{less-succ}[OF\ assms]$ **unfolding** under-def **by** auto

definition $\text{mergeSL} :: ('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow (('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$
where
 $\text{mergeSL } S\ L\ f\ i \equiv \text{if } \text{isSucc } i \text{ then } S\ (\text{pred } i)\ (f\ (\text{pred } i)) \text{ else } L\ f\ i$

8.5.2 Well-order recursion with (zero), succesor, and limit

definition $worecSL :: ('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow (('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$
where $worecSL S L \equiv worec (mergeSL S L)$

definition $adm-woL L \equiv \forall f g i. isLim i \wedge (\forall j \in underS i. f j = g j) \longrightarrow L f i = L g i$

lemma $mergeSL: adm-woL L \Longrightarrow adm-wo (mergeSL S L)$

unfolding $adm-wo-def adm-woL-def isLim-def$
by ($smt (verit, ccfv-threshold) isSucc-pred-diff isSucc-pred-in mergeSL-def underS-I$)

lemma $worec-fixpoint1: adm-wo H \Longrightarrow worec H i = H (worec H) i$
by ($metis worec-fixpoint$)

lemma $worecSL-isSucc:$

assumes $a: adm-woL L$ **and** $i: isSucc i$
shows $worecSL S L i = S (pred i) (worecSL S L (pred i))$
by ($metis a i mergeSL mergeSL-def worecSL-def worec-fixpoint$)

lemma $worecSL-succ:$

assumes $a: adm-woL L$ **and** $i: aboveS j \neq \{\}$
shows $worecSL S L (succ j) = S j (worecSL S L j)$
by ($simp add: a i isSucc-succ worecSL-isSucc$)

lemma $worecSL-isLim:$

assumes $a: adm-woL L$ **and** $i: isLim i$
shows $worecSL S L i = L (worecSL S L) i$
by ($metis a i isLim-def mergeSL mergeSL-def worecSL-def worec-fixpoint$)

definition $worecZSL :: 'b \Rightarrow ('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow (('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$
where $worecZSL Z S L \equiv worecSL S (\lambda f a. if a = zero then Z else L f a)$

lemma $worecZSL-zero:$

assumes $a: adm-woL L$
shows $worecZSL Z S L zero = Z$
by ($smt (verit, best) adm-woL-def assms isLim-zero worecSL-isLim worecZSL-def$)

lemma $worecZSL-succ:$

assumes $a: adm-woL L$ **and** $i: aboveS j \neq \{\}$
shows $worecZSL Z S L (succ j) = S j (worecZSL Z S L j)$
unfolding $worecZSL-def$
by ($smt (verit, best) a adm-woL-def i worecSL-succ$)

lemma $worecZSL-isLim:$

assumes $a: adm-woL L$ **and** $isLim i$ **and** $i \neq zero$
shows $worecZSL Z S L i = L (worecZSL Z S L) i$
proof–
let $?L = \lambda f a. if a = zero then Z else L f a$

have $worecZSL\ Z\ S\ L\ i = ?L\ (worecZSL\ Z\ S\ L)\ i$
unfolding $worecZSL\text{-}def$ **by** $(smt\ (verit,\ best)\ adm\text{-}woL\text{-}def\ assms\ worecSL\text{-}isLim)$
also have $\dots = L\ (worecZSL\ Z\ S\ L)\ i$ **using** $assms$ **by** $simp$
finally show $?thesis$.
qed

8.5.3 Well-order succ-lim induction

lemma $ord\text{-}cases$:

obtains j **where** $i = succ\ j$ **and** $aboveS\ j \neq \{\}$ **|** $isLim\ i$
by $(metis\ isLim\text{-}def\ isSucc\text{-}def)$

lemma $well\text{-}order\text{-}inductSL[case\text{-}names\ Suc\ Lim]$:

assumes $\bigwedge i. \llbracket aboveS\ i \neq \{\};\ P\ i \rrbracket \implies P\ (succ\ i)$ $\bigwedge i. \llbracket isLim\ i;\ \bigwedge j. j \in underS\ i \implies P\ j \rrbracket \implies P\ i$

shows $P\ i$

proof $(induction\ rule:\ well\text{-}order\text{-}induct)$

case $(1\ x)$

then show $?case$

by $(metis\ assms\ ord\text{-}cases\ succ\text{-}diff\ succ\text{-}in\ underS\text{-}E)$

qed

lemma $well\text{-}order\text{-}inductZSL[case\text{-}names\ Zero\ Suc\ Lim]$:

assumes $P\ zero$

and $\bigwedge i. \llbracket aboveS\ i \neq \{\};\ P\ i \rrbracket \implies P\ (succ\ i)$ **and**

$\bigwedge i. \llbracket isLim\ i;\ i \neq zero;\ \bigwedge j. j \in underS\ i \implies P\ j \rrbracket \implies P\ i$

shows $P\ i$

by $(metis\ assms\ well\text{-}order\text{-}inductSL)$

definition $isSuccOrd \equiv \exists\ j \in Field\ r. \forall\ i \in Field\ r. (i,j) \in r$

definition $isLimOrd \equiv \neg\ isSuccOrd$

lemma $isLimOrd\text{-}succ$:

assumes $isLimOrd$ **and** $i \in Field\ r$

shows $succ\ i \in Field\ r$

using $assms$ **unfolding** $isLimOrd\text{-}def\ isSuccOrd\text{-}def$

by $(metis\ REFL\ in\text{-}notinI\ refl\text{-}on\text{-}domain\ succ\text{-}smallest)$

lemma $isLimOrd\text{-}aboveS$:

assumes $l:\ isLimOrd$ **and** $i:\ i \in Field\ r$

shows $aboveS\ i \neq \{\}$

proof—

obtain j **where** $j \in Field\ r$ **and** $(j,i) \notin r$

using $assms$ **unfolding** $isLimOrd\text{-}def\ isSuccOrd\text{-}def$ **by** $auto$

hence $(i,j) \in r \wedge j \neq i$ **by** $(metis\ i\ max2\text{-}def\ max2\text{-}greater)$

thus $?thesis$ **unfolding** $aboveS\text{-}def$ **by** $auto$

qed

lemma *succ-aboveS-isLimOrd*:
assumes $\forall i \in \text{Field } r. \text{aboveS } i \neq \{\}$ $\wedge \text{succ } i \in \text{Field } r$
shows *isLimOrd*
using *assms isLimOrd-def isSuccOrd-def succ-not-in* **by** *blast*

lemma *isLim-iff*:
assumes *l*: *isLim* *i* **and** *j*: $j \in \text{underS } i$
shows $\exists k. k \in \text{underS } i \wedge j \in \text{underS } k$
by (*metis Order-Relation.underS-Field empty-iff isLim-supr j l underS-empty underS-supr*)

end

abbreviation *zero* $\equiv \text{wo-rel.zero}$
abbreviation *succ* $\equiv \text{wo-rel.succ}$
abbreviation *pred* $\equiv \text{wo-rel.pred}$
abbreviation *isSucc* $\equiv \text{wo-rel.isSucc}$
abbreviation *isLim* $\equiv \text{wo-rel.isLim}$
abbreviation *isLimOrd* $\equiv \text{wo-rel.isLimOrd}$
abbreviation *isSuccOrd* $\equiv \text{wo-rel.isSuccOrd}$
abbreviation *adm-woL* $\equiv \text{wo-rel.adm-woL}$
abbreviation *worecSL* $\equiv \text{wo-rel.worecSL}$
abbreviation *worecZSL* $\equiv \text{wo-rel.worecZSL}$

8.6 Projections of wellorders

definition *oproj* $r \ s \ f \equiv \text{Field } s \subseteq f^{-1}(\text{Field } r) \wedge \text{compat } r \ s \ f$

lemma *oproj-in*:
assumes *oproj* $r \ s \ f$ **and** $(a, a') \in r$
shows $(f \ a, f \ a') \in s$
using *assms unfolding oproj-def compat-def* **by** *auto*

lemma *oproj-Field*:
assumes *f*: *oproj* $r \ s \ f$ **and** *a*: $a \in \text{Field } r$
shows $f \ a \in \text{Field } s$
using *oproj-in[OF f] a unfolding Field-def* **by** *auto*

lemma *oproj-Field2*:
assumes *f*: *oproj* $r \ s \ f$ **and** *a*: $b \in \text{Field } s$
shows $\exists a \in \text{Field } r. f \ a = b$
using *assms unfolding oproj-def* **by** *auto*

lemma *oproj-under*:
assumes *f*: *oproj* $r \ s \ f$ **and** *a*: $a \in \text{under } r \ a'$
shows $f \ a \in \text{under } s \ (f \ a')$
using *oproj-in[OF f] a unfolding under-def* **by** *auto*

theorem *embedI*:

assumes *r*: *Well-order r* **and** *s*: *Well-order s*

and *f*: $\bigwedge a. a \in \text{Field } r \implies f a \in \text{Field } s \wedge f \text{ ' underS } r a \subseteq \text{underS } s (f a)$

shows $\exists g. \text{embed } r s g$

proof –

interpret *r*: *wo-rel r* **by** *unfold-locales (rule r)*

interpret *s*: *wo-rel s* **by** *unfold-locales (rule s)*

let *?G* = $\lambda g a. \text{suc } s (g \text{ ' underS } r a)$

define *g* **where** *g* = *worec r ?G*

have *adm*: *adm-wo r ?G* **unfolding** *r.adm-wo-def image-def* **by** *auto*

{fix *a* **assume** *a* $\in \text{Field } r$

hence *bij-betw g (under r a) (under s (g a))* \wedge

g a $\in \text{under } s (f a)$

proof(*induction a rule: r.underS-induct*)

case (1 *a*)

hence *a*: *a* $\in \text{Field } r$

and *IH1a*: $\bigwedge a1. a1 \in \text{underS } r a \implies \text{inj-on } g (\text{under } r a1)$

and *IH1b*: $\bigwedge a1. a1 \in \text{underS } r a \implies g \text{ ' under } r a1 = \text{under } s (g a1)$

and *IH2*: $\bigwedge a1. a1 \in \text{underS } r a \implies g a1 \in \text{under } s (f a1)$

unfolding *underS-def Field-def bij-betw-def* **by** *auto*

have *fa*: *f a* $\in \text{Field } s$ **using** *f[OF a]* **by** *auto*

have *g*: *g a* = *suc s (g ' underS r a)*

using *r.worec-fixpoint[OF adm]* **unfolding** *g-def fun-eq-iff* **by** *blast*

have *A0*: *g ' underS r a* $\subseteq \text{Field } s$

using *IH1b* **by** (*metis IH2 image-subsetI in-mono under-Field*)

{fix *a1* **assume** *a1*: *a1* $\in \text{underS } r a$

from *IH2[OF this]* **have** *g a1* $\in \text{under } s (f a1)$.

moreover **have** *f a1* $\in \text{underS } s (f a)$ **using** *f[OF a] a1* **by** *auto*

ultimately **have** *g a1* $\in \text{underS } s (f a)$ **by** (*metis s.ANTISYM s.TRANS*

under-underS-trans)

}

hence *fa-in*: *f a* $\in \text{AboveS } s (g \text{ ' underS } r a)$ **unfolding** *AboveS-def*

using *fa* **by** *simp (metis (lifting, full-types) mem-Collect-eq underS-def)*

hence *A*: *AboveS s (g ' underS r a)* $\neq \{\}$ **by** *auto*

have *ga*: *g a* $\in \text{Field } s$ **unfolding** *g* **using** *s.suc-inField[OF A0 A]* .

show *?case*

unfolding *bij-betw-def*

proof (*intro conjI*)

show *inj-on g (r.under a)*

by (*metis A IH1a IH1b a bij-betw-def g ga r s s.suc-greater subsetI*

wellorders-totally-ordered-aux)

show *g ' r.under a* = *s.under (g a)*

by (*metis A A0 IH1a IH1b a bij-betw-def g ga r s s.suc-greater wellorders-totally-ordered-aux*)

show *g a* $\in \text{s.under } (f a)$

by (*simp add: fa-in g s.suc-least-AboveS under-def*)

qed

qed

}

thus *?thesis* **unfolding** *embed-def* **by** *auto*
qed

corollary *ordLeq-def2*:

$r \leq_o s \iff \text{Well-order } r \wedge \text{Well-order } s \wedge$
 $(\exists f. \forall a \in \text{Field } r. f a \in \text{Field } s \wedge f ' \text{underS } r a \subseteq \text{underS } s (f a))$
using *embed-in-Field*[*of r s*] *embed-underS2*[*of r s*] *embedI*[*of r s*]
unfolding *ordLeq-def* **by** *fast*

lemma *iso-oproj*:

assumes *r*: *Well-order r* **and** *s*: *Well-order s* **and** *f*: *iso r s f*
shows *oproj r s f*
by (*metis embed-Field f iso-Field iso-iff iso-iff3 oproj-def r s*)

theorem *oproj-embed*:

assumes *r*: *Well-order r* **and** *s*: *Well-order s* **and** *f*: *oproj r s f*
shows $\exists g. \text{embed } s r g$
proof (*rule embedI*[*OF s r*, *of inv-into (Field r) f*], *unfold underS-def*, *safe*)
fix *b* **assume** $b \in \text{Field } s$
thus $\text{inv-into } (Field r) f b \in \text{Field } r$
using *oproj-Field2*[*OF f*] **by** (*metis imageI inv-into-into*)
next
fix *a b* **assume** $b \in \text{Field } s \wedge a \neq b \wedge (a, b) \in s$
 $\text{inv-into } (Field r) f a = \text{inv-into } (Field r) f b$
with *f* **show** *False*
by (*meson FieldI1 in-mono inv-into-injective oproj-def*)
next
fix *a b* **assume** $*: b \in \text{Field } s \wedge a \neq b \wedge (a, b) \in s$
 $\{ \text{assume } \text{notin}: (\text{inv-into } (Field r) f a, \text{inv-into } (Field r) f b) \notin r$
moreover
from $*(3)$ **have** $a \in \text{Field } s$ **unfolding** *Field-def* **by** *auto*
then **have** $(\text{inv-into } (Field r) f b, \text{inv-into } (Field r) f a) \in r$
by (*meson *(1) notin f in-mono inv-into-into oproj-def r wo-rel.in-notinI wo-rel.intro*)
ultimately **have** $(\text{inv-into } (Field r) f b, \text{inv-into } (Field r) f a) \in r$
using *r* **by** (*auto simp: well-order-on-def linear-order-on-def total-on-def*)
with $f[\text{unfolded } \text{oproj-def } \text{compat-def}] *(1) \langle a \in \text{Field } s \rangle$
 $f\text{-inv-into-}f[\text{of } b f \text{Field } r] f\text{-inv-into-}f[\text{of } a f \text{Field } r]$
have $(b, a) \in s$ **by** (*metis in-mono*)
with $*(2,3)$ *s* **have** *False*
by (*auto simp: well-order-on-def linear-order-on-def partial-order-on-def anti-sym-def*)
 $\} \text{thus } (\text{inv-into } (Field r) f a, \text{inv-into } (Field r) f b) \in r$ **by** *blast*
qed

corollary *oproj-ordLeq*:

assumes *r*: *Well-order r* **and** *s*: *Well-order s* **and** *f*: *oproj r s f*
shows $s \leq_o r$
using *f* *oproj-embed* *ordLess-iff ordLess-or-ordLeq r s* **by** *blast*

end

9 Ordinal Arithmetic

theory *Ordinal-Arithmetic*
imports *Wellorder-Constructions*
begin

definition *osum* :: '*a* rel \Rightarrow '*b* rel \Rightarrow ('*a* + '*b*) rel (**infixr** <+o> 70)
where

$r + o r' = \text{map-prod } \text{Inl } \text{Inl } 'r \cup \text{map-prod } \text{Inr } \text{Inr } 'r' \cup$
 $\{(\text{Inl } a, \text{Inr } a') \mid a a' . a \in \text{Field } r \wedge a' \in \text{Field } r'\}$

lemma *Field-osum*: $\text{Field}(r + o r') = \text{Inl } ' \text{Field } r \cup \text{Inr } ' \text{Field } r'$
unfolding *osum-def* *Field-def* **by** *auto*

lemma *osum-Refl*: $\llbracket \text{Refl } r; \text{Refl } r' \rrbracket \Longrightarrow \text{Refl } (r + o r')$

unfolding *refl-on-def* *Field-osum* **unfolding** *osum-def* **by** *blast*

lemma *osum-trans*:

assumes *TRANS*: *trans* *r* **and** *TRANS'*: *trans* *r'*

shows *trans* (*r* + o *r'*)

unfolding *trans-def*

proof(*safe*)

fix *x y z* **assume** *: (*x*, *y*) \in *r* + o *r'* (*y*, *z*) \in *r* + o *r'*

thus (*x*, *z*) \in *r* + o *r'*

proof (*cases x y z rule: sum.exhaust[case-product sum.exhaust sum.exhaust]*)

case (*Inl-Inl-Inl a b c*)

with * **have** (*a*,*b*) \in *r* (*b*,*c*) \in *r* **unfolding** *osum-def* **by** *auto*

with *TRANS* **have** (*a*,*c*) \in *r* **unfolding** *trans-def* **by** *blast*

with *Inl-Inl-Inl* **show** ?*thesis* **unfolding** *osum-def* **by** *auto*

next

case (*Inl-Inl-Inr a b c*)

with * **have** *a* \in *Field* *r* *c* \in *Field* *r'* **unfolding** *osum-def* *Field-def* **by** *auto*

with *Inl-Inl-Inr* **show** ?*thesis* **unfolding** *osum-def* **by** *auto*

next

case (*Inl-Inr-Inr a b c*)

with * **have** *a* \in *Field* *r* *c* \in *Field* *r'* **unfolding** *osum-def* *Field-def* **by** *auto*

with *Inl-Inr-Inr* **show** ?*thesis* **unfolding** *osum-def* **by** *auto*

next

case (*Inr-Inr-Inr a b c*)

with * **have** (*a*,*b*) \in *r'* (*b*,*c*) \in *r'* **unfolding** *osum-def* **by** *auto*

with *TRANS'* **have** (*a*,*c*) \in *r'* **unfolding** *trans-def* **by** *blast*

with *Inr-Inr-Inr* **show** ?*thesis* **unfolding** *osum-def* **by** *auto*

qed (*auto simp: osum-def*)

qed

lemma *osum-Preorder*: $\llbracket \text{Preorder } r; \text{Preorder } r' \rrbracket \implies \text{Preorder } (r +_o r')$
unfolding *preorder-on-def* **using** *osum-Refl* *osum-trans* **by** *blast*

lemma *osum-antisym*: $\llbracket \text{antisym } r; \text{antisym } r' \rrbracket \implies \text{antisym } (r +_o r')$
unfolding *antisym-def* *osum-def* **by** *auto*

lemma *osum-Partial-order*: $\llbracket \text{Partial-order } r; \text{Partial-order } r' \rrbracket \implies \text{Partial-order } (r +_o r')$
unfolding *partial-order-on-def* **using** *osum-Preorder* *osum-antisym* **by** *blast*

lemma *osum-Total*: $\llbracket \text{Total } r; \text{Total } r' \rrbracket \implies \text{Total } (r +_o r')$
unfolding *total-on-def* *Field-osum* **unfolding** *osum-def* **by** *blast*

lemma *osum-Linear-order*: $\llbracket \text{Linear-order } r; \text{Linear-order } r' \rrbracket \implies \text{Linear-order } (r +_o r')$
unfolding *linear-order-on-def* **using** *osum-Partial-order* *osum-Total* **by** *blast*

lemma *osum-wf*:
assumes *WF*: *wf* *r* **and** *WF'*: *wf* *r'*
shows *wf* (*r* +_o *r'*)
unfolding *wf-eq-minimal2* **unfolding** *Field-osum*
proof(*intro allI impI, elim conjE*)
fix *A* **assume** *: $A \subseteq \text{Inl } ' \text{Field } r \cup \text{Inr } ' \text{Field } r' \text{ and } **: A \neq \{\}$
obtain *B* **where** *B-def*: $B = A \text{ Int } \text{Inl } ' \text{Field } r$ **by** *blast*
show $\exists a \in A. \forall a' \in A. (a', a) \notin r +_o r'$
proof(*cases* $B = \{\}$)
case *False*
hence $B \neq \{\}$ $B \leq \text{Inl } ' \text{Field } r$ **using** *B-def* **by** *auto*
hence $\text{Inl } -' B \neq \{\}$ $\text{Inl } -' B \leq \text{Field } r$ **unfolding** *vimage-def* **by** *auto*
then obtain *a* **where** $1: a \in \text{Inl } -' B$ **and** $\forall a1 \in \text{Inl } -' B. (a1, a) \notin r$
using *WF* **unfolding** *wf-eq-minimal2* **by** *metis*
hence $\forall a1 \in A. (a1, \text{Inl } a) \notin r +_o r'$
unfolding *osum-def* **using** *B-def* ** **by** (*auto simp: vimage-def Field-def*)
thus *?thesis* **using** 1 **unfolding** *B-def* **by** *auto*
next
case *True*
hence $1: A \leq \text{Inr } ' \text{Field } r'$ **using** * *B-def* **by** *auto*
with ** **have** $\text{Inr } -' A \neq \{\}$ $\text{Inr } -' A \leq \text{Field } r'$ **unfolding** *vimage-def* **by** *auto*
with ** **obtain** *a'* **where** $2: a' \in \text{Inr } -' A$ **and** $\forall a1' \in \text{Inr } -' A. (a1', a') \notin r'$
using *WF'* **unfolding** *wf-eq-minimal2* **by** *metis*
hence $\forall a1' \in A. (a1', \text{Inr } a') \notin r +_o r'$
unfolding *osum-def* **using** ** 1 **by** (*auto simp: vimage-def Field-def*)
thus *?thesis* **using** 2 **by** *blast*
qed
qed

lemma *osum-minus-Id*:

```

assumes  $r$ : Total  $r \multimap (r \leq Id)$  and  $r'$ : Total  $r' \multimap (r' \leq Id)$ 
shows  $(r + o\ r') - Id \leq (r - Id) + o\ (r' - Id)$ 
unfolding osum-def Total-Id-Field[OF  $r$ ] Total-Id-Field[OF  $r'$ ] by auto

lemma osum-minus-Id1:
   $r \leq Id \implies (r + o\ r') - Id \leq (Inl\ '\ Field\ r \times Inr\ '\ Field\ r') \cup (map\text{-}prod\ Inr\ Inr\ '\ (r' - Id))$ 
unfolding osum-def by auto

lemma osum-minus-Id2:
   $r' \leq Id \implies (r + o\ r') - Id \leq (map\text{-}prod\ Inl\ Inl\ '\ (r - Id)) \cup (Inl\ '\ Field\ r \times Inr\ '\ Field\ r')$ 
unfolding osum-def by auto

lemma osum-wf-Id:
  assumes TOT: Total  $r$  and TOT': Total  $r'$  and WF: wf( $r - Id$ ) and WF': wf( $r' - Id$ )
  shows wf  $((r + o\ r') - Id)$ 
proof(cases  $r \leq Id \vee r' \leq Id$ )
  case False
  thus ?thesis
  using osum-minus-Id[of  $r\ r'$ ] assms osum-wf[of  $r - Id\ r' - Id$ ]
  wf-subset[of  $(r - Id) + o\ (r' - Id)\ (r + o\ r') - Id$ ] by auto
next
  have  $1$ : wf  $(Inl\ '\ Field\ r \times Inr\ '\ Field\ r')$  by (rule wf-Int-Times) auto
  case True
  thus ?thesis
proof (elim disjE)
  assume  $r \subseteq Id$ 
  thus wf  $((r + o\ r') - Id)$ 
  by (rule wf-subset[rotated, OF osum-minus-Id1 wf-Un[OF  $1\ wf\text{-}map\text{-}prod\text{-}image$ [OF WF']]]]) auto
next
  assume  $r' \subseteq Id$ 
  thus wf  $((r + o\ r') - Id)$ 
  by (rule wf-subset[rotated, OF osum-minus-Id2 wf-Un[OF wf-map-prod-image[OF WF]  $1$ ]]]]) auto
qed
qed

lemma osum-Well-order:
  assumes WELL: Well-order  $r$  and WELL': Well-order  $r'$ 
  shows Well-order  $(r + o\ r')$ 
  by (meson WELL WELL' osum-Linear-order osum-wf-Id well-order-on-def wo-rel.TOTAL wo-rel.intro)

lemma osum-embedL:
  assumes WELL: Well-order  $r$  and WELL': Well-order  $r'$ 
  shows embed  $r\ (r + o\ r')\ Inl$ 

```

proof –
have 1 : *Well-order* $(r +_o r')$ **using** *assms* **by** $(\text{auto simp add: osum-Well-order})$
moreover
have *compat* r $(r +_o r')$ *Inl* **unfolding** *compat-def osum-def* **by** *auto*
moreover
have *ofilter* $(r +_o r')$ $(\text{Inl } \text{'Field } r)$
unfolding *wo-rel.ofilter-def* $[\text{unfolded wo-rel-def, OF } 1]$ *Field-osum under-def*
unfolding *osum-def Field-def* **by** *auto*
ultimately show *?thesis* **using** *assms* **by** $(\text{auto simp add: embed-iff-compat-inj-on-ofilter})$
qed

corollary *osum-ordLeqL*:
assumes *WELL*: *Well-order* r **and** *WELL'*: *Well-order* r'
shows $r \leq_o r +_o r'$
using *assms* *osum-embedL osum-Well-order* **unfolding** *ordLeq-def* **by** *blast*

lemma *dir-image-alt*: *dir-image* r $f = \text{map-prod } f$ f $\text{' } r$
unfolding *dir-image-def map-prod-def* **by** *auto*

lemma *map-prod-ordIso*: $\llbracket \text{Well-order } r; \text{inj-on } f \text{ (Field } r) \rrbracket \implies \text{map-prod } f$ f $\text{' } r$
 $=_o r$
by $(\text{metis dir-image-alt dir-image-ordIso ordIso-symmetric})$

definition *oprod* :: $\text{'a rel} \Rightarrow \text{'b rel} \Rightarrow (\text{'a} \times \text{'b}) \text{ rel}$ (**infixr** $\langle *_o \rangle$ 80)
where $r *_o r' = \{((x1, y1), (x2, y2)).$
 $((y1, y2) \in r' - \text{Id} \wedge x1 \in \text{Field } r \wedge x2 \in \text{Field } r) \vee$
 $((y1, y2) \in \text{Restr Id (Field } r') \wedge (x1, x2) \in r)\}$

lemma *Field-oprod*: *Field* $(r *_o r') = \text{Field } r \times \text{Field } r'$
unfolding *oprod-def Field-def* **by** *auto blast+*

lemma *oprod-Refl*: $\llbracket \text{Refl } r; \text{Refl } r' \rrbracket \implies \text{Refl } (r *_o r')$
unfolding *refl-on-def Field-oprod* **unfolding** *oprod-def* **by** *auto*

lemma *oprod-trans*:
assumes *trans* r *trans* r' *antisym* r *antisym* r'
shows *trans* $(r *_o r')$
using *assms* **by** $(\text{clarsimp simp: trans-def antisym-def oprod-def})$ $(\text{metis FieldI1 FieldI2})$

lemma *oprod-Preorder*: $\llbracket \text{Preorder } r; \text{Preorder } r'; \text{antisym } r; \text{antisym } r' \rrbracket \implies \text{Preorder } (r *_o r')$
unfolding *preorder-on-def* **using** *oprod-Refl oprod-trans* **by** *blast*

lemma *oprod-antisym*: $\llbracket \text{antisym } r; \text{antisym } r' \rrbracket \implies \text{antisym } (r *_o r')$
unfolding *antisym-def oprod-def* **by** *auto*

lemma *oprod-Partial-order*: $\llbracket \text{Partial-order } r; \text{Partial-order } r' \rrbracket \implies \text{Partial-order } (r *_o r')$

unfolding *partial-order-on-def* **using** *oprod-Preorder* *oprod-antisym* **by** *blast*

lemma *oprod-Total*: $\llbracket \text{Total } r; \text{Total } r' \rrbracket \implies \text{Total } (r *o r')$
unfolding *total-on-def* *Field-oprod* **unfolding** *oprod-def* **by** *auto*

lemma *oprod-Linear-order*: $\llbracket \text{Linear-order } r; \text{Linear-order } r' \rrbracket \implies \text{Linear-order } (r *o r')$
unfolding *linear-order-on-def* **using** *oprod-Partial-order* *oprod-Total* **by** *blast*

lemma *oprod-wf*:
assumes *WF*: *wf* *r* **and** *WF'*: *wf* *r'*
shows *wf* (*r* *o *r'*)
unfolding *wf-eq-minimal2* **unfolding** *Field-oprod*
proof(*intro allI impI, elim conjE*)
fix *A* **assume** *: $A \subseteq \text{Field } r \times \text{Field } r'$ **and** **: $A \neq \{\}$
then obtain *y* **where** *y*: $y \in \text{snd } A \ \forall y' \in \text{snd } A. (y', y) \notin r'$
using *spec*[*OF* *WF'*[*unfolded wf-eq-minimal2*], *of* *snd* ' *A*] **by** *auto*
let ?*A* = *fst* ' $A \cap \{x. (x, y) \in A\}$
from * *y* **have** ?*A* $\neq \{\}$?*A* $\subseteq \text{Field } r$ **by** *auto*
then obtain *x* **where** *x*: $x \in ?A$ **and** $\forall x' \in ?A. (x', x) \notin r$
using *spec*[*OF* *WF*[*unfolded wf-eq-minimal2*], *of* ?*A*] **by** *auto*
with *y* **have** $\forall a' \in A. (a', (x, y)) \notin r *o r'$
unfolding *oprod-def* *mem-Collect-eq* *split-beta* *fst-conv* *snd-conv* *Id-def* **by** *auto*
moreover from *x* **have** $(x, y) \in A$ **by** *auto*
ultimately show $\exists a \in A. \forall a' \in A. (a', a) \notin r *o r'$ **by** *blast*
qed

lemma *oprod-minus-Id*:
assumes *r*: $\text{Total } r \neg (r \leq \text{Id})$ **and** *r'*: $\text{Total } r' \neg (r' \leq \text{Id})$
shows $(r *o r') - \text{Id} \leq (r - \text{Id}) *o (r' - \text{Id})$
unfolding *oprod-def* *Total-Id-Field*[*OF* *r*] *Total-Id-Field*[*OF* *r'*] **by** *auto*

lemma *oprod-minus-Id1*:
 $r \leq \text{Id} \implies r *o r' - \text{Id} \leq \{(x, y1), (x, y2)). x \in \text{Field } r \wedge (y1, y2) \in (r' - \text{Id})\}$
unfolding *oprod-def* **by** *auto*

lemma *wf-extend-oprod1*:
assumes *wf* *r*
shows *wf* $\{(x, y1), (x, y2)) . x \in A \wedge (y1, y2) \in r\}$
proof (*unfold wf-eq-minimal2, intro allI impI, elim conjE*)
fix *B*
assume *: $B \subseteq \text{Field } \{(x, y1), (x, y2)) . x \in A \wedge (y1, y2) \in r\}$ **and** $B \neq \{\}$
from *image-mono*[*OF* *, *of* *snd*] **have** *snd* ' $B \subseteq \text{Field } r$ **unfolding** *Field-def*
by *force*
with $\langle B \neq \{\} \rangle$ **obtain** *x* **where** *x*: $x \in \text{snd } B \ \forall x' \in \text{snd } B. (x', x) \notin r$
using *spec*[*OF* *assms*[*unfolded wf-eq-minimal2*], *of* *snd* ' *B*] **by** *auto*
then obtain *a* **where** $(a, x) \in B$ **by** *auto*
moreover
from * *x* **have** $\forall a' \in B. (a', (a, x)) \notin \{(x, y1), (x, y2)) . x \in A \wedge (y1, y2) \in r\}$

by *auto*
 ultimately show $\exists ax \in B. \forall a' \in B. (a', ax) \notin \{(x, y1), (x, y2)\} . x \in A \wedge (y1, y2) \in r\}$ by *blast*
 qed

lemma *oprod-minus-Id2*:
 $r' \leq Id \implies r * o r' - Id \leq \{(x1, y), (x2, y)\}. (x1, x2) \in (r - Id) \wedge y \in Field\}$
 unfolding *oprod-def* by *auto*

lemma *wf-extend-oprod2*:
 assumes *wf* *r*
 shows *wf* $\{(x1, y), (x2, y)\} . (x1, x2) \in r \wedge y \in A\}$
 proof (unfold *wf-eq-minimal2*, intro *allI impI, elim conjE*)
 fix *B*
 assume *: $B \subseteq Field \{(x1, y), (x2, y)\}. (x1, x2) \in r \wedge y \in A\}$ and $B \neq \{\}$
 from *image-mono*[*OF* *, *of fst*] have *fst* ' $B \subseteq Field\ r$ unfolding *Field-def* by *force*
 with $\langle B \neq \{\} \rangle$ obtain *x* where *x*: $x \in fst\ 'B \wedge \forall x' \in fst\ 'B. (x', x) \notin r$
 using *spec*[*OF* *assms*[*unfolded wf-eq-minimal2*], *of fst* '*B*] by *auto*
 then obtain *a* where $(x, a) \in B$ by *auto*
 moreover
 from * *x* have $\forall a' \in B. (a', (x, a)) \notin \{(x1, y), (x2, y)\}. (x1, x2) \in r \wedge y \in A\}$
 by *auto*
 ultimately show $\exists xa \in B. \forall a' \in B. (a', xa) \notin \{(x1, y), (x2, y)\}. (x1, x2) \in r \wedge y \in A\}$ by *blast*
 qed

lemma *oprod-wf-Id*:
 assumes *TOT*: *Total* *r* and *TOT'*: *Total* *r'* and *WF*: *wf*(*r* - *Id*) and *WF'*:
wf(*r'* - *Id*)
 shows *wf* $((r * o r') - Id)$
 proof (cases $r \leq Id \vee r' \leq Id$)
 case *False*
 thus ?thesis
 by (meson *TOT TOT' WF WF' oprod-minus-Id oprod-wf wf-subset*)
 next
 case *True*
 thus ?thesis using *wf-subset*[*OF* *wf-extend-oprod1*[*OF* *WF*] *oprod-minus-Id1*]
wf-subset[*OF* *wf-extend-oprod2*[*OF* *WF*] *oprod-minus-Id2*] by *auto*
 qed

lemma *oprod-Well-order*:
 assumes *WELL*: *Well-order* *r* and *WELL'*: *Well-order* *r'*
 shows *Well-order* $(r * o r')$
 by (meson *WELL WELL' linear-order-on-def oprod-Linear-order oprod-wf-Id well-order-on-def*)

lemma *oprod-embed*:

assumes *WELL*: Well-order r **and** *WELL'*: Well-order r' **and** $r' \neq \{\}$
shows $\text{embed } r (r * o r') (\lambda x. (x, \text{minim } r' (\text{Field } r')))$ **(is embed - - ?f)**
proof –
from *assms*(β) **have** $r': \text{Field } r' \neq \{\}$ **unfolding** *Field-def* **by** *auto*
have *minim*[*simp*]: $\text{minim } r' (\text{Field } r') \in \text{Field } r'$
using *wo-rel.minim-inField*[*unfolded wo-rel-def, OF WELL' - r'*] **by** *auto*
{ fix b
assume $b: (b, \text{minim } r' (\text{Field } r')) \in r'$
hence $b \in \text{Field } r'$ **unfolding** *Field-def* **by** *auto*
hence $(\text{minim } r' (\text{Field } r'), b) \in r'$
using *wo-rel.minim-least*[*unfolded wo-rel-def, OF WELL' subset-refl*] r' **by**
auto
with b **have** $b = \text{minim } r' (\text{Field } r')$
by (*metis WELL' antisym-def linear-order-on-def partial-order-on-def well-order-on-def*)
} **note** $*$ = *this*
have $1: \text{Well-order } (r * o r')$ **using** *assms* **by** (*auto simp add: oprod-Well-order*)
moreover
from r' **have** $\text{compat } r (r * o r') \text{ ?f}$ **unfolding** *compat-def oprod-def* **by** *auto*
moreover
from $*$ **have** $\text{ofilter } (r * o r') (\text{?f } ' \text{Field } r)$
unfolding *wo-rel.ofilter-def*[*unfolded wo-rel-def, OF 1*] *Field-oprod under-def*
unfolding *oprod-def* **by** *auto* (*auto simp: image-iff Field-def*)
moreover **have** $\text{inj-on } \text{?f } (\text{Field } r)$ **unfolding** *inj-on-def* **by** *auto*
ultimately show *?thesis* **using** *assms* **by** (*auto simp add: embed-iff-compat-inj-on-ofilter*)
qed

corollary *oprod-ordLeq*: $\llbracket \text{Well-order } r; \text{Well-order } r'; r' \neq \{\} \rrbracket \implies r \leq_o r * o r'$
using *oprod-embed oprod-Well-order* **unfolding** *ordLeq-def* **by** *blast*

definition *support* $z A f = \{x \in A. f x \neq z\}$

lemma *support-Un*[*simp*]: $\text{support } z (A \cup B) f = \text{support } z A f \cup \text{support } z B f$
unfolding *support-def* **by** *auto*

lemma *support-upd*[*simp*]: $\text{support } z A (f(x := z)) = \text{support } z A f - \{x\}$
unfolding *support-def* **by** *auto*

lemma *support-upd-subset*[*simp*]: $\text{support } z A (f(x := y)) \subseteq \text{support } z A f \cup \{x\}$
unfolding *support-def* **by** *auto*

lemma *fun-unequal-in-support*:
assumes $f \neq g \in \text{Func } A B \ g \in \text{Func } A C$
shows $(\text{support } z A f \cup \text{support } z A g) \cap \{a. f a \neq g a\} \neq \{\}$
using *assms* **by** (*simp add: Func-def support-def disjoint-iff fun-eq-iff*) *metis*

definition *fin-support* **where**
 $\text{fin-support } z A = \{f. \text{finite } (\text{support } z A f)\}$

lemma *finite-support*: $f \in \text{fin-support } z A \implies \text{finite } (\text{support } z A f)$

```

unfolding support-def fin-support-def by auto

lemma fin-support-Field-osum:
   $f \in \text{fin-support } z \text{ (Inl } \cdot A \cup \text{Inr } \cdot B) \iff$ 
   $(f \circ \text{Inl}) \in \text{fin-support } z A \wedge (f \circ \text{Inr}) \in \text{fin-support } z B$  (is ?L  $\iff$  ?R1  $\wedge$  ?R2)
proof safe
  assume ?L
  from <?L> show ?R1 unfolding fin-support-def support-def
    by (fastforce simp: image-iff elim: finite-surj[of - - case-sum id undefined])
  from <?L> show ?R2 unfolding fin-support-def support-def
    by (fastforce simp: image-iff elim: finite-surj[of - - case-sum undefined id])
next
  assume ?R1 ?R2
  thus ?L unfolding fin-support-def support-Un
    by (auto simp: support-def elim: finite-surj[of - - Inl] finite-surj[of - - Inr])
qed

lemma Func-upd:  $\llbracket f \in \text{Func } A B; x \in A; y \in B \rrbracket \implies f(x := y) \in \text{Func } A B$ 
  unfolding Func-def by auto

context wo-rel
begin

definition isMaxim :: 'a set  $\Rightarrow$  'a  $\Rightarrow$  bool
  where isMaxim A b  $\equiv b \in A \wedge (\forall a \in A. (a, b) \in r)$ 

definition maxim :: 'a set  $\Rightarrow$  'a
  where maxim A  $\equiv \text{THE } b. \text{isMaxim } A b$ 

lemma isMaxim-unique[intro]:  $\llbracket \text{isMaxim } A x; \text{isMaxim } A y \rrbracket \implies x = y$ 
  unfolding isMaxim-def using antisymD[OF ANTISYM, of x y] by auto

lemma maxim-isMaxim:  $\llbracket \text{finite } A; A \neq \{\}; A \subseteq \text{Field } r \rrbracket \implies \text{isMaxim } A (\text{maxim } A)$ 
  unfolding maxim-def
proof (rule theI', rule ex-ex1I[OF isMaxim-unique, rotated], assumption+,
  induct A rule: finite-induct)
  case (insert x A)
  thus ?case
  proof (cases A = {})
    case True
    moreover have isMaxim {x} x unfolding isMaxim-def using refl-onD[OF
  REFL] insert(5) by auto
    ultimately show ?thesis by blast
  next
  case False
  with insert(3,5) obtain y where isMaxim A y by blast
  with insert(2,5) have if (y, x)  $\in r$  then isMaxim (insert x A) x else isMaxim
  (insert x A) y

```


unfolding *isMaxim-def subset-eq* **by** (*metis insert-iff max2-def max2-equals1 max2-iff*)
thus *?thesis* **by** *metis*
qed
qed *simp*

lemma *maxim-in*: $\llbracket \text{finite } A; A \neq \{\}; A \subseteq \text{Field } r \rrbracket \implies \text{maxim } A \in A$
using *maxim-isMaxim* **unfolding** *isMaxim-def* **by** *auto*

lemma *maxim-greatest*: $\llbracket \text{finite } A; x \in A; A \subseteq \text{Field } r \rrbracket \implies (x, \text{maxim } A) \in r$
using *maxim-isMaxim* **unfolding** *isMaxim-def* **by** *auto*

lemma *isMaxim-zero*: $\text{isMaxim } A \text{ zero} \implies A = \{\text{zero}\}$
unfolding *isMaxim-def* **by** *auto*

lemma *maxim-insert*:
assumes *finite A A* $\neq \{\}$ $A \subseteq \text{Field } r$ $x \in \text{Field } r$
shows $\text{maxim } (\text{insert } x A) = \text{max2 } x (\text{maxim } A)$
proof –
from *assms* **have** $*$: $\text{isMaxim } (\text{insert } x A) (\text{maxim } (\text{insert } x A)) \text{ isMaxim } A (\text{maxim } A)$
using *maxim-isMaxim* **by** *auto*
show *?thesis*
proof (*cases* $(x, \text{maxim } A) \in r$)
case *True*
with $*(2)$ **have** $\text{isMaxim } (\text{insert } x A) (\text{maxim } A)$
by (*simp add: isMaxim-def*)
with $*(1)$ *True* **show** *?thesis*
unfolding *max2-def* **by** (*metis isMaxim-unique*)
next
case *False*
hence $(\text{maxim } A, x) \in r$ **by** (*metis* $*(2)$ *assms(3,4)* *in-mono in-notinI isMaxim-def*)
with $*(2)$ *assms(4)* **have** $\text{isMaxim } (\text{insert } x A) x$ **unfolding** *isMaxim-def*
using *transD[OF TRANS, of - maxim A x]* *refl-onD[OF REFL, of x]* **by** *blast*
with $*(1)$ *False* **show** *?thesis* **unfolding** *max2-def* **by** (*metis isMaxim-unique*)
qed
qed

lemma *maxim-Un*:
assumes *finite A A* $\neq \{\}$ $A \subseteq \text{Field } r$ *finite B B* $\neq \{\}$ $B \subseteq \text{Field } r$
shows $\text{maxim } (A \cup B) = \text{max2 } (\text{maxim } A) (\text{maxim } B)$
proof –
from *assms* **have** $*$: $\text{isMaxim } (A \cup B) (\text{maxim } (A \cup B)) \text{ isMaxim } A (\text{maxim } A)$
 $\text{isMaxim } B (\text{maxim } B)$
using *maxim-isMaxim* **by** *auto*
show *?thesis*
proof (*cases* $(\text{maxim } A, \text{maxim } B) \in r$)
case *True*

with $*(2,3)$ **have** $isMaxim (A \cup B) (maxim B)$ **unfolding** $isMaxim-def$
using $transD[OF TRANS, of - maxim A maxim B]$ **by** $blast$
with $*(1)$ **True show** $?thesis$ **unfolding** $max2-def$ **by** $(metis isMaxim-unique)$
next
case $False$
hence $(maxim B, maxim A) \in r$ **by** $(metis *(2,3) assms(3,6) in-mono in-notinI isMaxim-def)$
with $*(2,3)$ **have** $isMaxim (A \cup B) (maxim A)$
by $(metis *(1) False Un-iff isMaxim-def isMaxim-unique)$
with $*(1)$ $False$ **show** $?thesis$ **unfolding** $max2-def$ **by** $(metis isMaxim-unique)$
qed
qed

lemma $maxim-insert-zero$:
assumes $finite A A \neq \{\}$ $A \subseteq Field r$
shows $maxim (insert zero A) = maxim A$
using $assms finite.cases in-mono max2-def maxim-in maxim-insert subset-empty zero-in-Field zero-smallest$ **by** $fastforce$

lemma $maxim-equality$: $isMaxim A x \implies maxim A = x$
unfolding $maxim-def$ **by** $(rule the-equality) auto$

lemma $maxim-singleton$:
 $x \in Field r \implies maxim \{x\} = x$
using $refl-onD[OF REFL]$ **by** $(intro maxim-equality) (simp add: isMaxim-def)$

lemma $maxim-Int$: $\llbracket finite A; A \neq \{\}; A \subseteq Field r; maxim A \in B \rrbracket \implies maxim (A \cap B) = maxim A$
by $(rule maxim-equality) (auto simp: isMaxim-def intro: maxim-in maxim-greatest)$

lemma $maxim-mono$: $\llbracket X \subseteq Y; finite Y; X \neq \{\}; Y \subseteq Field r \rrbracket \implies (maxim X, maxim Y) \in r$
using $maxim-in[OF finite-subset, of X Y]$ **by** $(auto intro: maxim-greatest)$

definition $max-fun-diff f g \equiv maxim (\{a \in Field r. f a \neq g a\})$

lemma $max-fun-diff-commute$: $max-fun-diff f g = max-fun-diff g f$
unfolding $max-fun-diff-def$ **by** $metis$

lemma $zero-under$: $x \in Field r \implies zero \in under x$
unfolding $under-def$ **by** $(auto intro: zero-smallest)$

end

definition $FinFunc r s = Func (Field s) (Field r) \cap fin-support (zero r) (Field s)$

lemma $FinFuncD$: $\llbracket f \in FinFunc r s; x \in Field s \rrbracket \implies f x \in Field r$
unfolding $FinFunc-def Func-def$ **by** $(fastforce split: option.splits)$

locale *wo-rel2* =
 fixes *r s*
 assumes *rWELL*: *Well-order r*
 and *sWELL*: *Well-order s*
begin

interpretation *r*: *wo-rel r* **by** *unfold-locales (rule rWELL)*
interpretation *s*: *wo-rel s* **by** *unfold-locales (rule sWELL)*

abbreviation *SUPP* \equiv *support r.zero (Field s)*
abbreviation *FINFUNC* \equiv *FinFunc r s*
lemmas *FINFUNCD* = *FinFuncD[of - r s]*

lemma *fun-diff-alt*: $\{a \in \text{Field } s. f\ a \neq g\ a\} = (SUPP\ f \cup SUPP\ g) \cap \{a. f\ a \neq g\ a\}$
by (*auto simp: support-def*)

lemma *max-fun-diff-alt*:
s.max-fun-diff f g = *s.maxim ((SUPP f \cup SUPP g) \cap {*a. f a \neq g a*})*
unfolding *s.max-fun-diff-def fun-diff-alt ..*

lemma *isMaxim-max-fun-diff*: $\llbracket f \neq g; f \in \text{FINFUNC}; g \in \text{FINFUNC} \rrbracket \implies$
*s.isMaxim {*a* \in Field *s. f a \neq g a*}* (*s.max-fun-diff f g*)
using *fun-unequal-in-support[of f g]* **unfolding** *max-fun-diff-alt fun-diff-alt fun-eq-iff*
by (*intro s.maxim-isMaxim (auto simp: FinFunc-def fin-support-def support-def)*)

lemma *max-fun-diff-in*: $\llbracket f \neq g; f \in \text{FINFUNC}; g \in \text{FINFUNC} \rrbracket \implies$
s.max-fun-diff f g \in {*a* \in Field *s. f a \neq g a*}
using *isMaxim-max-fun-diff* **unfolding** *s.isMaxim-def* **by** *blast*

lemma *max-fun-diff-max*: $\llbracket f \neq g; f \in \text{FINFUNC}; g \in \text{FINFUNC}; x \in \{a \in \text{Field } s. f\ a \neq g\ a\} \rrbracket \implies$
(*x, s.max-fun-diff f g*) \in *s*
using *isMaxim-max-fun-diff* **unfolding** *s.isMaxim-def* **by** *blast*

lemma *max-fun-diff*:
 $\llbracket f \neq g; f \in \text{FINFUNC}; g \in \text{FINFUNC} \rrbracket \implies$
($\exists a\ b. a \neq b \wedge a \in \text{Field } r \wedge b \in \text{Field } r \wedge$
 $f\ (s.\text{max-fun-diff } f\ g) = a \wedge g\ (s.\text{max-fun-diff } f\ g) = b$)
using *isMaxim-max-fun-diff[of f g]* **unfolding** *s.isMaxim-def FinFunc-def Func-def*
by *auto*

lemma *max-fun-diff-le-eq*:
 $\llbracket (s.\text{max-fun-diff } f\ g, x) \in s; f \neq g; f \in \text{FINFUNC}; g \in \text{FINFUNC}; x \neq s.\text{max-fun-diff } f\ g \rrbracket \implies$
 $f\ x = g\ x$
using *max-fun-diff-max[of f g x]* *antisymD[OF s.ANTISYM, of s.max-fun-diff f g x]*
by (*auto simp: Field-def*)

lemma *max-fun-diff-max2*:
assumes *ineq*: $s.\text{max-fun-diff } f \ g = s.\text{max-fun-diff } g \ h \longrightarrow$
 $f \ (s.\text{max-fun-diff } f \ g) \neq h \ (s.\text{max-fun-diff } g \ h)$ **and**
 $fg: f \neq g$ **and** $gh: g \neq h$ **and** $fh: f \neq h$ **and**
 $f: f \in \text{FINFUNC}$ **and** $g: g \in \text{FINFUNC}$ **and** $h: h \in \text{FINFUNC}$
shows $s.\text{max-fun-diff } f \ h = s.\text{max2} \ (s.\text{max-fun-diff } f \ g) \ (s.\text{max-fun-diff } g \ h)$
(is $?fh = s.\text{max2} \ ?fg \ ?gh)$
proof (*cases* $?fg = ?gh$)
case *True*
with *ineq* **have** $f \ ?fg \neq h \ ?fg$ **by** *simp*
moreover
{ **fix** x **assume** $x: x \in \{a \in \text{Field } s. f \ a \neq h \ a\}$
hence $(x, ?fg) \in s$
proof (*cases* $x = ?fg$)
case *False* **show** *?thesis*
by (*metis* (*mono-tags*, *lifting*) *True* *assms*(5–7) *max-fun-diff-max mem-Collect-eq*
x)
qed (*simp* *add*: *reft-onD*[*OF* *s.REFL*])
}
ultimately **have** $s.\text{isMaxim } \{a \in \text{Field } s. f \ a \neq h \ a\} \ ?fg$
unfolding *s.isMaxim-def* **using** *max-fun-diff-in*[*OF* *fg f g*] **by** *simp*
hence $?fh = ?fg$ **using** *isMaxim-max-fun-diff*[*OF* *fh f h*] **by** *blast*
thus *?thesis* **unfolding** *True s.max2-def* **by** *simp*
next
case *False* **note** $* = \text{this}$
show *?thesis*
proof (*cases* $(?fg, ?gh) \in s$)
case *True*
hence $*: f \ ?gh = g \ ?gh$ **by** (*rule* *max-fun-diff-le-eq*[*OF* - *fg f g* **[symmetric]*])
hence $s.\text{isMaxim } \{a \in \text{Field } s. f \ a \neq h \ a\} \ ?gh$ **using** *isMaxim-max-fun-diff*[*OF*
 $gh \ g \ h$]
 $\text{isMaxim-max-fun-diff}[OF \ fg \ f \ g] \text{ transD}[OF \ s.TRANS - True]$
unfolding *s.isMaxim-def* **by** *auto*
hence $?fh = ?gh$ **using** *isMaxim-max-fun-diff*[*OF* *fh f h*] **by** *blast*
thus *?thesis* **using** *True* **unfolding** *s.max2-def* **by** *simp*
next
case *False*
with *max-fun-diff-in*[*OF* *fg f g*] *max-fun-diff-in*[*OF* *gh g h*] **have** *True*: $(?gh,$
 $?fg) \in s$
by (*blast* *intro*: *s.in-notinI*)
hence $*: g \ ?fg = h \ ?fg$ **by** (*rule* *max-fun-diff-le-eq*[*OF* - *gh g h* *])
hence $s.\text{isMaxim } \{a \in \text{Field } s. f \ a \neq h \ a\} \ ?fg$ **using** *isMaxim-max-fun-diff*[*OF*
 $gh \ g \ h$]
 $\text{isMaxim-max-fun-diff}[OF \ fg \ f \ g] \text{ True transD}[OF \ s.TRANS, of - - ?fg]$
unfolding *s.isMaxim-def* **by** *auto*
hence $?fh = ?fg$ **using** *isMaxim-max-fun-diff*[*OF* *fh f h*] **by** *blast*
thus *?thesis* **using** *False* **unfolding** *s.max2-def* **by** *simp*
qed

qed

definition *oexp* **where**

$oexp = \{(f, g) \cdot f \in FINFUNC \wedge g \in FINFUNC \wedge$
 $((let\ m = s.max\text{-}fun\text{-}diff\ f\ g\ in\ (f\ m, g\ m) \in r) \vee f = g)\}$

lemma *Field-oexp*: *Field* *oexp* = *FINFUNC*

unfolding *oexp-def* *FinFunc-def* **by** (*auto simp: Let-def Field-def*)

lemma *oexp-Refl*: *Refl* *oexp*

unfolding *refl-on-def Field-oexp* **unfolding** *oexp-def* **by** (*auto simp: Let-def*)

lemma *oexp-trans*: *trans* *oexp*

proof (*unfold trans-def, safe*)

fix *f g h* :: '*b* \Rightarrow '*a*

let *?fg* = *s.max-fun-diff* *f g*

and *?gh* = *s.max-fun-diff* *g h*

and *?fh* = *s.max-fun-diff* *f h*

assume *oexp*: (*f, g*) \in *oexp* (*g, h*) \in *oexp*

thus (*f, h*) \in *oexp*

proof (*cases f = g \vee g = h*)

case *False*

with *oexp* **have** *f* \in *FINFUNC* *g* \in *FINFUNC* *h* \in *FINFUNC*

(*f ?fg, g ?fg*) \in *r* (*g ?gh, h ?gh*) \in *r* **unfolding** *oexp-def Let-def* **by** *auto*

note * = *this False*

show *?thesis*

proof (*cases f \neq h*)

case *True*

show *?thesis*

proof (*cases ?fg = ?gh \longrightarrow f ?fg \neq h ?gh*)

case *True*

show *?thesis* **using** *max-fun-diff-max2*[*of f g h, OF True*] * $\langle f \neq h \rangle$

max-fun-diff-in

r.max2-iff[*OF FINFUNCD FINFUNCD*] *r.max2-equals1*[*OF FINFUNCD*

FINFUNCD] *max-fun-diff-le-eq*

s.in-notinI[*OF disjI1*] **unfolding** *oexp-def Let-def s.max2-def mem-Collect-eq*

by *safe metis*

next

case *False* **with** * **show** *?thesis* **unfolding** *oexp-def Let-def*

using *antisymD*[*OF r.ANTISYM, of g ?gh h ?gh*] *max-fun-diff-in*[*of g h*]

by *auto*

qed

qed (*auto simp: oexp-def *(3)*)

qed *auto*

qed

lemma *oexp-Preorder*: *Preorder* *oexp*

unfolding *preorder-on-def* **using** *oexp-Refl oexp-trans* **by** *blast*

```

lemma oexp-antisym: antisym oexp
proof (unfold antisym-def, safe, rule ccontr)
  fix f g assume  $(f, g) \in \text{oexp}$   $(g, f) \in \text{oexp}$   $g \neq f$ 
  thus False using refl-onD[OF r.REFL FINFUNC] max-fun-diff-in unfolding
oexp-def Let-def
  by (auto dest!: antisymD[OF r.ANTISYM] simp: s.max-fun-diff-commute)
qed

lemma oexp-Partial-order: Partial-order oexp
unfolding partial-order-on-def using oexp-Preorder oexp-antisym by blast

lemma oexp-Total: Total oexp
unfolding total-on-def Field-oexp unfolding oexp-def using FINFUNC max-fun-diff-in
by (auto simp: Let-def s.max-fun-diff-commute intro!: r.in-notinI)

lemma oexp-Linear-order: Linear-order oexp
unfolding linear-order-on-def using oexp-Partial-order oexp-Total by blast

definition const = ( $\lambda x. \text{if } x \in \text{Field } s \text{ then } r.\text{zero} \text{ else undefined}$ )

lemma const-in[simp]:  $x \in \text{Field } s \implies \text{const } x = r.\text{zero}$ 
unfolding const-def by auto

lemma const-notin[simp]:  $x \notin \text{Field } s \implies \text{const } x = \text{undefined}$ 
unfolding const-def by auto

lemma const-Int-Field[simp]:  $\text{Field } s \cap - \{x. \text{const } x = r.\text{zero}\} = \{\}$ 
by auto

lemma const-FINFUNC[simp]:  $\text{Field } r \neq \{\} \implies \text{const} \in \text{FINFUNC}$ 
unfolding FinFunc-def Func-def fin-support-def support-def const-def Int-iff mem-Collect-eq
using r.zero-in-Field by (metis (lifting) Collect-empty-eq finite.emptyI)

lemma const-least:
assumes  $\text{Field } r \neq \{\}$   $f \in \text{FINFUNC}$ 
shows  $(\text{const}, f) \in \text{oexp}$ 
using assms const-FINFUNC max-fun-diff max-fun-diff-in oexp-def by fastforce

lemma support-not-const:
assumes  $F \subseteq \text{FINFUNC}$  and  $\text{const} \notin F$ 
shows  $\forall f \in F. \text{finite } (\text{SUPP } f) \wedge \text{SUPP } f \neq \{\} \wedge \text{SUPP } f \subseteq \text{Field } s$ 
proof (intro ballI conjI)
  fix f assume  $f \in F$ 
  thus finite (SUPP f) SUPP f  $\subseteq \text{Field } s$ 
  using assms(1) unfolding FinFunc-def fin-support-def support-def by auto
show SUPP f  $\neq \{\}$ 
proof (rule ccontr, unfold not-not)
  assume SUPP f =  $\{\}$ 
  moreover from  $\langle f \in F \rangle$  assms(1) have  $f \in \text{FINFUNC}$  by blast

```

```

ultimately have  $f = \text{const}$ 
  by (auto simp: fun-eq-iff support-def FinFunc-def Func-def const-def)
  with assms(2)  $\langle f \in F \rangle$  show False by blast
qed
qed

lemma maxim-isMaxim-support:
  assumes  $F \subseteq \text{FINFUNC}$  and  $\text{const} \notin F$ 
  shows  $\forall f \in F. s.\text{isMaxim} (\text{SUPP } f) (s.\text{maxim} (\text{SUPP } f))$ 
  using assms s.maxim-isMaxim support-not-const by force

lemma oexp-empty2:  $\text{Field } s = \{\} \implies \text{oexp} = \{(\lambda x. \text{undefined}, \lambda x. \text{undefined})\}$ 
  unfolding oexp-def FinFunc-def fin-support-def support-def by auto

lemma oexp-empty:  $\llbracket \text{Field } r = \{\}; \text{Field } s \neq \{\} \rrbracket \implies \text{oexp} = \{\}$ 
  using FINFUNCDef oexp-def by auto

lemma fun-upd-FINFUNC:  $\llbracket f \in \text{FINFUNC}; x \in \text{Field } s; y \in \text{Field } r \rrbracket \implies f(x := y) \in \text{FINFUNC}$ 
  unfolding FinFunc-def Func-def fin-support-def
  by (auto intro: finite-subset[OF support-upd-subset])

lemma fun-upd-same-oexp:
  assumes  $(f, g) \in \text{oexp}$   $f x = g x$   $x \in \text{Field } s$   $y \in \text{Field } r$ 
  shows  $(f(x := y), g(x := y)) \in \text{oexp}$ 
proof -
  from assms(1) fun-upd-FINFUNC[OF - assms(3,4)] have fg:  $f(x := y) \in \text{FINFUNC}$   $g(x := y) \in \text{FINFUNC}$ 
  unfolding oexp-def by auto
  moreover from assms(2) have s.max-fun-diff  $(f(x := y)) (g(x := y)) = s.\text{max-fun-diff } f g$ 
  unfolding s.max-fun-diff-def by auto metis
  ultimately show ?thesis using assms refl-onD[OF r.REFL] unfolding oexp-def
  Let-def by auto
qed

lemma fun-upd-smaller-oexp:
  assumes  $f \in \text{FINFUNC}$   $x \in \text{Field } s$   $y \in \text{Field } r$   $(y, f x) \in r$ 
  shows  $(f(x := y), f) \in \text{oexp}$ 
  using assms fun-upd-FINFUNC[OF assms(1-3)] s.maxim-singleton[of x]
  unfolding oexp-def FinFunc-def Let-def fin-support-def s.max-fun-diff-def by
  (auto simp: fun-eq-iff)

lemma oexp-wf-Id: wf (oexp - Id)
proof (cases  $\text{Field } r = \{\} \vee \text{Field } s = \{\}$ )
  case True thus ?thesis using oexp-empty oexp-empty2 by fastforce
next
  case False
  hence Fields:  $\text{Field } s \neq \{\} \text{Field } r \neq \{\}$  by simp-all

```

```

hence [simp]: r.zero ∈ Field r by (intro r.zero-in-Field)
have const[simp]:  $\bigwedge F. \llbracket \text{const} \in F; F \subseteq \text{FINFUNC} \rrbracket \implies \exists f0 \in F. \forall f \in F. (f0, f)$ 
∈ oexp
  using const-least[OF Fields(2)] by auto
show ?thesis
  unfolding Linear-order-wf-diff-Id[OF oexp-Linear-order] Field-oexp
proof (intro allI impI)
  fix A assume A:  $A \subseteq \text{FINFUNC}$   $A \neq \{\}$ 
  { fix y F
    have  $F \subseteq \text{FINFUNC} \wedge (\exists f \in F. y = s.\text{maxim}(\text{SUPP } f)) \longrightarrow$ 
       $(\exists f0 \in F. \forall f \in F. (f0, f) \in \text{oexp})$  (is ?P F y)
    proof (induct y arbitrary: F rule: s.well-order-induct)
      case (1 y)
      show ?case
      proof (intro impI, elim conjE bexE)
        fix f assume F:  $F \subseteq \text{FINFUNC}$   $f \in F$   $y = s.\text{maxim}(\text{SUPP } f)$ 
        thus  $\exists f0 \in F. \forall f \in F. (f0, f) \in \text{oexp}$ 
        proof (cases const ∈ F)
          case False
          with F have maxF:  $\forall f \in F. s.\text{isMaxim}(\text{SUPP } f) (s.\text{maxim}(\text{SUPP } f))$ 
            and SUPPF:  $\forall f \in F. \text{finite}(\text{SUPP } f) \wedge \text{SUPP } f \neq \{\} \wedge \text{SUPP } f \subseteq$ 
Field s
            using maxim-isMaxim-support support-not-const by auto
            define z where  $z = s.\text{minim} \{s.\text{maxim}(\text{SUPP } f) \mid f. f \in F\}$ 
            from F SUPPF maxF have zmin:  $s.\text{isMinim} \{s.\text{maxim}(\text{SUPP } f) \mid f. f$ 
∈ F} z
            unfolding z-def by (intro s.minim-isMinim) (auto simp: s.isMaxim-def)
            with F have zy:  $(z, y) \in s$  unfolding s.isMinim-def by auto
            hence zField:  $z \in \text{Field } s$  unfolding Field-def by auto
            define x0 where  $x0 = r.\text{minim} \{f z \mid f. f \in F \wedge z = s.\text{maxim}(\text{SUPP}$ 
f)\}
            from F(1,2) maxF(1) SUPPF zmin
            have x0min:  $r.\text{isMinim} \{f z \mid f. f \in F \wedge z = s.\text{maxim}(\text{SUPP } f)\}$  x0
              unfolding x0-def s.isMaxim-def s.isMinim-def
              by (blast intro!: r.minim-isMinim FinFuncD[of - r s])
            with maxF(1) SUPPF F(1) have x0Field:  $x0 \in \text{Field } r$ 
              unfolding r.isMinim-def s.isMaxim-def by (auto intro!: FINFUNC D)
            from x0min maxF(1) SUPPF F(1) have x0notzero:  $x0 \neq r.\text{zero}$ 
            unfolding r.isMinim-def s.isMaxim-def FinFunc-def Func-def support-def
              by fastforce
            define G where  $G = \{f(z := r.\text{zero}) \mid f. f \in F \wedge z = s.\text{maxim}(\text{SUPP}$ 
f)  $\wedge f z = x0\}$ 
            from zmin x0min have  $G \neq \{\}$  unfolding G-def z-def s.isMinim-def
r.isMinim-def by blast
            have GF:  $G \subseteq (\lambda f. f(z := r.\text{zero})) \text{ ` } F$  unfolding G-def by auto
            have  $G \subseteq \text{fin-support } r.\text{zero} (\text{Field } s)$ 
              unfolding FinFunc-def fin-support-def
              using F(1) FinFunc-def G-def fin-support-def by fastforce
            moreover from F GF zField have  $G \subseteq \text{Func}(\text{Field } s) (\text{Field } r)$ 

```


using *Func-upd*[*of - Field s Field r z r.zero*] **unfolding** *FinFunc-def*
 by *auto*
 ultimately have $G: G \subseteq \text{FINFUNC}$ **unfolding** *FinFunc-def* **by** *blast*
 hence $\exists g0 \in G. \forall g \in G. (g0, g) \in \text{oe xp}$
proof (*cases const* $\in G$)
 case *False*
 with G have $\text{max}G: \forall g \in G. s.\text{isMaxim} (\text{SUPP } g) (s.\text{maxim} (\text{SUPP } g))$
 and $\text{SUPPG}: \forall g \in G. \text{finite} (\text{SUPP } g) \wedge \text{SUPP } g \neq \{\} \wedge \text{SUPP } g \subseteq \text{Field } s$
 using *maxim-isMaxim-support support-not-const* **by** *auto*
 define y' where $y' = s.\text{minim} \{s.\text{maxim} (\text{SUPP } f) \mid f. f \in G\}$
 from $G \text{ SUPPG } \text{max}G \langle G \neq \{\} \rangle$ have $y'\text{min}: s.\text{isMinim} \{s.\text{maxim} (\text{SUPP } f) \mid f. f \in G\} y'$
unfolding $y'\text{-def}$ **by** (*intro s.minim-isMinim*) (*auto simp: s.isMaxim-def*)
moreover
 have $\forall g \in G. z \notin \text{SUPP } g$ **unfolding** *support-def G-def* **by** *auto*
moreover
 { **fix** g **assume** $g: g \in G$
 then obtain f where $f \in F$ $g = f(z := r.\text{zero})$ **and** $z: z = s.\text{maxim} (\text{SUPP } f)$
unfolding $G\text{-def}$ **by** *auto*
 with $\text{SUPPF } \text{bspec}[OF \text{SUPPG } g]$ have $(s.\text{maxim} (\text{SUPP } g), z) \in s$
unfolding z **by** (*intro s.maxim-mono*) *auto*
 }
moreover from $y'\text{min}$ have $\bigwedge g. g \in G \implies (y', s.\text{maxim} (\text{SUPP } g)) \in s$
unfolding *s.isMinim-def* **by** *auto*
 ultimately have $y' \neq z$ (y', z) $\in s$ **using** $\text{max}G$
unfolding *s.isMinim-def s.isMaxim-def* **by** *auto*
 with zy have $y' \neq y$ (y', y) $\in s$ **using** *antisymD*[*OF s.ANTISYM*]
transD[*OF s.TRANS*]
by *blast+*
moreover from $\langle G \neq \{\} \rangle$ have $\exists g \in G. y' = \text{wo-rel.maxim } s (\text{SUPP } g)$ **using** $y'\text{min}$
by (*auto simp: G-def s.isMinim-def*)
 ultimately show *?thesis* **using** *mp*[*OF spec* [*OF mp* [*OF spec* [*OF 1*]]],
of y' G] G **by** *auto*
qed simp
 then obtain $g0$ where $g0: g0 \in G \forall g \in G. (g0, g) \in \text{oe xp}$ **by** *blast*
 hence $g0z: g0 z = r.\text{zero}$ **unfolding** $G\text{-def}$ **by** *auto*
 define $f0$ where $f0 = g0(z := x0)$
 with $x0\text{notzero } z\text{Field}$ have $\text{SUPP } f0 = \text{SUPP } g0 \cup \{z\}$ **unfolding** *support-def* **by** *auto*
 from $g0z$ have $f0z: f0(z := r.\text{zero}) = g0$ **unfolding** $f0\text{-def fun-upd-upd}$
by *auto*
 have $f0: f0 \in F$ **using** $x0\text{min } g0(1)$
Func-elim[*OF subsetD* [*OF subset-trans* [*OF F(1)* [*unfolded FinFunc-def*]
Int-lower1]]] $z\text{Field}$

```

      unfolding f0-def r.isMinim-def G-def by (force simp: fun-upd-idem)
      from g0(1) maxF(1) have maxf0: s.maxim (SUPP f0) = z unfolding
SUPP G-def
      by (intro s.maxim-equality) (auto simp: s.isMaxim-def)
      show ?thesis
      proof (intro bexI[OF - f0] ballI)
        fix f assume f: f ∈ F
        show (f0, f) ∈ oexp
        proof (cases f0 = f)
          case True thus ?thesis by (metis F(1) Field-oexp f0 in-mono oexp-Refl
refl-onD)
        next
          case False
          thus ?thesis
          proof (cases s.maxim (SUPP f) = z ∧ f z = x0)
            case True
            with f have f(z := r.zero) ∈ G unfolding G-def by blast
            with g0(2) f0z have (f0(z := r.zero), f(z := r.zero)) ∈ oexp by
auto
            hence oexp: (f0(z := r.zero, z := x0), f(z := r.zero, z := x0)) ∈
oexp
            by (elim fun-upd-same-oexp[OF - - zField x0Field]) simp
            with f F(1) x0min True
            have (f(z := x0), f) ∈ oexp unfolding G-def r.isMinim-def
            by (intro fun-upd-smaller-oexp[OF - zField x0Field]) auto
            with oexp show ?thesis using transD[OF oexp-trans, of f0 f(z :=
x0) f]
            unfolding f0-def by auto
          next
            case False
            thus ?thesis
            proof (cases s.maxim (SUPP f) = z)
              case True
              with notG have f0 z ≠ f z unfolding f0-def by auto
              hence f0 z ≠ f z by metis
              with True maxf0 f0 f SUPPF have s.max-fun-diff f0 f = z
              using s.maxim-Un[of SUPP f0 SUPP f, unfolded s.max2-def]
              unfolding max-fun-diff-alt by (intro trans[OF s.maxim-Int])
auto
              moreover
              from x0min True f have (x0, f z) ∈ r unfolding r.isMinim-def
by auto
              ultimately show ?thesis using f f0 F(1) unfolding oexp-def
f0-def by auto
            next
              case False
              with notG have *: (z, s.maxim (SUPP f)) ∈ s z ≠ s.maxim
(SUPP f)
              using zmin f unfolding s.isMinim-def G-def by auto

```

```

      have f0f: f0 (s.maxim (SUPP f)) = r.zero
    proof (rule ccontr)
      assume f0 (s.maxim (SUPP f)) ≠ r.zero
      with f SUPPF maxF(1) have s.maxim (SUPP f) ∈ SUPP f0
        unfolding support-def[of - - f0] s.isMaxim-def by auto
      with SUPPF f0 have (s.maxim (SUPP f), z) ∈ s unfolding
maxf0[symmetric]
        by (auto intro: s.maxim-greatest)
      with * antisymD[OF s.ANTISYM] show False by simp
    qed
  moreover
    have f (s.maxim (SUPP f)) ≠ r.zero
      using bspec[OF maxF(1) f, unfolded s.isMaxim-def] by (auto
simp: support-def)
    with f0f * f f0 maxf0 SUPPF
    have s.max-fun-diff f0 f = s.maxim (SUPP f0 ∪ SUPP f)
      unfolding max-fun-diff-alt using s.maxim-Un[of SUPP f0 SUPP
f]
    by (intro s.maxim-Int) (auto simp: s.max2-def)
  moreover have s.maxim (SUPP f0 ∪ SUPP f) = s.maxim (SUPP
f)
    using s.maxim-Un[of SUPP f0 SUPP f] * maxf0 SUPPF f0 f
    by (auto simp: s.max2-def)
    ultimately show ?thesis using f f0 F(1) maxF(1) SUPPF
unfolding oexp-def Let-def
      by (fastforce simp: s.isMaxim-def intro!: r.zero-smallest
FINFUNC D)
    qed
  qed
  qed
  qed
  qed simp
  qed
  qed
}
with A show ∃ a ∈ A. ∀ a' ∈ A. (a, a') ∈ oexp
  by blast
qed
qed

```

lemma *oexp-Well-order: Well-order oexp*
unfolding *well-order-on-def* **using** *oexp-Linear-order oexp-wf-Id* **by** *blast*

interpretation *o: wo-rel oexp* **by** *unfold-locales (rule oexp-Well-order)*

lemma *zero-oexp: Field r ≠ {} ⟹ o.zero = const*
by (*metis Field-oexp const-FINFUNC const-least o.Field-ofilter o.equals-minim o.ofilter-def o.zero-def*)

end

notation $wo\text{-}rel2.oexp$ (**infixl** $\langle \hat{\ }o \rangle$ 90)

lemmas $oexp\text{-}def = wo\text{-}rel2.oexp\text{-}def[unfolded\ wo\text{-}rel2\text{-}def, OF\ conjI]$

lemmas $oexp\text{-}Well\text{-}order = wo\text{-}rel2.oexp\text{-}Well\text{-}order[unfolded\ wo\text{-}rel2\text{-}def, OF\ conjI]$

lemmas $Field\text{-}oexp = wo\text{-}rel2.Field\text{-}oexp[unfolded\ wo\text{-}rel2\text{-}def, OF\ conjI]$

definition $ozero = \{\}$

lemma $ozero\text{-}Well\text{-}order[simp]$: *Well-order* $ozero$

unfolding $ozero\text{-}def$ **by** *simp*

lemma $ozero\text{-}ordIso[simp]$: $ozero =_o ozero$

unfolding $ozero\text{-}def\ ordIso\text{-}def\ iso\text{-}def[abs\text{-}def]\ embed\text{-}def\ bij\text{-}betw\text{-}def$ **by** *auto*

lemma $Field\text{-}ozero[simp]$: *Field* $ozero = \{\}$

unfolding $ozero\text{-}def$ **by** *simp*

lemma $iso\text{-}ozero\text{-}empty[simp]$: $r =_o ozero = (r = \{\})$

unfolding $ozero\text{-}def\ ordIso\text{-}def\ iso\text{-}def[abs\text{-}def]\ embed\text{-}def\ bij\text{-}betw\text{-}def$
by (*auto dest: well-order-on-domain*)

lemma $ozero\text{-}ordLeq$:

assumes *Well-order* r **shows** $ozero \leq_o r$

using *assms unfolding ozero-def ordLeq-def embed-def[abs-def] under-def* **by** *auto*

definition $oone = \{((), ())\}$

lemma $oone\text{-}Well\text{-}order[simp]$: *Well-order* $oone$

unfolding $oone\text{-}def$ **unfolding** $well\text{-}order\text{-}on\text{-}def\ linear\text{-}order\text{-}on\text{-}def\ partial\text{-}order\text{-}on\text{-}def$
 $preorder\text{-}on\text{-}def\ total\text{-}on\text{-}def\ refl\text{-}on\text{-}def\ trans\text{-}def\ antisym\text{-}def$ **by** *auto*

lemma $Field\text{-}oone[simp]$: *Field* $oone = \{ () \}$

unfolding $oone\text{-}def$ **by** *simp*

lemma $oone\text{-}ordIso$: $oone =_o \{(x, x)\}$

unfolding $ordIso\text{-}def\ oone\text{-}def\ well\text{-}order\text{-}on\text{-}def\ linear\text{-}order\text{-}on\text{-}def\ partial\text{-}order\text{-}on\text{-}def$
 $preorder\text{-}on\text{-}def\ total\text{-}on\text{-}def\ refl\text{-}on\text{-}def\ trans\text{-}def\ antisym\text{-}def$

by (*auto simp: iso-def embed-def bij-betw-def under-def inj-on-def intro!: exI[of - $\lambda\text{-}. x$]*)

lemma $osum\text{-}ordLeqR$: *Well-order* $r \implies \text{Well-order } s \implies s \leq_o r +_o s$

unfolding $ordLeq\text{-}def2\ underS\text{-}def$

by (*auto intro!: exI[of - Inr] osum-Well-order*) (*auto simp add: osum-def Field-def*)

lemma $osum\text{-}congL$:

assumes $r =_o s$ **and** t : *Well-order* t

shows $r +_o t =_o s +_o t$ (**is** $?L =_o ?R$)

proof –
 from *assms*(1) obtain *f* where *r*: *Well-order* *r* and *s*: *Well-order* *s* and *f*: *iso*
r s f
 unfolding *ordIso-def* by *blast*
 let *?f* = *map-sum f id*
 from *f* have *inj-on ?f (Field ?L)*
 unfolding *Field-osum iso-def bij-betw-def inj-on-def* by *fastforce*
 with *f* have *bij-betw ?f (Field ?L) (Field ?R)*
 unfolding *Field-osum iso-def bij-betw-def image-image image-Un* by *auto*
 moreover from *f* have *compat ?L ?R ?f*
 unfolding *osum-def iso-iff3[OF r s] compat-def bij-betw-def*
 by (*auto simp: map-prod-imageI*)
 ultimately have *iso ?L ?R ?f* by (*subst iso-iff3*) (*auto intro: osum-Well-order*
r s t)
 thus *?thesis* unfolding *ordIso-def* by (*auto intro: osum-Well-order r s t*)
qed

lemma *osum-congR*:
 assumes *r =o s* and *t*: *Well-order* *t*
 shows *t +o r =o t +o s* (*is ?L =o ?R*)
proof –
 from *assms*(1) obtain *f* where *r*: *Well-order* *r* and *s*: *Well-order* *s* and *f*: *iso*
r s f
 unfolding *ordIso-def* by *blast*
 let *?f* = *map-sum id f*
 from *f* have *inj-on ?f (Field ?L)*
 unfolding *Field-osum iso-def bij-betw-def inj-on-def* by *fastforce*
 with *f* have *bij-betw ?f (Field ?L) (Field ?R)*
 unfolding *Field-osum iso-def bij-betw-def image-image image-Un* by *auto*
 moreover from *f* have *compat ?L ?R ?f*
 unfolding *osum-def iso-iff3[OF r s] compat-def bij-betw-def*
 by (*auto simp: map-prod-imageI*)
 ultimately have *iso ?L ?R ?f* by (*subst iso-iff3*) (*auto intro: osum-Well-order*
r s t)
 thus *?thesis* unfolding *ordIso-def* by (*auto intro: osum-Well-order r s t*)
qed

lemma *osum-cong*:
 assumes *t =o u* and *r =o s*
 shows *t +o r =o u +o s*
 using *ordIso-transitive[OF osum-congL[OF assms(1)] osum-congR[OF assms(2)]]*
assms[unfolded ordIso-def] by *auto*

lemma *Well-order-empty[simp]*: *Well-order* {}
 unfolding *Field-empty* by (*rule well-order-on-empty*)

lemma *well-order-on-singleton[simp]*: *well-order-on* {*x*} {(*x*, *x*)}

unfolding *well-order-on-def linear-order-on-def partial-order-on-def preorder-on-def*
total-on-def

Field-def refl-on-def trans-def antisym-def **by** *auto*

lemma *oexp-empty[simp]*:
assumes *Well-order r*
shows $r \hat{=} \{\} = \{(\lambda x. \text{undefined}, \lambda x. \text{undefined})\}$
unfolding *oexp-def[OF assms Well-order-empty] FinFunc-def fin-support-def support-def* **by** *auto*

lemma *oexp-empty2[simp]*:
assumes *Well-order r r $\neq \{\}$*
shows $\{\} \hat{=} r = \{\}$
proof –
from *assms(2)* **have** *Field r $\neq \{\}$* **unfolding** *Field-def* **by** *auto*
thus *?thesis*
by (*simp add: assms(1) wo-rel2.intro wo-rel2.oexp-empty*)
qed

lemma *oprod-zero[simp]*: $\{\} * o r = \{\} r * o \{\} = \{\}$
unfolding *oprod-def* **by** *simp-all*

lemma *oprod-congL*:
assumes $r = o s$ **and** t : *Well-order t*
shows $r * o t = o s * o t$ (**is** $?L = o ?R$)
proof –
from *assms(1)* **obtain** f **where** r : *Well-order r* **and** s : *Well-order s* **and** f : *iso r s f*
unfolding *ordIso-def* **by** *blast*
let $?f = \text{map-prod } f \text{ id}$
from f **have** *inj-on ?f (Field ?L)*
unfolding *Field-oprod iso-def bij-betw-def inj-on-def* **by** *fastforce*
with f **have** *bij-betw ?f (Field ?L) (Field ?R)*
unfolding *Field-oprod iso-def bij-betw-def* **by** (*auto intro!: map-prod-surj-on*)
moreover from f **have** *compat ?L ?R ?f*
unfolding *iso-iff3[OF r s] compat-def oprod-def bij-betw-def*
by (*auto simp: map-prod-imageI*)
ultimately have *iso ?L ?R ?f* **by** (*subst iso-iff3*) (*auto intro: oprod-Well-order r s t*)
thus *?thesis* **unfolding** *ordIso-def* **by** (*auto intro: oprod-Well-order r s t*)
qed

lemma *oprod-congR*:
assumes $r = o s$ **and** t : *Well-order t*
shows $t * o r = o t * o s$ (**is** $?L = o ?R$)
proof –
from *assms(1)* **obtain** f **where** r : *Well-order r* **and** s : *Well-order s* **and** f : *iso r s f*
unfolding *ordIso-def* **by** *blast*
let $?f = \text{map-prod id } f$
from f **have** *inj-on ?f (Field ?L)*

unfolding *Field-oprod iso-def bij-betw-def inj-on-def* **by** *fastforce*
with *f* **have** *bij-betw ?f (Field ?L) (Field ?R)*
unfolding *Field-oprod iso-def bij-betw-def* **by** *(auto intro!: map-prod-surj-on)*
moreover from *f well-order-on-domain[OF r]* **have** *compat ?L ?R ?f*
unfolding *iso-iff3[OF r s] compat-def oprod-def bij-betw-def*
by *(auto simp: map-prod-imageI dest: inj-onD)*
ultimately have *iso ?L ?R ?f* **by** *(subst iso-iff3) (auto intro: oprod-Well-order r s t)*
thus ?thesis unfolding ordIso-def **by** *(auto intro: oprod-Well-order r s t)*
qed

lemma *oprod-cong*:
assumes *t =o u and r =o s*
shows *t *o r =o u *o s*
using *ordIso-transitive[OF oprod-congL[OF assms(1)] oprod-congR[OF assms(2)]]*
assms[unfolded ordIso-def] **by** *auto*

lemma *Field-singleton[simp]*: *Field {(z,z)} = {z}*
by *(metis well-order-on-Field well-order-on-singleton)*

lemma *zero-singleton[simp]*: *zero {(z,z)} = z*
using *wo-rel.zero-in-Field[unfolded wo-rel-def, of {(z, z)}] well-order-on-singleton[of z]*
by *auto*

lemma *FinFunc-singleton*: *FinFunc {(z,z)} s = {λx. if x ∈ Field s then z else undefined}*
unfolding *FinFunc-def Func-def fin-support-def support-def*
by *(auto simp: fun-eq-iff split: if-split-asm intro!: finite-subset[of - { }])*

lemma *oone-ordIso-oexp*:
assumes *r =o oone and s: Well-order s*
shows *r ^o s =o oone (is ?L =o ?R)*
proof –
from *⟨r =o oone⟩* **obtain** *f* **where** **: ∀x∈Field r. ∀y∈Field r. x = y and f ‘*
Field r = {() }
and *r: Well-order r*
unfolding *ordIso-def oone-def* **by** *(auto simp add: iso-def [abs-def] bij-betw-def inj-on-def)*
then obtain *x* **where** *x ∈ Field r* **by** *auto*
with *** **have** *Fr: Field r = {x}* **by** *auto*
interpret *r: wo-rel r* **by** *unfold-locales (rule r)*
from *Fr well-order-on-domain[OF r] refl-onD[OF r.REFL, of x]* **have** *r-def: r*
= {(x, x)} **by** *fast*
interpret *wo-rel2 r s* **by** *unfold-locales (rule r, rule s)*
have *bij-betw (λx. ()) (Field ?L) (Field ?R)*
unfolding *bij-betw-def Field-oexp* **by** *(auto simp: r-def FinFunc-singleton)*
moreover have *compat ?L ?R (λx. ())* **unfolding** *compat-def oone-def* **by** *auto*
ultimately have *iso ?L ?R (λx. ())* **using** *s oone-Well-order*

by (subst iso-iff3) (auto intro: oexp-Well-order)
 thus ?thesis using s oone-Well-order unfolding ordIso-def by (auto intro:
 oexp-Well-order)
 qed

context
 fixes r s t
 assumes r: Well-order r
 assumes s: Well-order s
 assumes t: Well-order t
 begin

lemma osum-zeroL: ozero +o r =o r
 using r unfolding osum-def ozero-def by (auto intro: map-prod-ordIso)

lemma osum-zeroR: r +o ozero =o r
 using r unfolding osum-def ozero-def by (auto intro: map-prod-ordIso)

lemma osum-assoc: (r +o s) +o t =o r +o s +o t (is ?L =o ?R)

proof –
 let ?f =
 λrst. case rst of Inl (Inl r) ⇒ Inl r | Inl (Inr s) ⇒ Inr (Inl s) | Inr t ⇒ Inr
 (Inr t)
 have bij-betw ?f (Field ?L) (Field ?R)
 unfolding Field-osum bij-betw-def inj-on-def by (auto simp: image-Un im-
 age-iff)
 moreover
 have compat ?L ?R ?f
 proof (unfold compat-def, safe)
 fix a b
 assume (a, b) ∈ ?L
 thus (?f a, ?f b) ∈ ?R
 unfolding osum-def[of r +o s t] osum-def[of r s +o t] Field-osum
 unfolding osum-def Field-osum image-iff image-Un map-prod-def
 by fastforce
 qed
 ultimately have iso ?L ?R ?f using r s t by (subst iso-iff3) (auto intro:
 osum-Well-order)
 thus ?thesis using r s t unfolding ordIso-def by (auto intro: osum-Well-order)
 qed

lemma osum-monoR:
 assumes s <o t
 shows r +o s <o r +o t (is ?L <o ?R)
 proof –
 from assms obtain f where s: Well-order s and t: Well-order t and embedS s
 t f
 unfolding ordLess-def by blast

hence *: *inj-on* f (*Field* s) *compat* s t f *ofilter* t (f ‘ *Field* s) f ‘ *Field* $s \subset \text{Field } t$
 using *embed-iff-compat-inj-on-ofilter*[*OF* s t , *of* f] *embedS-iff*[*OF* s , *of* t f]
 unfolding *embedS-def* by *auto*
 let $?f = \text{map-sum } \text{id } f$
 from *(1) have *inj-on* $?f$ (*Field* $?L$) unfolding *Field-osum inj-on-def* by *fastforce*
 moreover
 from *(2,4) have *compat* $?L$ $?R$ $?f$ unfolding *compat-def osum-def map-prod-def*
 by *fastforce*
 moreover
 interpret t : *wo-rel* t by *unfold-locales* (*rule* t)
 interpret rt : *wo-rel* $?R$ by *unfold-locales* (*rule* *osum-Well-order*[*OF* r t])
 from *(3) have *ofilter* $?R$ ($?f$ ‘ *Field* $?L$)
 unfolding *t.ofilter-def rt.ofilter-def Field-osum image-Un image-image under-def*
 by (*auto simp: osum-def intro!: imageI*) (*auto simp: Field-def*)
 ultimately have *embed* $?L$ $?R$ $?f$ using *embed-iff-compat-inj-on-ofilter*[*of* $?L$ $?R$ $?f$]
 by (*auto intro: osum-Well-order r s t*)
 moreover
 from *(4) have $?f$ ‘ *Field* $?L \subset \text{Field } ?R$ unfolding *Field-osum image-Un image-image* by *auto*
 ultimately have *embedS* $?L$ $?R$ $?f$ using *embedS-iff*[*OF* *osum-Well-order*[*OF* r s], *of* $?R$ $?f$] by *auto*
 thus $?thesis$ unfolding *ordLess-def* by (*auto intro: osum-Well-order r s t*)
 qed

lemma *osum-monoL*:

assumes $r \leq_o s$
 shows $r +_o t \leq_o s +_o t$
proof –
 from *assms* obtain f where f : $\forall a \in \text{Field } r. f \ a \in \text{Field } s \wedge f$ ‘ *underS* r $a \subseteq \text{underS } s$ ($f \ a$)
 unfolding *ordLeq-def2* by *blast*
 let $?f = \text{map-sum } f \ \text{id}$
 from f have $\forall a \in \text{Field } (r +_o t).$
 $?f \ a \in \text{Field } (s +_o t) \wedge ?f$ ‘ *underS* $(r +_o t)$ $a \subseteq \text{underS } (s +_o t)$ ($?f \ a$)
 unfolding *Field-osum underS-def* by (*fastforce simp: osum-def*)
 thus $?thesis$ unfolding *ordLeq-def2* by (*auto intro: osum-Well-order r s t*)
 qed

lemma *oprod-zeroL*: *zero* $*_o r =_o \text{zero}$

using *zero-ordIso* unfolding *zero-def* by *simp*

lemma *oprod-zeroR*: $r *_o \text{zero} =_o \text{zero}$

using *zero-ordIso* unfolding *zero-def* by *simp*

lemma *oprod-ooneR*: $r *_o \text{oone} =_o r$ (is $?L =_o ?R$)

proof –

have *bij-betw fst* (*Field ?L*) (*Field ?R*) **unfolding** *Field-oprod bij-betw-def inj-on-def*
by *simp*
moreover **have** *compat ?L ?R fst* **unfolding** *compat-def oprod-def* **by** *auto*
ultimately **have** *iso ?L ?R fst* **using** *r oone-Well-order*
by (*subst iso-iff3*) (*auto intro: oprod-Well-order*)
thus *?thesis* **using** *r oone-Well-order* **unfolding** *ordIso-def* **by** (*auto intro: oprod-Well-order*)
qed

lemma *oprod-ooneL: oone *o r =o r (is ?L =o ?R)*
proof –
have *bij-betw snd* (*Field ?L*) (*Field ?R*) **unfolding** *Field-oprod bij-betw-def*
inj-on-def **by** *simp*
moreover **have** *Refl r* **by** (*rule wo-rel.REFL[unfolded wo-rel-def, OF r]*)
hence *compat ?L ?R snd* **unfolding** *compat-def oprod-def refl-on-def* **by** *auto*
ultimately **have** *iso ?L ?R snd* **using** *r oone-Well-order*
by (*subst iso-iff3*) (*auto intro: oprod-Well-order*)
thus *?thesis* **using** *r oone-Well-order* **unfolding** *ordIso-def* **by** (*auto intro: oprod-Well-order*)
qed

lemma *oprod-monoR:*
assumes *ozero <o r s <o t*
shows *r *o s <o r *o t (is ?L <o ?R)*
proof –
from *assms* **obtain** *f* **where** *s: Well-order s* **and** *t: Well-order t* **and** *embedS s t f*
unfolding *ordLess-def* **by** *blast*
hence ***: *inj-on f* (*Field s*) *compat s t f ofilter t* (*f ' Field s*) *f ' Field s* \subset *Field t*
using *embed-iff-compat-inj-on-ofilter[OF s t, of f]* *embedS-iff[OF s, of t f]*
unfolding *embedS-def* **by** *auto*
let *?f = map-prod id f*
from **(1)* **have** *inj-on ?f* (*Field ?L*) **unfolding** *Field-oprod inj-on-def* **by** *fast-force*
moreover
from **(2,4)* *the-inv-into-f-f[OF *(1)]* **have** *compat ?L ?R ?f* **unfolding** *compat-def oprod-def*
by *auto* (*metis well-order-on-domain t, metis well-order-on-domain s*)
moreover
interpret *t: wo-rel t* **by** *unfold-locales* (*rule t*)
interpret *rt: wo-rel ?R* **by** *unfold-locales* (*rule oprod-Well-order[OF r t]*)
from **(3)* **have** *ofilter ?R* (*?f ' Field ?L*)
unfolding *t.ofilter-def rt.ofilter-def Field-oprod under-def*
by (*auto simp: oprod-def image-iff*) (*fast | metis r well-order-on-domain*) +
ultimately **have** *embed ?L ?R ?f* **using** *embed-iff-compat-inj-on-ofilter[of ?L ?R ?f]*
by (*auto intro: oprod-Well-order r s t*)
moreover
from *not-ordLess-ordIso[OF assms(1)]* **have** *r ≠ {}* **by** (*metis ozero-def ozero-ordIso*)

hence $\text{Field } r \neq \{\}$ **unfolding** *Field-def* **by** *auto*
 with $\ast(4)$ **have** $?f \text{ ' } \text{Field } ?L \subset \text{Field } ?R$ **unfolding** *Field-oprod*
by *auto* (*metis SigmaD2 SigmaI map-prod-surj-on*)
 ultimately **have** $\text{embedS } ?L ?R ?f$ **using** $\text{embedS-iff}[OF \text{ oprod-Well-order}[OF$
 $r \ s], \text{ of } ?R ?f]$ **by** *auto*
 thus $?thesis$ **unfolding** *ordLess-def* **by** (*auto intro: oprod-Well-order r s t*)
qed

lemma *oprod-monoL*:

assumes $r \leq_o s$

shows $r \ast_o t \leq_o s \ast_o t$

proof –

from *assms* **obtain** f **where** $f: \forall a \in \text{Field } r. f \ a \in \text{Field } s \wedge f \text{ ' } \text{underS } r \ a \subseteq$
 $\text{underS } s \ (f \ a)$

unfolding *ordLeq-def2* **by** *blast*

let $?f = \text{map-prod } f \text{ id}$

from f **have** $\forall a \in \text{Field } (r \ast_o t).$

$?f \ a \in \text{Field } (s \ast_o t) \wedge ?f \text{ ' } \text{underS } (r \ast_o t) \ a \subseteq \text{underS } (s \ast_o t) \ (?f \ a)$

unfolding *Field-oprod underS-def* **unfolding** *map-prod-def oprod-def* **by** *auto*

thus $?thesis$ **unfolding** *ordLeq-def2* **by** (*auto intro: oprod-Well-order r s t*)

qed

lemma *oprod-assoc*: $(r \ast_o s) \ast_o t =_o r \ast_o s \ast_o t$ (**is** $?L =_o ?R$)

proof –

let $?f = \lambda((a,b),c). (a,b,c)$

have $\text{bij-betw } ?f \ (\text{Field } ?L) \ (\text{Field } ?R)$

unfolding *Field-oprod bij-betw-def inj-on-def* **by** (*auto simp: image-Un image-iff*)

moreover

have $\text{compat } ?L ?R ?f$

proof (*unfold compat-def, safe*)

fix $a1 \ a2 \ a3 \ b1 \ b2 \ b3$

assume $((a1, a2), a3), ((b1, b2), b3) \in ?L$

thus $((a1, a2, a3), (b1, b2, b3)) \in ?R$

unfolding *oprod-def[of r *o s t] oprod-def[of r s *o t] Field-oprod*

unfolding *oprod-def Field-oprod image-iff image-Un* **by** *fast*

qed

ultimately have $\text{iso } ?L ?R ?f$ **using** $r \ s \ t$ **by** (*subst iso-iff3*) (*auto intro: oprod-Well-order*)

thus $?thesis$ **using** $r \ s \ t$ **unfolding** *ordIso-def* **by** (*auto intro: oprod-Well-order*)

qed

lemma *oprod-osum*: $r \ast_o (s +_o t) =_o r \ast_o s +_o r \ast_o t$ (**is** $?L =_o ?R$)

proof –

let $?f = \lambda(a,bc). \text{ case } bc \text{ of } \text{Inl } b \Rightarrow \text{Inl } (a, b) \mid \text{Inr } c \Rightarrow \text{Inr } (a, c)$

have $\text{bij-betw } ?f \ (\text{Field } ?L) \ (\text{Field } ?R)$ **unfolding** *Field-oprod Field-osum bij-betw-def inj-on-def*

by (*fastforce simp: image-Un image-iff split: sum.splits*)

moreover

```

have compat ?L ?R ?f
proof (unfold compat-def, intro allI impI)
  fix a b
  assume (a, b) ∈ ?L
  thus (?f a, ?f b) ∈ ?R
    unfolding oprod-def[of r s +o t] osum-def[of r *o s r *o t] Field-oprod
Field-osum
    unfolding oprod-def osum-def Field-oprod Field-osum image-iff image-Un by
auto
qed
ultimately have iso ?L ?R ?f using r s t
  by (subst iso-iff3) (auto intro: oprod-Well-order osum-Well-order)
  thus ?thesis using r s t unfolding ordIso-def by (auto intro: oprod-Well-order
osum-Well-order)
qed

lemma ozero-oexp:  $\neg (s =_o \text{ozero}) \implies \text{ozero} \hat{=}_o s =_o \text{ozero}$ 
  by (fastforce simp add: oexp-def[OF ozero-Well-order s] FinFunc-def Func-def
intro: FieldI1)

lemma oone-oexp:  $\text{oone} \hat{=}_o s =_o \text{oone} \text{ (is } ?L =_o ?R \text{)}$ 
  by (rule oone-ordIso-oexp[OF ordIso-reflexive[OF oone-Well-order] s])

lemma oexp-monoR:
  assumes  $\text{oone} <_o r \ s <_o t$ 
  shows  $r \hat{=}_o s <_o r \hat{=}_o t \text{ (is } ?L <_o ?R \text{)}$ 
proof -
  interpret rs: wo-rel2 r s by unfold-locales (rule r, rule s)
  interpret rt: wo-rel2 r t by unfold-locales (rule r, rule t)
  interpret rexpt: wo-rel r  $\hat{=}_o t$  by unfold-locales (rule rt.oexp-Well-order)
  interpret r: wo-rel r by unfold-locales (rule r)
  interpret s: wo-rel s by unfold-locales (rule s)
  interpret t: wo-rel t by unfold-locales (rule t)
  have Field r  $\neq \{\}$  by (metis assms(1) internalize-ordLess not-psubset-empty)
  moreover
  { assume Field r = {r.zero}
    hence r = {(r.zero, r.zero)} using refl-onD[OF r.REFL, of r.zero] unfolding
Field-def by auto
    hence r =o oone by (metis oone-ordIso ordIso-symmetric)
    with not-ordLess-ordIso[OF assms(1)] have False by (metis ordIso-symmetric)
  }
  ultimately obtain x where x:  $x \in \text{Field } r \ r.\text{zero} \in \text{Field } r \ x \neq r.\text{zero}$ 
  by (metis insert-iff r.zero-in-Field subsetI subset-singletonD)
  moreover from assms(2) obtain f where embedS s t f unfolding ordLess-def
by blast
  hence *: inj-on f (Field s) compat s t f ofilter t (f ' Field s) f ' Field s  $\subset$  Field t
  using embed-iff-compat-inj-on-ofilter[OF s t, of f] embedS-iff[OF s, of t f]
  unfolding embedS-def by auto
  note invff = the-inv-into-f-f[OF *(1)] and injfD = inj-onD[OF *(1)]

```

```

define  $F$  where  $[abs-def]$ :  $F\ g\ z =$ 
  (if  $z \in f \text{ ' } Field\ s$  then  $g\ (the-inv-into\ (Field\ s)\ f\ z)$ 
  else if  $z \in Field\ t$  then  $r.zero$  else  $undefined$ ) for  $g\ z$ 
from  $*(4)\ x(2)\ the-inv-into-f-eq[OF\ *(1)]$  have  $FLR$ :  $F \text{ ' } Field\ ?L \subseteq Field\ ?R$ 
unfolding  $rt.Field-oexp\ rs.Field-oexp\ FinFunc-def\ Func-def\ fin-support-def\ sup-$ 
 $port-def\ F-def$ 
by  $(fastforce\ split:\ option.splits\ if-split-asm\ elim!:\ finite-surj[of\ -\ -\ f])$ 
have  $inj-on\ F\ (Field\ ?L)$  unfolding  $rs.Field-oexp\ inj-on-def\ fun-eq-iff$ 
proof  $safe$ 
  fix  $g\ h\ x$  assume  $g \in FinFunc\ r\ s\ h \in FinFunc\ r\ s\ \forall y. F\ g\ y = F\ h\ y$ 
  with  $invff$  show  $g\ x = h\ x$  unfolding  $F-def\ fun-eq-iff\ FinFunc-def\ Func-def$ 
  by  $auto\ (metis\ image-eqI)$ 
qed
moreover
have  $compat\ ?L\ ?R\ F$  unfolding  $compat-def\ rs.oexp-def\ rt.oexp-def$ 
proof  $(safe\ elim!:\ bspec[OF\ iffD1[OF\ image-subset-iff\ FLR[unfolded\ rs.Field-oexp\ rt.Field-oexp]]])$ 
  fix  $g\ h$  assume  $gh$ :  $g \in FinFunc\ r\ s\ h \in FinFunc\ r\ s\ F\ g \neq F\ h$ 
  let  $m = s.max-fun-diff\ g\ h$  in  $(g\ m,\ h\ m) \in r$ 
  hence  $g \neq h$  by  $auto$ 
  note  $max-fun-diff-in = rs.max-fun-diff-in[OF\ \langle g \neq h \rangle\ gh(1,2)]$ 
  and  $max-fun-diff-max = rs.max-fun-diff-max[OF\ \langle g \neq h \rangle\ gh(1,2)]$ 
  with  $*(4)\ invff\ *(2)$  have  $t.max-fun-diff\ (F\ g)\ (F\ h) = f\ (s.max-fun-diff\ g\ h)$ 
  unfolding  $t.max-fun-diff-def\ compat-def$ 
  by  $(intro\ t.maxim-equality)\ (auto\ simp:\ t.isMaxim-def\ F-def\ dest:\ injfD)$ 
  with  $gh\ invff\ max-fun-diff-in$ 
  show let  $m = t.max-fun-diff\ (F\ g)\ (F\ h)$  in  $(F\ g\ m,\ F\ h\ m) \in r$ 
  unfolding  $F-def\ Let-def$  by  $(auto\ simp:\ dest:\ injfD)$ 
qed
moreover
from  $FLR$  have  $ofilter\ ?R\ (F \text{ ' } Field\ ?L)$ 
unfolding  $rexp.ofilter-def\ under-def\ rs.Field-oexp\ rt.Field-oexp$  unfolding
 $rt.oexp-def$ 
proof  $(safe\ elim!:\ imageI)$ 
  fix  $g\ h$  assume  $gh$ :  $g \in FinFunc\ r\ s\ h \in FinFunc\ r\ t\ F\ g \in FinFunc\ r\ t$ 
  let  $m = t.max-fun-diff\ h\ (F\ g)$  in  $(h\ m,\ F\ g\ m) \in r$ 
  thus  $h \in F \text{ ' } FinFunc\ r\ s$ 
  proof  $(cases\ h = F\ g)$ 
    case  $False$ 
    hence  $max-Field$ :  $t.max-fun-diff\ h\ (F\ g) \in \{a \in Field\ t. h\ a \neq F\ g\ a\}$ 
    by  $(rule\ rt.max-fun-diff-in[OF\ -\ gh(2,3)])$ 
    { assume  $*$ :  $t.max-fun-diff\ h\ (F\ g) \notin f \text{ ' } Field\ s$ 
      with  $max-Field$  have  $**$ :  $F\ g\ (t.max-fun-diff\ h\ (F\ g)) = r.zero$  unfolding
 $F-def$  by  $auto$ 
      with  $*\ gh(4)$  have  $h\ (t.max-fun-diff\ h\ (F\ g)) = r.zero$  unfolding  $Let-def$ 
by  $auto$ 
      with  $**$  have  $False$  using  $max-Field\ gh(2,3)$  unfolding  $FinFunc-def\ Func-def$  by  $auto$ 
    }
  }

```

```

hence max-f-Field: t.max-fun-diff h (F g) ∈ f ‘ Field s by blast
{ fix z assume z: z ∈ Field t – f ‘ Field s
  have (t.max-fun-diff h (F g), z) ∈ t
  proof (rule ccontr)
    assume (t.max-fun-diff h (F g), z) ∉ t
    hence (z, t.max-fun-diff h (F g)) ∈ t using t.in-notinI[of t.max-fun-diff
h (F g) z]
      z max-Field by auto
    hence z ∈ f ‘ Field s using *(3) max-f-Field unfolding t.ofilter-def
under-def
      by fastforce
    with z show False by blast
  qed
  hence h z = r.zero using rt.max-fun-diff-le-eq[OF - False gh(2,3), of z]
    z max-f-Field unfolding F-def by auto
} note ** = this
with *(3) gh(2) have h = F ( $\lambda x. \text{if } x \in \text{Field } s \text{ then } h(f\ x) \text{ else undefined}$ )
using invff
  unfolding F-def fun-eq-iff FinFunc-def Func-def Let-def t.ofilter-def under-def
by auto
  moreover from gh(2) *(1,3) have ( $\lambda x. \text{if } x \in \text{Field } s \text{ then } h(f\ x) \text{ else undefined}$ ) ∈ FinFunc r s
  unfolding FinFunc-def Func-def fin-support-def support-def t.ofilter-def
under-def
    by (auto intro: subset-inj-on elim!: finite-imageD[OF finite-subset[rotated]])
  ultimately show ?thesis by (rule image-eqI)
qed simp
qed
ultimately have embed ?L ?R F using embed-iff-compat-inj-on-ofilter[of ?L ?R
F]
  by (auto intro: oexp-Well-order r s t)
moreover
from FLR have F ‘ Field ?L ⊂ Field ?R
proof (intro psubsetI)
  from *(4) obtain z where z: z ∈ Field t z ∉ f ‘ Field s by auto
  define h where [abs-def]: h z =
    (if z ‘ ∈ Field t then if z = z then x else r.zero else undefined) for z ‘
  from z x(3) have rt.SUPP h = {z} unfolding support-def h-def by simp
  with x have h ∈ Field ?R unfolding h-def rt.Field-oexp FinFunc-def Func-def
fin-support-def
    by auto
  moreover
  { fix g
    from z have F g z = r.zero h z = x unfolding support-def h-def F-def by
auto
    with x(3) have F g ≠ h unfolding fun-eq-iff by fastforce
  }
  hence h ∉ F ‘ Field ?L by blast
  ultimately show F ‘ Field ?L ≠ Field ?R by blast

```

```

qed
ultimately have embedS ?L ?R F using embedS-iff[OF rs.oexp-Well-order, of
?R F] by auto
thus ?thesis unfolding ordLess-def using r s t by (auto intro: oexp-Well-order)
qed

lemma oexp-monoL:
  assumes  $r \leq_o s$ 
  shows  $r \hat{\circ} t \leq_o s \hat{\circ} t$ 
proof -
  interpret rt: wo-rel2 r t by unfold-locales (rule r, rule t)
  interpret st: wo-rel2 s t by unfold-locales (rule s, rule t)
  interpret r: wo-rel r by unfold-locales (rule r)
  interpret s: wo-rel s by unfold-locales (rule s)
  interpret t: wo-rel t by unfold-locales (rule t)
  show ?thesis
  proof (cases t = {})
    case True thus ?thesis using r s unfolding ordLeq-def2 underS-def by auto
  next
    case False thus ?thesis
    proof (cases r = {})
      case True thus ?thesis using t <t ≠ {}> st.oexp-Well-order ozero-ordLeq[unfolded
zero-def]
        by auto
    next
      case False
      from assms obtain f where f: embed r s f unfolding ordLeq-def by blast
      hence f-underS:  $\forall a \in \text{Field } r. f a \in \text{Field } s \wedge f ' \text{underS } r a \subseteq \text{underS } s (f a)$ 
        using embed-in-Field embed-underS2 rt.rWELL by fastforce
      from f <t ≠ {}> False have *:  $\text{Field } r \neq \{\} \text{Field } s \neq \{\} \text{Field } t \neq \{\}$ 
        unfolding Field-def embed-def under-def bij-betw-def by auto
      with f obtain x where s.zero = f x  $x \in \text{Field } r$  unfolding embed-def
      bij-betw-def
        using s.zero-under subsetD[OF under-Field[of r]]
      by (metis (no-types, lifting) f-inv-into-f f-underS inv-into-into r.zero-in-Field)
      with f have fz:  $f r.\text{zero} = s.\text{zero}$  and inj: inj-on f (Field r) and compat:
      compat r s f
        unfolding embed-iff-compat-inj-on-ofilter[OF r s] compat-def
        by (fastforce intro: s.leq-zero-imp)+
      let ?f =  $\lambda g x. \text{if } x \in \text{Field } t \text{ then } f (g x) \text{ else undefined}$ 
      { fix g assume g:  $g \in \text{Field } (r \hat{\circ} t)$ 
        with fz f-underS have Field-fg:  $?f g \in \text{Field } (s \hat{\circ} t)$ 
        unfolding st.Field-oexp rt.Field-oexp FinFunc-def Func-def fin-support-def
        support-def
          by (auto elim!: finite-subset[rotated])
        moreover
        have ?f 'underS (r  $\hat{\circ} t$ ) g  $\subseteq$  underS (s  $\hat{\circ} t$ ) (?f g)
        proof safe
          fix h

```

```

    assume h-underS:  $h \in \text{underS } (r \hat{\circ} t) g$ 
    hence  $h \in \text{Field } (r \hat{\circ} t)$  unfolding underS-def Field-def by auto
    with fz f-underS have Field-fh:  $?f h \in \text{Field } (s \hat{\circ} t)$ 
    unfolding st.Field-oexp rt.Field-oexp FinFunc-def Func-def fin-support-def
support-def
    by (auto elim!: finite-subset[rotated])
    from h-underS have  $h \neq g$  and hg:  $(h, g) \in \text{rt.oexp}$  unfolding underS-def
by auto
    with f inj have neg:  $?f h \neq ?f g$ 
    unfolding fun-eq-iff inj-on-def rt.oexp-def map-option-case FinFunc-def
Func-def Let-def
    by simp metis
    with hg have  $t.\text{max-fun-diff } (?f h) (?f g) = t.\text{max-fun-diff } h g$  unfolding
rt.oexp-def
    using rt.max-fun-diff[OF  $\langle h \neq g \rangle$ ] rt.max-fun-diff-in[OF  $\langle h \neq g \rangle$ ]
    by (subst t.max-fun-diff-def, intro t.maxim-equality)
    (auto simp: t.isMaxim-def intro: inj-onD[OF inj] intro!: rt.max-fun-diff-max)
    with Field-fg Field-fh hg fz f-underS compat neg have  $(?f h, ?f g) \in \text{st.oexp}$ 
    using rt.max-fun-diff[OF  $\langle h \neq g \rangle$ ] rt.max-fun-diff-in[OF  $\langle h \neq g \rangle$ ]
unfolding st.Field-oexp
    unfolding rt.oexp-def st.oexp-def Let-def compat-def by auto
    with neg show  $?f h \in \text{underS } (s \hat{\circ} t) (?f g)$  unfolding underS-def by
auto
    qed
    ultimately have  $?f g \in \text{Field } (s \hat{\circ} t) \wedge ?f \text{ 'underS } (r \hat{\circ} t) g \subseteq \text{underS}$ 
 $(s \hat{\circ} t) (?f g)$ 
    by blast
  }
  thus ?thesis unfolding ordLeq-def2 by (fastforce intro: oexp-Well-order r s
t)
qed
qed
qed

```

lemma ordLeq-oexp2:

```

  assumes oone < o r
  shows  $s \leq_o r \hat{\circ} s$ 
proof –
  interpret rs: wo-rel2 r s by unfold-locales (rule r, rule s)
  interpret r: wo-rel r by unfold-locales (rule r)
  interpret s: wo-rel s by unfold-locales (rule s)
  from assms well-order-on-domain[OF r] obtain x where
    x:  $x \in \text{Field } r$   $r.\text{zero} \in \text{Field } r$   $x \neq r.\text{zero}$ 
  unfolding ordLess-def oone-def embedS-def[abs-def] bij-betw-def embed-def un-
der-def
  by (auto simp: image-def)
  (metis (lifting) equals0D mem-Collect-eq r.zero-in-Field singletonI)
  let  $?f = \lambda a b. \text{if } b \in \text{Field } s \text{ then if } b = a \text{ then } x \text{ else } r.\text{zero} \text{ else undefined}$ 
  from x(3) have SUPP:  $\bigwedge y. y \in \text{Field } s \implies \text{rs.SUPP } (?f y) = \{y\}$  unfolding

```



```

support-def by auto
{ fix y assume y: y ∈ Field s
  with x(1,2) SUPP have ?f y ∈ Field (r ^o s) unfolding rs.Field-oexp
    by (auto simp: FinFunc-def Func-def fin-support-def)
  moreover
  have ?f ' underS s y ⊆ underS (r ^o s) (?f y)
  proof safe
    fix z
    assume z ∈ underS s y
    hence z: z ≠ y (z, y) ∈ s z ∈ Field s unfolding underS-def Field-def by
auto
    from x(3) y z(1,3) have ?f z ≠ ?f y unfolding fun-eq-iff by auto
    moreover
    { from x(1,2) have ?f z ∈ FinFunc r s ?f y ∈ FinFunc r s
      unfolding FinFunc-def Func-def fin-support-def by (auto simp: SUPP[OF
z(3)] SUPP[OF y])
      moreover
      from x(3) y z(1,2) refl-onD[OF s.REFL] have s.max-fun-diff (?f z) (?f y)
= y
        unfolding rs.max-fun-diff-alt SUPP[OF z(3)] SUPP[OF y]
        by (intro s.maxim-equality) (auto simp: s.isMaxim-def)
      ultimately have (?f z, ?f y) ∈ rs.oexp using y x(1)
      unfolding rs.oexp-def Let-def by auto
    }
    ultimately show ?f z ∈ underS (r ^o s) (?f y) unfolding underS-def by
blast
  }
  qed
  ultimately have ?f y ∈ Field (r ^o s) ∧ ?f ' underS s y ⊆ underS (r ^o s) (?f
y) by blast
}
thus ?thesis unfolding ordLeq-def2 by (fast intro: oexp-Well-order r s)
qed

lemma FinFunc-osum:
  fg ∈ FinFunc r (s +o t) = (fg o Inl ∈ FinFunc r s ∧ fg o Inr ∈ FinFunc r t)
  (is ?L = (?R1 ∧ ?R2))
proof safe
  assume ?L
  from ⟨?L⟩ show ?R1 unfolding FinFunc-def Field-osum Func-def Int-iff fin-support-Field-osum
o-def
    by (auto split: sum.splits)
  from ⟨?L⟩ show ?R2 unfolding FinFunc-def Field-osum Func-def Int-iff fin-support-Field-osum
o-def
    by (auto split: sum.splits)
next
  assume ?R1 ?R2
  thus ?L unfolding FinFunc-def Field-osum Func-def
    by (auto simp: fin-support-Field-osum o-def image-iff split: sum.splits) (metis
sumE)

```

qed

lemma *max-fun-diff-eq-Inl*:

assumes *wo-rel.max-fun-diff* ($s + o\ t$) (*case-sum* $f1\ g1$) (*case-sum* $f2\ g2$) = *Inl* x
 $case-sum\ f1\ g1 \neq case-sum\ f2\ g2$
 $case-sum\ f1\ g1 \in FinFunc\ r\ (s + o\ t)$ $case-sum\ f2\ g2 \in FinFunc\ r\ (s + o\ t)$
shows *wo-rel.max-fun-diff* $s\ f1\ f2 = x$ (**is** $?P$) $g1 = g2$ (**is** $?Q$)
proof –
interpret *st*: *wo-rel* $s + o\ t$ **by** *unfold-locales* (*rule* *osum-Well-order*[*OF* $s\ t$])
interpret *s*: *wo-rel* s **by** *unfold-locales* (*rule* s)
interpret *rst*: *wo-rel2* $r\ s + o\ t$ **by** *unfold-locales* (*rule* r , *rule* *osum-Well-order*[*OF* $s\ t$])
from *assms*(1) **have** *: *st.isMaxim* $\{a \in Field\ (s + o\ t). case-sum\ f1\ g1\ a \neq case-sum\ f2\ g2\ a\}$ (*Inl* x)
using *rst.isMaxim-max-fun-diff*[*OF* *assms*(2–4)] **by** *simp*
hence *s.isMaxim* $\{a \in Field\ s. f1\ a \neq f2\ a\}$ x
unfolding *st.isMaxim-def* *s.isMaxim-def* *Field-osum* **by** (*auto* *simp*: *osum-def*)
thus $?P$ **unfolding** *s.max-fun-diff-def* **by** (*rule* *s.maxim-equality*)
from *assms*(3,4) **have** **: $g1 \in FinFunc\ r\ t\ g2 \in FinFunc\ r\ t$ **unfolding** *FinFunc-osum*
by (*auto* *simp*: *o-def*)
show $?Q$
proof
fix x
from * ** **show** $g1\ x = g2\ x$ **unfolding** *st.isMaxim-def* *Field-osum* *FinFunc-def* *Func-def* *fun-eq-iff*
unfolding *osum-def* **by** (*case-tac* $x \in Field\ t$) *auto*
qed
qed

lemma *max-fun-diff-eq-Inr*:

assumes *wo-rel.max-fun-diff* ($s + o\ t$) (*case-sum* $f1\ g1$) (*case-sum* $f2\ g2$) = *Inr* x
 $case-sum\ f1\ g1 \neq case-sum\ f2\ g2$
 $case-sum\ f1\ g1 \in FinFunc\ r\ (s + o\ t)$ $case-sum\ f2\ g2 \in FinFunc\ r\ (s + o\ t)$
shows *wo-rel.max-fun-diff* $t\ g1\ g2 = x$ (**is** $?P$) $g1 \neq g2$ (**is** $?Q$)
proof –
interpret *st*: *wo-rel* $s + o\ t$ **by** *unfold-locales* (*rule* *osum-Well-order*[*OF* $s\ t$])
interpret *t*: *wo-rel* t **by** *unfold-locales* (*rule* t)
interpret *rst*: *wo-rel2* $r\ s + o\ t$ **by** *unfold-locales* (*rule* r , *rule* *osum-Well-order*[*OF* $s\ t$])
from *assms*(1) **have** *: *st.isMaxim* $\{a \in Field\ (s + o\ t). case-sum\ f1\ g1\ a \neq case-sum\ f2\ g2\ a\}$ (*Inr* x)
using *rst.isMaxim-max-fun-diff*[*OF* *assms*(2–4)] **by** *simp*
hence *t.isMaxim* $\{a \in Field\ t. g1\ a \neq g2\ a\}$ x
unfolding *st.isMaxim-def* *t.isMaxim-def* *Field-osum* **by** (*auto* *simp*: *osum-def*)
thus $?P\ ?Q$ **unfolding** *t.max-fun-diff-def* *fun-eq-iff*
by (*auto* *intro*: *t.maxim-equality* *simp*: *t.isMaxim-def*)
qed

```

lemma oexp-osum:  $r \hat{o} (s + o t) = o (r \hat{o} s) * o (r \hat{o} t)$  (is  $?R = o ?L$ )
proof (rule ordIso-symmetric)
  interpret rst: wo-rel2  $r s + o t$  by unfold-locales (rule  $r$ , rule osum-Well-order[ $OF$ 
 $s t$ ])
  interpret rs: wo-rel2  $r s$  by unfold-locales (rule  $r$ , rule  $s$ )
  interpret rt: wo-rel2  $r t$  by unfold-locales (rule  $r$ , rule  $t$ )
  let  $?f = \lambda(f, g). \text{case-sum } f g$ 
  have bij-betw  $?f$  (Field  $?L$ ) (Field  $?R$ )
    unfolding bij-betw-def rst.Field-oexp rs.Field-oexp rt.Field-oexp Field-oprod
proof (intro conjI)
  show inj-on  $?f$  (FinFunc  $r s \times \text{FinFunc } r t$ ) unfolding inj-on-def
    by (auto simp: fun-eq-iff split: sum.splits)
  show  $?f ' (FinFunc r s \times \text{FinFunc } r t) = \text{FinFunc } r (s + o t)$ 
proof safe
  fix  $fg$  assume  $fg \in \text{FinFunc } r (s + o t)$ 
  thus  $fg \in ?f ' (FinFunc r s \times \text{FinFunc } r t)$ 
    by (intro image-eqI[of - - (fg o Inl, fg o Inr)])
    (auto simp: FinFunc-osum fun-eq-iff split: sum.splits)
  qed (auto simp: FinFunc-osum o-def)
qed
moreover have compat  $?L ?R ?f$ 
  unfolding compat-def rst.Field-oexp rs.Field-oexp rt.Field-oexp oprod-def
  unfolding rst.oexp-def Let-def rs.oexp-def rt.oexp-def
  by (fastforce simp: Field-osum FinFunc-osum o-def split: sum.splits
    dest: max-fun-diff-eq-Inl max-fun-diff-eq-Inr)
  ultimately have iso  $?L ?R ?f$  using  $r s t$ 
  by (subst iso-iff3) (auto intro: oexp-Well-order oprod-Well-order osum-Well-order)
  thus  $?L = o ?R$  using  $r s t$  unfolding ordIso-def
  by (auto intro: oexp-Well-order oprod-Well-order osum-Well-order)
qed

definition rev-curr  $f b = (\text{if } b \in \text{Field } t \text{ then } \lambda a. f (a, b) \text{ else undefined})$ 

lemma rev-curr-FinFunc:
  assumes Field: Field  $r \neq \{\}$ 
  shows rev-curr ' ( $\text{FinFunc } r (s * o t) = \text{FinFunc } (r \hat{o} s) t$ )
proof safe
  interpret rs: wo-rel2  $r s$  by unfold-locales (rule  $r$ , rule  $s$ )
  interpret rst: wo-rel2  $r \hat{o} s t$  by unfold-locales (rule oexp-Well-order[ $OF r s$ ],
 $rule t$ )
  fix  $g$  assume  $g: g \in \text{FinFunc } r (s * o t)$ 
  hence finite (rst.SUPP (rev-curr  $g$ ))  $\forall x \in \text{Field } t. \text{finite } (rs.SUPP (\text{rev-curr } g$ 
 $x))$ 
  unfolding FinFunc-def Field-oprod rs.Field-oexp Func-def fin-support-def sup-
port-def
   $rs.zero-oexp[OF \text{Field}]$  rev-curr-def by (auto simp: fun-eq-iff rs.const-def elim!:
finite-surj)
  with  $g$  show rev-curr  $g \in \text{FinFunc } (r \hat{o} s) t$ 

```

```

    unfolding FinFunc-def Field-oprod rs.Field-oexp Func-def
    by (auto simp: rev-curr-def fin-support-def)
next
  interpret rs: wo-rel2 r s by unfold-locales (rule r, rule s)
  interpret rst: wo-rel2 r  $\hat{o}$  s t by unfold-locales (rule oexp-Well-order[OF r s],
rule t)
  fix fg assume *: fg  $\in$  FinFunc (r  $\hat{o}$  s) t
  let ?g =  $\lambda(a, b).$  if (a, b)  $\in$  Field (s *o t) then fg b a else undefined
  show fg  $\in$  rev-curr ' FinFunc r (s *o t)
  proof (rule image-eqI[of - - ?g])
    show fg = rev-curr ?g
  proof
    fix x
    from * show fg x = rev-curr ?g x
    unfolding FinFunc-def rs.Field-oexp Func-def rev-curr-def Field-oprod by
auto
  qed
next
  have **: ( $\bigcup g \in fg$  ' Field t. rs.SUPP g) =
    ( $\bigcup g \in fg$  ' Field t - {rs.const}. rs.SUPP g)
    unfolding support-def by auto
  from * have ***:  $\forall g \in fg$  ' Field t. finite (rs.SUPP g) finite (rst.SUPP fg)
  unfolding rs.Field-oexp FinFunc-def Func-def fin-support-def Option.these-def
by force+
  hence finite (fg ' Field t - {rs.const}) using *
    unfolding support-def rs.zero-oexp[OF Field] FinFunc-def Func-def
    by (elim finite-surj[of - - fg]) (fastforce simp: image-iff Option.these-def)
  with *** have finite (( $\bigcup g \in fg$  ' Field t. rs.SUPP g)  $\times$  rst.SUPP fg)
    by (subst **) (auto intro!: finite-cartesian-product)
  with * show ?g  $\in$  FinFunc r (s *o t)
    unfolding Field-oprod rs.Field-oexp FinFunc-def Func-def fin-support-def
Option.these-def
    support-def rs.zero-oexp[OF Field] by (auto elim!: finite-subset[rotated])
  qed
qed

lemma rev-curr-app-FinFunc[elim!]:
   $\llbracket f \in \text{FinFunc } r \text{ (s *o t)}; z \in \text{Field } t \rrbracket \implies \text{rev-curr } f \ z \in \text{FinFunc } r \ s$ 
  unfolding rev-curr-def FinFunc-def Func-def Field-oprod fin-support-def sup-
port-def
  by (auto elim: finite-surj)

lemma max-fun-diff-oprod:
  assumes Field: Field r  $\neq$  {} and f  $\neq$  g f  $\in$  FinFunc r (s *o t) g  $\in$  FinFunc r
(s *o t)
  defines m  $\equiv$  wo-rel.max-fun-diff t (rev-curr f) (rev-curr g)
  shows wo-rel.max-fun-diff (s *o t) f g =
    (wo-rel.max-fun-diff s (rev-curr f m) (rev-curr g m), m)
  proof -

```

```

interpret st: wo-rel s *o t by unfold-locales (rule oprod-Well-order[OF s t])
interpret s: wo-rel s by unfold-locales (rule s)
interpret t: wo-rel t by unfold-locales (rule t)
interpret r-st: wo-rel2 r s *o t by unfold-locales (rule r, rule oprod-Well-order[OF
s t])
interpret rs: wo-rel2 r s by unfold-locales (rule r, rule s)
interpret rst: wo-rel2 r  $\hat{o}$  s t by unfold-locales (rule oexp-Well-order[OF r s],
rule t)
from fun-unequal-in-support[OF assms(2), of Field (s *o t) Field r Field r]
assms(3,4)
have diff1: rev-curr f  $\neq$  rev-curr g
  rev-curr f  $\in$  FinFunc (r  $\hat{o}$  s) t rev-curr g  $\in$  FinFunc (r  $\hat{o}$  s) t using
rev-curr-FinFunc[OF Field]
  unfolding fun-eq-iff rev-curr-def[abs-def] FinFunc-def support-def Field-oprod
  by auto fast
hence diff2: rev-curr f m  $\neq$  rev-curr g m rev-curr f m  $\in$  FinFunc r s rev-curr g
m  $\in$  FinFunc r s
  using rst.max-fun-diff[OF diff1] assms(3,4) rst.max-fun-diff-in unfolding
m-def by auto
show ?thesis unfolding st.max-fun-diff-def
proof (intro st.maxim-equality, unfold st.isMaxim-def Field-oprod, safe)
  show s.max-fun-diff (rev-curr f m) (rev-curr g m)  $\in$  Field s
    using rs.max-fun-diff-in[OF diff2] by auto
  next
    show m  $\in$  Field t using rst.max-fun-diff-in[OF diff1] unfolding m-def by
auto
  next
    assume f (s.max-fun-diff (rev-curr f m) (rev-curr g m), m) =
      g (s.max-fun-diff (rev-curr f m) (rev-curr g m), m)
      (is f (?x, m) = g (?x, m))
    hence rev-curr f m ?x = rev-curr g m ?x unfolding rev-curr-def by auto
    with rs.max-fun-diff-in[OF diff2] show False by auto
  next
    fix x y assume f (x, y)  $\neq$  g (x, y) x  $\in$  Field s y  $\in$  Field t
    thus ((x, y), (s.max-fun-diff (rev-curr f m) (rev-curr g m), m))  $\in$  s *o t
      using rst.max-fun-diff-in[OF diff1] rs.max-fun-diff-in[OF diff2] diff1 diff2
      rst.max-fun-diff-max[OF diff1, of y] rs.max-fun-diff-le-eq[OF - diff2, of x]
      unfolding oprod-def m-def rev-curr-def fun-eq-iff by (auto intro: s.in-notinI)
    qed
  qed
qed

lemma oexp-oexp: (r  $\hat{o}$  s)  $\hat{o}$  t =o r  $\hat{o}$  (s *o t) (is ?R =o ?L)
proof (cases r = {})
case True
  interpret rs: wo-rel2 r s by unfold-locales (rule r, rule s)
  interpret rst: wo-rel2 r  $\hat{o}$  s t by unfold-locales (rule oexp-Well-order[OF r s],
rule t)
  show ?thesis
  proof (cases s = {}  $\vee$  t = {})

```

```

case True with  $\langle r = \{\} \rangle$  show ?thesis
  by (auto simp: oexp-empty [OF oexp-Well-order [OF Well-order-empty s]]
    intro!: ordIso-transitive [OF ordIso-symmetric [OF oone-ordIso] oone-ordIso]
    ordIso-transitive [OF oone-ordIso-oexp [OF ordIso-symmetric [OF oone-ordIso]
t] oone-ordIso])
next
  case False
  hence  $s * o \ t \neq \{\}$  unfolding oprod-def Field-def by fastforce
  with False show ?thesis
    using  $\langle r = \{\} \rangle$  ozero-ordIso
    by (auto simp add: s t oprod-Well-order ozero-def)
qed
next
  case False
  hence Field: Field  $r \neq \{\}$  by (metis Field-def Range-empty-iff Un-empty)
  show ?thesis
  proof (rule ordIso-symmetric)
    interpret r-st: wo-rel2  $r \ s * o \ t$  by unfold-locals (rule r, rule oprod-Well-order [OF s t])
    interpret rs: wo-rel2  $r \ s$  by unfold-locals (rule r, rule s)
    interpret rst: wo-rel2  $r \ \hat{o} \ s \ t$  by unfold-locals (rule oexp-Well-order [OF r s],
rule t)
    have bij: bij-betw rev-curr (Field ?L) (Field ?R)
    unfolding bij-betw-def r-st.Field-oexp rst.Field-oexp Field-oprod proof (intro conjI)
      show inj-on rev-curr (FinFunc  $r \ (s * o \ t)$ )
      unfolding inj-on-def FinFunc-def Func-def Field-oprod rs.Field-oexp rev-curr-def [abs-def]
      by (auto simp: fun-eq-iff) metis
      show rev-curr ‘ (FinFunc  $r \ (s * o \ t)$ ) = FinFunc ( $r \ \hat{o} \ s$ )  $t$  by (rule rev-curr-FinFunc [OF Field])
    qed
    moreover
    have compat ?L ?R rev-curr
    unfolding compat-def proof safe
      fix fg1 fg2 assume fg: (fg1, fg2) ∈ r  $\hat{o}$  (s * o t)
      show (rev-curr fg1, rev-curr fg2)  $\in r \ \hat{o} \ s \ \hat{o} \ t$ 
      proof (cases fg1 = fg2)
        assume  $fg1 \neq fg2$ 
        with fg show ?thesis
          using rst.max-fun-diff-in [of rev-curr fg1 rev-curr fg2]
            max-fun-diff-oprod [OF Field, of fg1 fg2] rev-curr-FinFunc [OF Field, symmetric]
          unfolding r-st.Field-oexp rs.Field-oexp rst.Field-oexp unfolding r-st.oexp-def rst.oexp-def
          by (auto simp: rs.oexp-def Let-def) (auto simp: rev-curr-def [abs-def])
        next
          assume  $fg1 = fg2$ 
          with fg bij show ?thesis unfolding r-st.Field-oexp rs.Field-oexp rst.Field-oexp bij-betw-def

```

```

      by (auto simp: r-st.oexp-def rst.oexp-def)
    qed
  qed
  ultimately have iso ?L ?R rev-curr using r s t
    by (subst iso-iff3) (auto intro: oexp-Well-order oprod-Well-order)
  thus ?L =o ?R using r s t unfolding ordIso-def
    by (auto intro: oexp-Well-order oprod-Well-order)
  qed
qed

end

end

```

10 Cardinal-Order Relations

```

theory Cardinal-Order-Relation
  imports Wellorder-Constructions
begin

```

```

declare
  card-order-on-well-order-on[simp]
  card-of-card-order-on[simp]
  card-of-well-order-on[simp]
  Field-card-of[simp]
  card-of-Card-order[simp]
  card-of-Well-order[simp]
  card-of-least[simp]
  card-of-unique[simp]
  card-of-mono1[simp]
  card-of-mono2[simp]
  card-of-cong[simp]
  card-of-Field-ordIso[simp]
  card-of-underS[simp]
  ordLess-Field[simp]
  card-of-empty[simp]
  card-of-empty1[simp]
  card-of-image[simp]
  card-of-singl-ordLeq[simp]
  Card-order-singl-ordLeq[simp]
  card-of-Pow[simp]
  Card-order-Pow[simp]
  card-of-Plus1[simp]
  Card-order-Plus1[simp]
  card-of-Plus2[simp]
  Card-order-Plus2[simp]
  card-of-Plus-mono1[simp]
  card-of-Plus-mono2[simp]
  card-of-Plus-mono[simp]

```

card-of-Plus-cong2[simp]
card-of-Plus-cong[simp]
card-of-Un-Plus-ordLeq[simp]
card-of-Times1[simp]
card-of-Times2[simp]
card-of-Times3[simp]
card-of-Times-mono1[simp]
card-of-Times-mono2[simp]
card-of-ordIso-finite[simp]
card-of-Times-same-infinite[simp]
card-of-Times-infinite-simps[simp]
card-of-Plus-infinite1[simp]
card-of-Plus-infinite2[simp]
card-of-Plus-ordLess-infinite[simp]
card-of-Plus-ordLess-infinite-Field[simp]
infinite-cartesian-product[simp]
cardSuc-Card-order[simp]
cardSuc-greater[simp]
cardSuc-ordLeq[simp]
cardSuc-ordLeq-ordLess[simp]
cardSuc-mono-ordLeq[simp]
cardSuc-invar-ordIso[simp]
card-of-cardSuc-finite[simp]
cardSuc-finite[simp]
card-of-Plus-ordLeq-infinite-Field[simp]
curr-in[intro, simp]

10.1 Cardinal of a set

lemma *card-of-inj-rel*: **assumes** *INJ*: $\bigwedge x y y'. \llbracket (x, y) \in R; (x, y') \in R \rrbracket \implies y = y'$
shows $|\{y. \exists x. (x, y) \in R\}| \leq_o |\{x. \exists y. (x, y) \in R\}|$

proof –

let $?Y = \{y. \exists x. (x, y) \in R\}$ **let** $?X = \{x. \exists y. (x, y) \in R\}$
let $?f = \lambda y. \text{SOME } x. (x, y) \in R$
have $?f ' ?Y \leq ?X$ **by** (*auto dest: someI*)
moreover have *inj-on* $?f ?Y$
by (*metis (mono-tags, lifting) assms inj-onI mem-Collect-eq*)
ultimately show $|\{Y\}| \leq_o |\{X\}|$ **using** *card-of-ordLeq* **by** *blast*
qed

lemma *card-of-unique2*: $\llbracket \text{card-order-on } B \text{ } r; \text{bij-betw } f \text{ } A \text{ } B \rrbracket \implies r =_o |A|$
using *card-of-ordIso* *card-of-unique* *ordIso-equivalence* **by** *blast*

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 *Card-order-omax*:
assumes *finite R* **and** $R \neq \{\}$ **and** $\forall r \in R. \text{Card-order } r$

shows *Card-order* (*omax* *R*)
by (*simp add: assms omax-in*)

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 $\forall r \in ?R. \text{Card-order } r$
using *card-of-Card-order* **by** *auto*
 ultimately show *?thesis* **by**(*rule Card-order-omax*)
qed

10.2 Cardinals versus set operations on arbitrary sets

lemma *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-Un1[simp]*: $|A| \leq_o |A \cup B|$
by *fastforce*

lemma *card-of-diff[simp]*: $|A - B| \leq_o |A|$
by *fastforce*

lemma *subset-ordLeq-strict*:
 assumes $A \leq B$ and $|A| <_o |B|$
 shows $A < B$
using *assms ordLess-irreflexive* **by** *blast*

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-empty4*:
 $|\{\}::'b\ set| <_o |A::'a\ set| = (A \neq \{\})$
by (*metis card-of-empty card-of-ordLess2 image-is-empty not-ordLess-ordLeq*)

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

lemma *Well-order-card-of-empty*:
 $\text{Well-order } r \implies |\{\}| \leq_o r$
by *simp*

lemma *card-of-UNIV[simp]*:
 $|A::'a\ set| \leq_o |UNIV::'a\ set|$

by *simp*

lemma *card-of-UNIV2[simp]*:
Card-order $r \implies (r :: 'a \text{ rel}) \leq_o |UNIV :: 'a \text{ set}|$
using *card-of-UNIV[of Field r]* *card-of-Field-ordIso*
ordIso-symmetric ordIso-ordLeq-trans **by** *blast*

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 $inj\text{-}on\ (image\ f)\ (Pow\ A) \wedge (image\ f)\ 'A \leq (Pow\ B)$
by (*auto simp: inj-on-image-Pow image-Pow-mono*)
thus *?thesis* **using** *card-of-ordLeq[of Pow A]* **by** *metis*
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 $|Pow\ A| =_o |Pow\ B|$
by (*meson assms card-of-Pow-mono ordIso-iff-ordLeq*)

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*

corollary *Card-order-Plus-empty1*:
Card-order $r \implies r =_o |(Field\ r) <+> \{\}|$
using *card-of-Plus-empty1 card-of-Field-ordIso ordIso-equivalence* **by** *blast*

corollary *Card-order-Plus-empty2*:
Card-order $r \implies r =_o |\{\}| <+> (Field\ r)|$
using *card-of-Plus-empty2 card-of-Field-ordIso ordIso-equivalence* **by** *blast*

lemma *card-of-Un2[simp]*:
shows $|A| \leq_o |B \cup A|$
by *fastforce*

lemma *Un-Plus-bij-betw*:
assumes $A\ Int\ B = \{\}$
shows $\exists f. \text{bij-betw}\ f\ (A \cup B)\ (A <+> B)$
proof –

have *bij-betw* ($\lambda c. \text{if } c \in A \text{ then } \text{Inl } c \text{ else } \text{Inr } c$) ($A \cup B$) ($A <+> B$)
using *assms unfolding bij-betw-def inj-on-def* **by** *auto*
thus *?thesis* **by** *blast*
qed

lemma *card-of-Un-Plus-ordIso*:
assumes $A \text{ Int } B = \{\}$
shows $|A \cup B| =_o |A <+> B|$
by (*meson Un-Plus-bij-betw assms card-of-ordIso*)

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-Times-singl1*: $|A| =_o |A \times \{b\}|$
proof –
have *bij-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*:
 $\text{Card-order } r \implies r =_o |(Field\ r) \times \{b\}|$
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-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*:
 $\text{Card-order } r \implies r =_o |\{a\} \times (Field\ r)|$
using *card-of-Times-singl2* [*of - a*] *card-of-Field-ordIso ordIso-equivalence* **by** *blast*

lemma *card-of-Times-assoc*: $|(A \times B) \times C| =_o |A \times B \times C|$
proof –
let $?f = \lambda((a,b),c). (a,(b,c))$
have $A \times B \times C \subseteq ?f ' ((A \times B) \times C)$
proof
fix x **assume** $x \in A \times B \times C$
then obtain $a\ b\ c$ **where** $*$: $a \in A\ b \in B\ c \in C\ x = (a, b, c)$ **by** *blast*
let $?x = ((a, b), c)$
from $*$ **have** $?x \in (A \times B) \times C\ x = ?f\ ?x$ **by** *auto*
thus $x \in ?f ' ((A \times B) \times C)$ **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-Times-cong1*[simp]:
 assumes $|A| =_o |B|$
 shows $|A \times C| =_o |B \times C|$
 using *assms* by (simp add: *ordIso-iff-ordLeq*)

lemma *card-of-Times-cong2*[simp]:
 assumes $|A| =_o |B|$
 shows $|C \times A| =_o |C \times B|$
 using *assms* by (simp add: *ordIso-iff-ordLeq*)

lemma *card-of-Times-mono*[simp]:
 assumes $|A| \leq_o |B|$ and $|C| \leq_o |D|$
 shows $|A \times C| \leq_o |B \times D|$
 using *assms card-of-Times-mono1*[of A B C] *card-of-Times-mono2*[of C D B]
ordLeq-transitive[of |A × C|] by blast

corollary *ordLeq-Times-mono*:
 assumes $r \leq_o r'$ and $p \leq_o p'$
 shows $|(Field\ r) \times (Field\ p)| \leq_o |(Field\ r') \times (Field\ p')|$
 using *assms card-of-mono2*[of r r'] *card-of-mono2*[of p p'] *card-of-Times-mono*
 by blast

corollary *ordIso-Times-cong1*[simp]:
 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

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: *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-mono2*:
 assumes *inj-on* f ($I :: 'i \text{ set}$) and $f ' I \leq (J :: 'j \text{ set})$
 shows $|\text{SIGMA } i : I. (A :: 'j \Rightarrow 'a \text{ set}) (f i)| \leq_o |\text{SIGMA } j : J. A j|$
proof –
 let $?LEFT = \text{SIGMA } i : I. A (f i)$
 let $?RIGHT = \text{SIGMA } j : J. A j$
 obtain u where $u\text{-def}$: $u = (\lambda(i :: 'i, a :: 'a). (f i, a))$ by *blast*
 have $\text{inj-on } u \text{ ?LEFT} \wedge u ' ?LEFT \leq ?RIGHT$
 using *assms* **unfolding** $u\text{-def}$ *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 $|\text{SIGMA } i : I. A (f i)| \leq_o |\text{SIGMA } j : J. B j|$
proof –
 have $\forall i \in I. |A(f i)| \leq_o |B(f i)|$
 using *IM LEQ* by *blast*
 hence $|\text{SIGMA } i : I. A (f i)| \leq_o |\text{SIGMA } i : I. B (f i)|$
 using *card-of-Sigma-mono1* [of I] by *metis*
 moreover have $|\text{SIGMA } i : I. B (f i)| \leq_o |\text{SIGMA } j : J. B j|$
 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 $|\text{SIGMA } i : I. \text{Field}(p i)| \leq_o |\text{SIGMA } i : I. \text{Field}(r i)|$
 using *assms* by (*auto simp*: *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 $|\text{SIGMA } i : I. \text{Field}(p(f i))| \leq_o |\text{SIGMA } j : J. \text{Field}(r j)|$
 using *assms card-of-mono2 card-of-Sigma-mono* [of $f I J \lambda i. \text{Field}(p i)$] by *metis*

lemma *ordIso-Sigma-cong1*:
 assumes $\forall i \in I. p i =_o r i$
 shows $|\text{SIGMA } i : I. \text{Field}(p i)| =_o |\text{SIGMA } i : I. \text{Field}(r i)|$
 using *assms* by (*auto simp*: *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 $|\text{SIGMA } i : I. \text{Field}(p(f i))| =_o |\text{SIGMA } j : J. \text{Field}(r j)|$
 using *assms card-of-cong card-of-Sigma-cong*
 [of $f I J \lambda j. \text{Field}(p j) \lambda j. \text{Field}(r j)$] by *blast*

lemma *card-of-UNION-Sigma2*:
 assumes $\bigwedge i j. \llbracket \{i,j\} \leq I; i \neq j \rrbracket \implies A \ i \ \text{Int} \ A \ j = \{\}$
 shows $|\bigcup_{i \in I}. A \ i| =_o |\text{Sigma } I \ A|$
proof –
 let $?L = \bigcup_{i \in I}. A \ i$ let $?R = \text{Sigma } I \ A$
 have $|?L| \leq_o |?R|$ **using** *card-of-UNION-Sigma* .
 moreover have $|?R| \leq_o |?L|$
proof –
 have *inj-on snd* $?R$
 unfolding *inj-on-def* **using** *assms* **by** *auto*
 moreover have *snd* ‘ $?R \leq ?L$ **by** *auto*
 ultimately show *?thesis* **using** *card-of-ordLeq* **by** *blast*
qed
 ultimately show *?thesis* **by** (*simp add: ordIso-iff-ordLeq*)
qed

corollary *Plus-into-Times*:
 assumes $A2: a1 \neq a2 \wedge \{a1, a2\} \leq A$ **and** $B2: b1 \neq b2 \wedge \{b1, b2\} \leq B$
 shows $\exists f. \text{inj-on } f \ (A <+> B) \wedge f \text{ ‘ } (A <+> B) \leq A \times B$
using *assms* **by** (*auto simp: 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 $\exists (f::'a + 'b \Rightarrow 'a * 'b). \text{inj } f$
using *assms* *Plus-into-Times*[*of* $a1 \ a2 \ \text{UNIV } b1 \ b2 \ \text{UNIV}$]
by *auto*

corollary *Times-same-infinite-bij-betw*:
 assumes $\neg \text{finite } A$
 shows $\exists f. \text{bij-betw } f \ (A \times A) \ A$
using *assms* **by** (*auto simp: card-of-ordIso*)

corollary *Times-same-infinite-bij-betw-types*:
 assumes $\text{INF}: \neg \text{finite}(\text{UNIV}::'a \ \text{set})$
 shows $\exists (f::('a * 'a) \Rightarrow 'a). \text{bij } f$
using *assms* *Times-same-infinite-bij-betw*[*of* $\text{UNIV}::'a \ \text{set}$]
by *auto*

corollary *Times-infinite-bij-betw*:
 assumes $\text{INF}: \neg \text{finite } A$ **and** $\text{NE}: B \neq \{\}$ **and** $\text{INJ}: \text{inj-on } g \ B \wedge g \text{ ‘ } B \leq A$
 shows $(\exists f. \text{bij-betw } f \ (A \times B) \ A) \wedge (\exists h. \text{bij-betw } h \ (B \times A) \ A)$
proof –
 have $|B| \leq_o |A|$ **using** *INJ* *card-of-ordLeq* **by** *blast*
 thus *?thesis* **using** *INF* *NE*
by (*auto simp: card-of-ordIso card-of-Times-infinite*)
qed

corollary *Times-infinite-bij-betw-types*:

assumes $\neg \text{finite}(UNIV::'a \text{ set})$ **and** $\text{inj}(g::'b \Rightarrow 'a)$
shows $(\exists (f::('b * 'a) \Rightarrow 'a). \text{bij } f) \wedge (\exists (h::('a * 'b) \Rightarrow 'a). \text{bij } h)$
using *assms Times-infinite-bij-betw*[of $UNIV::'a \text{ set}$ $UNIV::'b \text{ set}$ g]
by *auto*

lemma *card-of-Times-ordLeq-infinite*:

$\llbracket \neg \text{finite } C; |A| \leq_o |C|; |B| \leq_o |C| \rrbracket \Longrightarrow |A \times B| \leq_o |C|$
by (*simp add: card-of-Sigma-ordLeq-infinite*)

corollary *Plus-infinite-bij-betw*:

assumes *INF*: $\neg \text{finite } A$ **and** *INJ*: $\text{inj-on } g \ B \wedge g \text{ ' } B \leq A$
shows $(\exists f. \text{bij-betw } f \ (A <+> B) \ A) \wedge (\exists h. \text{bij-betw } h \ (B <+> A) \ A)$

proof–

have $|B| \leq_o |A|$ **using** *INJ card-of-ordLeq* **by** *blast*
thus *?thesis* **using** *INF*
by (*auto simp: card-of-ordIso*)

qed

corollary *Plus-infinite-bij-betw-types*:

assumes $\neg \text{finite}(UNIV::'a \text{ set})$ **and** $\text{inj}(g::'b \Rightarrow 'a)$
shows $(\exists (f::('b + 'a) \Rightarrow 'a). \text{bij } f) \wedge (\exists (h::('a + 'b) \Rightarrow 'a). \text{bij } h)$
using *assms Plus-infinite-bij-betw*[of $UNIV::'a \text{ set}$ g $UNIV::'b \text{ set}$]
by *auto*

lemma *card-of-Un-infinite*:

assumes *INF*: $\neg \text{finite } A$ **and** *LEQ*: $|B| \leq_o |A|$
shows $|A \cup B| =_o |A| \wedge |B \cup A| =_o |A|$
by (*simp add: INF LEQ card-of-Un-ordLeq-infinite-Field ordIso-iff-ordLeq*)

lemma *card-of-Un-infinite-simps*[*simp*]:

$\llbracket \neg \text{finite } A; |B| \leq_o |A| \rrbracket \Longrightarrow |A \cup B| =_o |A|$
 $\llbracket \neg \text{finite } A; |B| \leq_o |A| \rrbracket \Longrightarrow |B \cup A| =_o |A|$
using *card-of-Un-infinite* **by** *auto*

lemma *card-of-Un-diff-infinite*:

assumes *INF*: $\neg \text{finite } A$ **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 \neg *finite B* by *auto*
 ultimately have *False*
 using *card-of-Un-infinite 1 ordIso-equivalence(1,3) LESS not-ordLess-ordIso*
 by *metis*
 }
 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 \neg *finite C* by *auto*
 hence $|C| =_o |A|$
 using *card-of-Un-infinite 1 2 ordIso-equivalence(1,3)* by *metis*
 thus ?*thesis* unfolding *C-def* .
 qed

corollary *Card-order-Un-infinite*:
 assumes *INF*: \neg *finite(Field r)* and *CARD*: *Card-order r* and
LEQ: $p \leq_o r$
 shows $|(Field\ r) \cup (Field\ p)| =_o r \wedge |(Field\ p) \cup (Field\ r)| =_o r$
proof–
 have $|Field\ r \cup Field\ p| =_o |Field\ r| \wedge$
 $|Field\ p \cup Field\ r| =_o |Field\ r|$
 using *assms* by (*auto simp: card-of-Un-infinite*)
 thus ?*thesis*
 using *assms card-of-Field-ordIso[of r]*
ordIso-transitive[of |Field r \cup Field p|]
ordIso-transitive[of - |Field r|] by *blast*
 qed

corollary *subset-ordLeq-diff-infinite*:
 assumes *INF*: \neg *finite B* and *SUB*: $A \leq B$ and *LESS*: $|A| <_o |B|$
 shows \neg *finite (B - A)*
 using *assms card-of-Un-diff-infinite card-of-ordIso-finite* by *blast*

lemma *card-of-Times-ordLess-infinite[simp]*:
 assumes *INF*: \neg *finite 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: card-of-empty4*)
next
 assume *Case2*: $\neg(A = {} \vee B = {})$


```

{assume *:  $|C| \leq_o |A \times B|$ 
  hence  $\neg \text{finite } (A \times B)$  using INF card-of-ordLeq-finite by blast
  hence  $1: \neg \text{finite } A \vee \neg \text{finite } B$  using finite-cartesian-product by blast
  {assume Case21:  $|A| \leq_o |B|$ 
    hence  $\neg \text{finite } B$  using 1 card-of-ordLeq-finite by blast
    hence  $|A \times B| =_o |B|$  using Case2 Case21
      by (auto simp: card-of-Times-infinite)
    hence False using LESS2 not-ordLess-ordLeq * ordLeq-ordIso-trans by blast
  }
  moreover
  {assume Case22:  $|B| \leq_o |A|$ 
    hence  $\neg \text{finite } A$  using 1 card-of-ordLeq-finite by blast
    hence  $|A \times B| =_o |A|$  using Case2 Case22
      by (auto simp: 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 × B| |C|]
  card-of-Well-order[of A × B] card-of-Well-order[of C] by auto
qed

```

lemma *card-of-Times-ordLess-infinite-Field[simp]*:

assumes *INF*: $\neg \text{finite } (\text{Field } r)$ 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 $|A| <_o |?C| \quad |B| <_o |?C|$

using *LESS1 LESS2 ordLess-ordIso-trans* by blast+

hence $|A \times B| <_o |?C|$ using *INF*

card-of-Times-ordLess-infinite by blast

thus ?thesis using *1 ordLess-ordIso-trans* by blast

qed

lemma *ordLeq-finite-Field*:

assumes $r \leq_o s$ and *finite* (*Field* *s*)

shows *finite* (*Field* *r*)

by (metis *assms card-of-mono2 card-of-ordLeq-infinite*)

lemma *ordIso-finite-Field*:

assumes $r =_o s$

shows *finite* (*Field* *r*) \longleftrightarrow *finite* (*Field* *s*)

by (metis *assms ordIso-iff-ordLeq ordLeq-finite-Field*)

10.3 Cardinals versus set operations involving infinite sets

lemma *finite-iff-cardOf-nat*:

finite A = (|A| < o |UNIV :: nat set|)
by (*meson card-of-Well-order infinite-iff-card-of-nat not-ordLeq-iff-ordLess*)

lemma *finite-ordLess-infinite2[simp]*:

assumes *finite A and ¬finite B*
shows $|A| < o |B|$
by (*meson assms card-of-Well-order card-of-ordLeq-finite not-ordLeq-iff-ordLess*)

lemma *infinite-card-of-insert*:

assumes $\neg \text{finite } A$
shows $|\text{insert } a \ A| = o |A|$

proof–

have *iA: insert a A = A ∪ {a}* **by** *simp*
show *?thesis*
using *infinite-imp-bij-betw2[OF assms]* **unfolding** *iA*
by (*metis bij-betw-inv card-of-ordIso*)

qed

lemma *card-of-Un-singl-ordLess-infinite1*:

assumes $\neg \text{finite } B$ **and** $|A| < o |B|$
shows $|\{a\} \ Un \ A| < o |B|$
by (*metis assms card-of-Un-ordLess-infinite finite.emptyI finite-insert finite-ordLess-infinite2*)

lemma *card-of-Un-singl-ordLess-infinite*:

assumes $\neg \text{finite } B$
shows $|A| < o |B| \longleftrightarrow |\{a\} \ Un \ A| < o |B|$
using *assms card-of-Un-singl-ordLess-infinite1[of B A]*
using *card-of-Un2 ordLeq-ordLess-trans* **by** *blast*

10.4 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 ⇒ nat ⇒ 'a list set*

where *nlists A n ≡ {l. set l ≤ A ∧ length l = n}*

lemma *lists-UNION-nlists*: *lists A = (⋃ n. nlists A n)*

unfolding *lists-eq-set nlists-def* **by** *blast*

lemma *card-of-lists*: $|A| \leq o |\text{lists } A|$

proof–

let *?h = λ a. [a]*
have *inj-on ?h A ∧ ?h ‘ A ≤ lists A*
unfolding *inj-on-def lists-eq-set* **by** *auto*
thus *?thesis* **by** (*metis card-of-ordLeq*)

qed

lemma *nlists-0*: $nlists\ A\ 0 = \{\emptyset\}$

unfolding *nlists-def* **by** *auto*

lemma *nlists-not-empty*:

assumes $A \neq \{\}$

shows $nlists\ A\ n \neq \{\}$

proof (*induction n*)

case (*Suc n*)

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*

then show *?case* **by** *auto*

qed (*simp add: nlists-0*)

lemma *card-of-nlists-Succ*: $|nlists\ A\ (Suc\ n)| =_o |A \times (nlists\ A\ n)|$

proof–

let $?B = A \times (nlists\ A\ n)$ **let** $?h = \lambda(a,l). a \# l$

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 *clarify*

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** $2: l = a \# l'$ **by** (*auto simp: length-Suc-conv*)

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 $\neg finite\ A$

shows $|nlists\ A\ n| \leq_o |A|$

proof(*induction n*)

case 0

have $A \neq \{\}$ **using** *assms* **by** *auto*

then show *?case*

by (*simp add: nlists-0*)

next

case (*Suc n*)

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 *Suc assms* **by** *auto*

ultimately show *?case*

using *ordIso-transitive ordIso-iff-ordLeq* by *blast*
qed

lemma *Card-order-lists*: $\text{Card-order } r \implies r \leq_o |\text{lists}(\text{Field } r)|$
using *card-of-lists card-of-Field-ordIso ordIso-ordLeq-trans ordIso-symmetric* by *blast*

lemma *Union-set-lists*: $\bigcup (\text{set } ' (\text{lists } A)) = A$
proof –
{ **fix** *a* **assume** $a \in A$
hence $\text{set } [a] \leq A \wedge a \in \text{set } [a]$ by *auto*
hence $\exists l. \text{set } l \leq A \wedge a \in \text{set } l$ by *blast* }
then show *?thesis* by *force*
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 $(\text{map } f) ' (\text{lists } A) \leq \text{lists } B$
using *assms* by *force*

lemma *map-lists-surjective*:
assumes $f ' A = B$
shows $(\text{map } f) ' (\text{lists } A) = \text{lists } B$
by (*metis assms lists-image*)

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: inj-on-map-lists map-lists-surjective*)

lemma *card-of-lists-mono[simp]*:
assumes $|A| \leq_o |B|$
shows $|\text{lists } A| \leq_o |\text{lists } B|$
proof –
obtain *f* **where** $\text{inj-on } f A \wedge f ' A \leq B$
using *assms card-of-ordLeq[of A B]* by *auto*
hence $\text{inj-on } (\text{map } f) (\text{lists } A) \wedge (\text{map } f) ' (\text{lists } A) \leq (\text{lists } B)$
by (*auto simp: inj-on-map-lists map-lists-mono*)
thus *?thesis* **using** *card-of-ordLeq[of lists A]* by *metis*
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  $|lists\ A| =_o |lists\ B|$ 
  by (meson assms card-of-lists-mono ordIso-iff-ordLeq)

lemma card-of-lists-infinite[simp]:
  assumes  $\neg finite\ A$ 
  shows  $|lists\ A| =_o |A|$ 
proof –
  have  $|lists\ A| \leq_o |A|$  unfolding lists-UNION-nlists
  by (rule card-of-UNION-ordLeq-infinite[OF assms - ballI[OF card-of-nlists-infinite[OF assms]]])
  (metis infinite-iff-card-of-nat assms)
  thus ?thesis using card-of-lists ordIso-iff-ordLeq by blast
qed

lemma Card-order-lists-infinite:
  assumes Card-order  $r$  and  $\neg finite(Field\ r)$ 
  shows  $|lists(Field\ r)| =_o r$ 
  using assms card-of-lists-infinite card-of-Field-ordIso ordIso-transitive by blast

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

corollary lists-infinite-bij-betw:
  assumes  $\neg finite\ A$ 
  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  $\neg finite(UNIV :: 'a\ set)$ 
  shows  $\exists (f :: 'a\ list \Rightarrow 'a). bij\ f$ 
  using assms lists-infinite-bij-betw by fastforce

```

10.5 Cardinals versus the finite powerset operator

```

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 metis
qed

```

lemma *Card-order-Fpow*: *Card-order* $r \implies r \leq_o |Fpow(Field\ r)|$
using *card-of-Fpow card-of-Field-ordIso ordIso-ordLeq-trans ordIso-symmetric* **by**
blast

lemma *image-Fpow-surjective*:
assumes $f : A = B$
shows $(image\ f) : (Fpow\ A) = Fpow\ B$
proof –
have $\bigwedge C. \llbracket C \subseteq f : A; finite\ C \rrbracket \implies C \in (\cdot) f : \{X. X \subseteq A \wedge finite\ X\}$
by (*simp add: finite-subset-image image-iff*)
then show *?thesis*
using *assms* **by** (*force simp add: Fpow-def*)
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: inj-on-image-Fpow image-Fpow-surjective*)

lemma *card-of-Fpow-mono[simp]*:
assumes $|A| \leq_o |B|$
shows $|Fpow\ A| \leq_o |Fpow\ B|$
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* $(image\ f)\ (Fpow\ A) \wedge (image\ f) : (Fpow\ A) \leq (Fpow\ B)$
by (*auto simp: inj-on-image-Fpow image-Fpow-mono*)
thus *?thesis* **using** *card-of-ordLeq[of Fpow A]* **by** *auto*
qed

lemma *ordIso-Fpow-mono*:
assumes $r \leq_o r'$
shows $|Fpow(Field\ r)| \leq_o |Fpow(Field\ r')|$
using *assms card-of-mono2 card-of-Fpow-mono* **by** *blast*

lemma *card-of-Fpow-cong[simp]*:
assumes $|A| =_o |B|$
shows $|Fpow\ A| =_o |Fpow\ B|$
by (*meson assms card-of-Fpow-mono ordIso-iff-ordLeq*)

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-eq-set 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 $\neg \text{finite } A$
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 $\neg \text{finite } A$
shows $\exists f. \text{bij-betw } f\ (Fpow\ A)\ A$
using *assms card-of-Fpow-infinite card-of-ordIso* **by** *blast*

10.6 The cardinal ω and the finite cardinals

10.6.1 First as well-orders

lemma *Field-natLess*: *Field natLess* = (*UNIV::nat set*)
by (*auto simp: Field-def natLess-def*)

lemma *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}*
proof –
have $\forall t < n. t \in \text{Field } \text{natLeq}$
by (*simp add: Field-natLeq*)
moreover have $\forall x < n. \forall t. (t, x) \in \text{natLeq} \longrightarrow t < n$
by (*simp add: natLeq-def*)
ultimately show *?thesis*
by (*auto simp: natLeq-wo-rel wo-rel.ofilter-def under-def*)
qed

lemma *natLeq-ofilter-leq*: *ofilter natLeq {0 .. n}*
by (*metis (no-types) atLeastLessThanSuc-atLeastAtMost natLeq-ofilter-less*)

lemma *natLeq-UNIV-ofilter*: *wo-rel.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 $A = \text{UNIV} \vee (\exists n. A = \{0 \dots n\})$
proof –
{assume $A \neq \text{UNIV}$ **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*
 then have $A = \{0 ..< n\}$
 proof(*auto simp: 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 $\forall m\ n. n \in A \wedge m \leq n \longrightarrow m \in A$ using *natLeq-wo-rel*
 by(*auto simp: natLeq-def wo-rel.ofilter-def under-def*)
 thus $A = UNIV \vee (\exists n. A = \{0 ..< n\})$ using *closed-nat-set-iff* by *blast*
 next
 assume $A = UNIV \vee (\exists n. A = \{0 ..< n\})$
 thus *ofilter natLeq A*
 by(*auto simp: natLeq-ofilter-less natLeq-UNIV-ofilter*)
 qed

lemma *natLeq-under-leq*: $under\ natLeq\ n = \{0 .. n\}$
 unfolding *under-def natLeq-def* by *auto*

lemma *natLeq-on-ofilter-less-eq*:
 $n \leq m \implies wo-rel.ofilter\ (natLeq-on\ m)\ \{0 ..< n\}$
 by (*auto simp: natLeq-on-wo-rel wo-rel.ofilter-def Field-natLeq-on under-def*)

lemma *natLeq-on-ofilter-iff*:
 $wo-rel.ofilter\ (natLeq-on\ m)\ A = (\exists n \leq m. A = \{0 ..< n\})$
proof(*rule iffI*)
 assume *: *wo-rel.ofilter (natLeq-on m) A*
 hence $1: A \leq \{0 ..< m\}$
 by (*auto simp: natLeq-on-wo-rel wo-rel.ofilter-def under-def Field-natLeq-on*)
 hence $\forall n1\ n2. n2 \in A \wedge n1 \leq n2 \longrightarrow n1 \in A$
 using * by(*fastforce simp add: natLeq-on-wo-rel wo-rel.ofilter-def under-def*)
 hence $A = UNIV \vee (\exists n. A = \{0 ..< n\})$ 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 *wo-rel.ofilter (natLeq-on m) A* by (*auto simp: natLeq-on-ofilter-less-eq*)

qed

corollary *natLeq-on-ofilter:*

ofilter(natLeq-on n) {0 ..< n}
by (*auto simp: natLeq-on-ofilter-less-eq*)

lemma *natLeq-on-ofilter-less:*

assumes $n < m$ **shows** *ofilter (natLeq-on m) {0 .. n}*

proof –

have $Suc\ n \leq m$
using *assms* **by** *simp*
then show *?thesis*
using *natLeq-on-ofilter-iff* **by** *auto*

qed

lemma *natLeq-on-ordLess-natLeq: natLeq-on n <= natLeq*

proof –

have *well-order natLeq*
using *Field-natLeq natLeq-Well-order* **by** *auto*
moreover have $\bigwedge n. \text{well-order-on } \{na. na < n\} \text{ (natLeq-on } n)$
using *Field-natLeq-on natLeq-on-Well-order* **by** *presburger*
ultimately show *?thesis*
by (*simp add: Field-natLeq Field-natLeq-on finite-ordLess-infinite*)

qed

lemma *natLeq-on-injective:*

$\text{natLeq-on } m = \text{natLeq-on } n \implies m = n$
using *Field-natLeq-on[of m] Field-natLeq-on[of n]*
atLeastLessThan-injective[of m n, unfolded atLeastLessThan-def] **by** *blast*

lemma *natLeq-on-injective-ordIso:*

$(\text{natLeq-on } m =_o \text{natLeq-on } n) = (m = n)$

proof(*auto simp: natLeq-on-Well-order ordIso-reflexive*)

assume $\text{natLeq-on } m =_o \text{natLeq-on } n$

then obtain *f* **where** *bij-betw f {x. x < m} {x. x < n}*

using *Field-natLeq-on unfolding ordIso-def iso-def[abs-def]* **by** *auto*

thus $m = n$ **using** *atLeastLessThan-injective2[of f m n]*

unfolding *atLeast-0 atLeastLessThan-def lessThan-def Int-UNIV-left* **by** *blast*

qed

10.6.2 Then as cardinals

lemma *ordIso-natLeq-infinite1:*

$|A| =_o \text{natLeq} \implies \neg \text{finite } A$

by (*meson finite-iff-ordLess-natLeq not-ordLess-ordIso*)

lemma *ordIso-natLeq-infinite2:*

$\text{natLeq} =_o |A| \implies \neg \text{finite } A$

using *ordIso-imp-ordLeq infinite-iff-natLeq-ordLeq* **by** *blast*

lemma *ordIso-natLeq-on-imp-finite*:
 $|A| =_o \text{natLeq-on } n \implies \text{finite } A$
unfolding *ordIso-def iso-def[abs-def]*
by (*auto simp: Field-natLeq-on bij-betw-finite*)

lemma *natLeq-on-Card-order*: *Card-order (natLeq-on n)*
proof –
{ **fix** *r* **assume** *well-order-on {x. x < n}* *r*
hence *natLeq-on n ≤_o r*
using *finite-atLeastLessThan natLeq-on-well-order-on*
finite-well-order-on-ordIso ordIso-iff-ordLeq **by** *blast*
}
then show *?thesis*
unfolding *card-order-on-def* **using** *Field-natLeq-on natLeq-on-Well-order* **by**
presburger
qed

corollary *card-of-Field-natLeq-on*:
 $|\text{Field } (\text{natLeq-on } n)| =_o \text{natLeq-on } n$
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*:
 $|\{0 ..< n\}| =_o \text{natLeq-on } n$
using *Field-natLeq-on card-of-Field-natLeq-on*
unfolding *atLeast-0 atLeastLessThan-def lessThan-def Int-UNIV-left* **by** *auto*

lemma *natLeq-on-ordLeq-less-eq*:
 $((\text{natLeq-on } m) \leq_o (\text{natLeq-on } n)) = (m \leq n)$
proof
assume *natLeq-on m ≤_o natLeq-on n*
then obtain *f* **where** *inj-on f {x. x < m} ∧ f ‘ {x. x < m} ≤ {x. x < n}*
unfolding *ordLeq-def* **using**
embed-inj-on[OF natLeq-on-Well-order[of m], of natLeq-on n, unfolded Field-natLeq-on]
embed-Field Field-natLeq-on **by** *blast*
thus $m \leq n$ **using** *atLeastLessThan-less-eq2*
unfolding *atLeast-0 atLeastLessThan-def lessThan-def Int-UNIV-left* **by** *blast*
next
assume $m \leq n$
hence *inj-on id {0..<m} ∧ id ‘ {0..<m} ≤ {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 ≤_o natLeq-on n*
using *card-of-less ordIso-ordLeq-trans ordLeq-ordIso-trans ordIso-symmetric* **by**
blast
qed

lemma *natLeq-on-ordLeq-less*:
 $((\text{natLeq-on } m) <_o (\text{natLeq-on } n)) = (m < n)$
using *not-ordLeq-iff-ordLess*[*OF natLeq-on-Well-order natLeq-on-Well-order, of*
n m]
natLeq-on-ordLeq-less-eq[*of n m*] **by** *linarith*

lemma *ordLeq-natLeq-on-imp-finite*:
assumes $|A| \leq_o \text{natLeq-on } n$
shows *finite A*
proof –
have $|A| \leq_o |\{0 \dots n\}|$
using *assms card-of-less ordIso-symmetric ordLeq-ordIso-trans* **by** *blast*
thus *?thesis* **by** (*auto simp: card-of-ordLeq-finite*)
qed

10.6.3 "Backward compatibility" with the numeric cardinal operator for finite sets

lemma *finite-card-of-iff-card2*:
assumes *FIN: finite A and FIN': finite B*
shows $(|A| \leq_o |B|) = (\text{card } A \leq \text{card } B)$
using *assms card-of-ordLeq*[*of A B*] *inj-on-iff-card-le*[*of A B*] **by** *blast*

lemma *finite-imp-card-of-natLeq-on*:
assumes *finite A*
shows $|A| =_o \text{natLeq-on } (\text{card } A)$
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*:
 $\text{finite } A = (\exists n. |A| =_o \text{natLeq-on } n)$
using *finite-imp-card-of-natLeq-on*[*of A*]
by (*auto simp: ordIso-natLeq-on-imp-finite*)

lemma *finite-card-of-iff-card*:
assumes *FIN: finite A and FIN': finite B*
shows $(|A| =_o |B|) = (\text{card } A = \text{card } B)$
using *assms card-of-ordIso*[*of A B*] *bij-betw-iff-card*[*of A B*] **by** *blast*

lemma *finite-card-of-iff-card3*:
assumes *FIN: finite A and FIN': finite B*
shows $(|A| <_o |B|) = (\text{card } A < \text{card } B)$
proof –
have $(|A| <_o |B|) = (\sim (|B| \leq_o |A|))$ **by** *simp*
also have $\dots = (\sim (\text{card } B \leq \text{card } A))$
using *assms* **by** (*simp add: finite-card-of-iff-card2*)

also have ... = (card A < card B) by auto
 finally show ?thesis .
 qed

lemma card-Field-natLeq-on:
 card(Field(natLeq-on n)) = n
 using Field-natLeq-on card-atLeastLessThan by auto

10.7 The successor of a cardinal

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-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 ∧ Well-order r' ∧ Well-order(cardSuc r) ∧ 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-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 ∧ Card-order r' ∧ 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 ∧ Well-order r' ∧ Well-order p
 unfolding card-order-on-def by force
 have FIELD: Field r = {0.._n} ∧ Field p = {0.._(Suc n)}
 unfolding r-def p-def Field-natLeq-on atLeast-0 atLeastLessThan-def lessThan-def
 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 $\text{LESS: } |\text{Field } r| <_o |\text{Field } r'|$
 using $\text{CARD card-of-Field-ordIso ordLess-ordIso-trans ordIso-symmetric}$ by blast
 blast

 have $r' \leq_o p$ using $\text{CARD unfolding } r\text{-def } r'\text{-def } p\text{-def}$
 using $\text{natLeq-on-ordLeq-less cardSuc-ordLess-ordLeq}$ by blast
 moreover have $p \leq_o r'$
 proof –
 {assume $r' <_o p$
 then obtain f where $0: \text{embedS } r' p f$ unfolding ordLess-def by force
 let $?q = \text{Restr } p (f \text{ ` } \text{Field } r')$
 have $1: \text{embed } r' p f$ using 0 unfolding embedS-def by force
 hence $2: f \text{ ` } \text{Field } r' < \{0..<(\text{Suc } n)\}$
 using $\text{WELL FIELD } 0$ by $(\text{auto simp: embedS-iff})$
 have $\text{wo-rel.ofilter } p (f \text{ ` } \text{Field } r')$ using $\text{embed-Field-ofilter } 1 \text{ WELL}$ by blast
 then obtain m where $m \leq \text{Suc } n$ and $3: f \text{ ` } (\text{Field } r') = \{0..<m\}$
 unfolding $p\text{-def}$ by $(\text{auto simp: natLeq-on-ofilter-iff})$
 hence $4: m \leq n$ using 2 by force

 have $\text{bij-betw } f (\text{Field } r') (f \text{ ` } (\text{Field } r'))$
 using $\text{WELL embed-inj-on}[OF - 1]$ unfolding bij-betw-def by blast
 moreover have $\text{finite}(f \text{ ` } (\text{Field } r'))$ using $3 \text{ finite-atLeastLessThan}[of 0 m]$
 by force
 ultimately have $5: \text{finite } (\text{Field } r') \wedge \text{card}(\text{Field } r') = \text{card } (f \text{ ` } (\text{Field } r'))$
 using $\text{bij-betw-same-card bij-betw-finite}$ by metis
 hence $\text{card}(\text{Field } r') \leq \text{card}(\text{Field } r)$ using $3 \ 4 \ \text{FIELD}$ by force
 hence $|\text{Field } r'| \leq_o |\text{Field } r|$ using $\text{FIN } 5 \text{ finite-card-of-iff-card2}$ by blast
 hence False using $\text{LESS not-ordLess-ordLeq}$ by auto
 }
 thus $?thesis$ using WELL CARD by fastforce
 qed
 ultimately show $?thesis$ using ordIso-iff-ordLeq unfolding $r'\text{-def } p\text{-def}$ by blast
 blast
 qed

 lemma $\text{card-of-Plus-ordLeq-infinite[simp]}$:
 assumes $\neg \text{finite } C$ and $|A| \leq_o |C|$ and $|B| \leq_o |C|$
 shows $|A <+> B| \leq_o |C|$
 by (simp add: assms)

 lemma $\text{card-of-Un-ordLeq-infinite[simp]}$:
 assumes $\neg \text{finite } C$ and $|A| \leq_o |C|$ and $|B| \leq_o |C|$
 shows $|A \text{ Un } B| \leq_o |C|$
 using $\text{assms card-of-Plus-ordLeq-infinite card-of-Un-Plus-ordLeq ordLeq-transitive}$
 by metis

10.8 Others

lemma *under-mono[simp]*:
assumes *Well-order r* **and** $(i,j) \in r$
shows $\text{under } r \ i \subseteq \text{under } r \ j$
using *assms* **unfolding** *under-def order-on-defs trans-def* **by** *blast*

lemma *underS-under*:
assumes $i \in \text{Field } r$
shows $\text{underS } r \ i = \text{under } r \ i - \{i\}$
using *assms* **unfolding** *underS-def under-def* **by** *auto*

lemma *relChain-under*:
assumes *Well-order r*
shows $\text{relChain } r \ (\lambda i. \text{under } r \ i)$
using *assms* **unfolding** *relChain-def* **by** *auto*

lemma *card-of-infinite-diff-finite*:
assumes $\neg \text{finite } A$ **and** *finite B*
shows $|A - B| =_o |A|$
by (*metis assms card-of-Un-diff-infinite finite-ordLess-infinite2*)

lemma *infinite-card-of-diff-singl*:
assumes $\neg \text{finite } A$
shows $|A - \{a\}| =_o |A|$
by (*metis assms card-of-infinite-diff-finite finite.emptyI finite-insert*)

lemma *card-of-vimage*:
assumes $B \subseteq \text{range } f$
shows $|B| \leq_o |f - ` B|$
by (*metis Int-absorb2 assms image-vimage-eq order-refl surj-imp-ordLeq*)

lemma *surj-card-of-vimage*:
assumes *surj f*
shows $|B| \leq_o |f - ` B|$
by (*metis assms card-of-vimage subset-UNIV*)

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* $r \neq \{\}$ **and** *a: a ∈ 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  $(\lambda \ a. \ \{a\}) \ A$  unfolding inj-on-def by auto
  moreover have  $(\lambda \ a. \ \{a\}) \ 'A \subseteq Bpow \ r \ A$ 
    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  $A$ :  $\neg finite \ A$ 
  shows  $\neg finite \ (Bpow \ r \ A)$ 
    using ordLeq-card-Bpow[OF rc r]
    by (metis A card-of-ordLeq-infinite)

definition Func-option where
  Func-option  $A \ B \equiv$ 
   $\{f. (\forall \ a. \ f \ a \neq None \longleftrightarrow a \in A) \wedge (\forall \ a \in A. \ case \ f \ a \ of \ Some \ b \Rightarrow b \in B \mid None \Rightarrow True)\}$ 

lemma card-of-Func-option-Func:
   $|Func-option \ A \ B| = o \ |Func \ A \ B|$ 
proof (rule ordIso-symmetric, unfold card-of-ordIso[symmetric], intro exI)
  let  $?F = \lambda \ f \ a. \ if \ a \in A \ then \ Some \ (f \ a) \ else \ None$ 
  show bij-betw  $?F \ (Func \ A \ B) \ (Func-option \ A \ B)$ 
    unfolding bij-betw-def unfolding inj-on-def proof(intro conjI ballI impI)
    fix  $f \ g$  assume  $f: f \in Func \ A \ B$  and  $g: g \in Func \ A \ B$  and  $eq: ?F \ f = ?F \ g$ 
    show  $f = g$ 
    proof(rule ext)
      fix  $a$ 
      show  $f \ a = g \ a$ 
        by (smt (verit) Func-def eq f g mem-Collect-eq option.inject)
    qed
  next
  show  $?F \ 'Func \ A \ B = Func-option \ A \ B$ 
  proof safe
    fix  $f$  assume  $f: f \in Func-option \ A \ B$ 
    define  $g$  where [abs-def]:  $g \ a = (case \ f \ a \ of \ Some \ b \Rightarrow b \mid None \Rightarrow undefined)$ 
  for  $a$ 
    have  $g \in Func \ A \ B$ 
    using  $f$  unfolding g-def Func-def Func-option-def by force+
    moreover have  $f = ?F \ g$ 

```

```

proof(rule ext)
  fix a show  $f\ a = ?F\ g\ a$ 
    using f unfolding Func-option-def g-def by (cases a ∈ A) force+
qed
ultimately show  $f \in ?F\ ' (Func\ A\ B)$  by blast
qed(unfold Func-def Func-option-def, auto)
qed
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-Pfunc*:

```

Func-option A B  $\subseteq$  Pfunc A B
unfolding Func-option-def Pfunc-def by auto

```

lemma *Pfunc-Func-option*:

```

Pfunc A B = ( $\bigcup A' \in Pow\ A. Func-option\ A'\ B$ )
proof safe
  fix f assume f:  $f \in Pfunc\ A\ B$ 
  show  $f \in (\bigcup A' \in Pow\ A. Func-option\ A'\ B)$ 
  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  $f \in Func-option\ ?A'\ B$  using f unfolding Func-option-def Pfunc-def
by auto
  qed
next
  fix f A' assume  $f \in Func-option\ A'\ B$  and  $A' \subseteq A$ 
  thus  $f \in Pfunc\ A\ B$  unfolding Func-option-def Pfunc-def by auto
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
  define F where [abs-def]:  $F\ f1\ a =$ 
    (if  $a \in A2$  then (if  $a \in A1$  then  $f1\ a$  else bb) else undefined) for f1 :: 'a  $\Rightarrow$  'b
and a
  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  $f a = g a$ 
      by (smt (verit) A12 F-def Func-def eq f g mem-Collect-eq subsetD)
    qed
  qed
qed(insert bb, unfold Func-def F-def, force)
qed

```

```

lemma card-of-Func-option-mono:
  fixes  $A1 A2 :: 'a \text{ set}$  and  $B :: 'b \text{ set}$ 
  assumes  $A12: A1 \subseteq A2$  and  $B: B \neq \{\}$ 
  shows  $|Func\text{-option } A1 B| \leq_o |Func\text{-option } A2 B|$ 
  by (metis card-of-Func-mono[OF A12 B] card-of-Func-option-Func
    ordIso-ordLeq-trans ordLeq-ordIso-trans ordIso-symmetric)

```

```

lemma card-of-Pfunc-Pow-Func-option:
  assumes  $B \neq \{\}$ 
  shows  $|Pfunc A B| \leq_o |Pow A \times Func\text{-option } A B|$ 
proof-
  have  $|Pfunc A B| =_o |\bigcup A' \in Pow A. Func\text{-option } A' B|$  (is  $=_o ?K$ )
    unfolding Pfunc-Func-option by(rule card-of-refl)
  also have  $?K \leq_o |Sigma (Pow A) (\lambda A'. Func\text{-option } A' B)|$  using card-of-UNION-Sigma
  .
  also have  $|Sigma (Pow A) (\lambda A'. Func\text{-option } A' B)| \leq_o |Pow A \times Func\text{-option } A B|$ 
    by (simp add: asms card-of-Func-option-mono card-of-Sigma-mono1)
  finally show ?thesis .
qed

```

```

lemma Bpow-ordLeq-Func-Field:
  assumes  $rc: Card\text{-order } r$  and  $r: Field r \neq \{\}$  and  $A: \neg finite A$ 
  shows  $|Bpow r A| \leq_o |Func (Field r) A|$ 
proof-
  let  $?F = \lambda f. \{x \mid x a. f a = x \wedge a \in Field r\}$ 
  {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
    define  $f$  where [abs-def]:  $f i = (if i \in Field r \text{ then } F i \text{ else undefined})$  for  $i$ 
    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: ¬finite (Bpow r A) using infinite-Bpow[OF rc r A] .
 have |Bpow r A| =o |Bpow r A - {{}}|
   by (metis 2 infinite-card-of-diff-singl ordIso-symmetric)
}
ultimately show ?thesis by (metis ordIso-ordLeq-trans)
qed

```

```

lemma empty-in-Func[simp]:
  B ≠ {} ⟹ (λx. undefined) ∈ Func {} B
by simp

```

```

lemma Func-mono[simp]:
  assumes B1 ⊆ B2
  shows Func A B1 ⊆ Func A B2
using assms unfolding Func-def by force

```

```

lemma Pfunc-mono[simp]:
  assumes A1 ⊆ A2 and B1 ⊆ B2
  shows Pfunc A B1 ⊆ Pfunc A B2
using assms unfolding Pfunc-def
by (force split: option.split-asm option.split)

```

```

lemma card-of-Func-UNIV-UNIV:
  |Func (UNIV::'a set) (UNIV::'b set)| =o |UNIV::('a ⇒ 'b) set|
using card-of-Func-UNIV[of UNIV::'b set] by auto

```

```

lemma ordLeq-Func:
  assumes {b1,b2} ⊆ B b1 ≠ b2
  shows |A| ≤o |Func A B|
  unfolding card-of-ordLeq[symmetric] proof(intro exI conjI)
  let ?F = λx a. if a ∈ A then (if a = x then b1 else b2) else undefined
  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 by auto
qed

```

```

lemma infinite-Func:
  assumes A: ¬finite A and B: {b1,b2} ⊆ B b1 ≠ b2
  shows ¬finite (Func A B)
using ordLeq-Func[OF B] by (metis A card-of-ordLeq-finite)

```

10.9 Infinite cardinals are limit ordinals

```

lemma card-order-infinite-isLimOrd:
  assumes c: Card-order r and i: ¬finite (Field r)
  shows isLimOrd r
proof-
  have 0: wo-rel r and 00: Well-order r
    using c unfolding card-order-on-def wo-rel-def by auto
  hence rr: Refl r by (metis wo-rel.REFL)

```

```

show ?thesis unfolding wo-rel.isLimOrd-def[OF 0] wo-rel.isSuccOrd-def[OF 0]

proof safe
  fix j assume j ∈ Field r and ∀ i ∈ Field r. (i, j) ∈ r
  then show False
    by (metis Card-order-trans c i infinite-Card-order-limit)
qed
qed

lemma insert-Chain:
  assumes Refl r C ∈ Chains r and i ∈ Field r and ⋀ j. j ∈ C ⇒ (j, i) ∈ r ∨
  (i, j) ∈ r
  shows insert i C ∈ Chains r
  using assms unfolding Chains-def by (auto dest: refl-onD)

lemma Collect-insert: {R j | j. j ∈ insert j1 J} = insert (R j1) {R j | j. j ∈ J}
  by auto

lemma Field-init-seg-of[simp]:
  Field init-seg-of = UNIV
  unfolding Field-def init-seg-of-def by auto

lemma refl-init-seg-of[intro, simp]: refl init-seg-of
  unfolding refl-on-def Field-def by auto

lemma regularCard-all-ex:
  assumes r: Card-order r regularCard r
  and As: ⋀ i j b. b ∈ B ⇒ (i, j) ∈ r ⇒ P i b ⇒ P j b
  and Bsub: ∀ b ∈ B. ∃ i ∈ Field r. P i b
  and cardB: |B| < o r
  shows ∃ i ∈ Field r. ∀ b ∈ B. P i b
proof–
  let ?As = λ i. {b ∈ B. P i b}
  have ∃ i ∈ Field r. B ≤ ?As i
  apply(rule regularCard-UNION) using assms unfolding relChain-def by auto
  thus ?thesis by auto
qed

lemma relChain-stabilize:
  assumes rc: relChain r As and AsB: (⋃ i ∈ Field r. As i) ⊆ B and Br: |B| < o
  r
  and ir: ¬finite (Field r) and cr: Card-order r
  shows ∃ i ∈ Field r. As (succ r i) = As i
proof(rule ccontr, auto)
  have 0: wo-rel r and 00: Well-order r
  unfolding wo-rel-def by (metis card-order-on-well-order-on cr)+
  have L: isLimOrd r using ir cr by (metis card-order-infinite-isLimOrd)
  have AsBs: (⋃ i ∈ Field r. As (succ r i)) ⊆ B
  using AsB L by (simp add: 0 Sup-le-iff wo-rel.isLimOrd-succ)

```

```

assume  $As\text{-}s: \forall i \in Field\ r. As\ (succ\ r\ i) \neq As\ i$ 
have  $1: \forall i\ j. (i, j) \in r \wedge i \neq j \longrightarrow As\ i \subseteq As\ j$ 
proof safe
  fix  $i\ j$  assume  $1: (i, j) \in r\ i \neq j$  and  $Asij: As\ i = As\ j$ 
  hence  $rij: (succ\ r\ i, j) \in r$  by (metis  $0\ wo\text{-}rel.\text{succ}\text{-}smallest$ )
  hence  $As\ (succ\ r\ i) \subseteq As\ j$  using  $rc$  unfolding  $relChain\text{-}def$  by auto
  moreover
    { have  $(i, succ\ r\ i) \in r$ 
      by (meson  $0\ 1(1)\ FieldI1\ L\ wo\text{-}rel.\text{isLimOrd}\text{-}aboveS\ wo\text{-}rel.\text{succ}\text{-}in$ )
      hence  $As\ i \subseteq As\ (succ\ r\ i)$  using  $As\text{-}s\ rc\ rij$  unfolding  $relChain\text{-}def\ Field\text{-}def$ 
    }
  by auto
  }
  ultimately show False unfolding  $Asij$  by auto
qed (insert  $rc$ , unfold  $relChain\text{-}def$ , auto)
hence  $\forall i \in Field\ r. \exists a. a \in As\ (succ\ r\ i) - As\ i$ 
  using  $wo\text{-}rel.\text{succ}\text{-}in[OF\ 0]\ AsB$ 
  by(metis  $0\ card\text{-}order\text{-}infinite\text{-}isLimOrd\ cr\ ir\ psubset\text{-}imp\text{-}ex\text{-}mem$ 
     $wo\text{-}rel.\text{isLimOrd}\text{-}aboveS\ wo\text{-}rel.\text{succ}\text{-}diff$ )
then obtain  $f$  where  $f: \forall i \in Field\ r. f\ i \in As\ (succ\ r\ i) - As\ i$  by metis
have  $inj\text{-}on\ f\ (Field\ r)$  unfolding  $inj\text{-}on\text{-}def$ 
proof safe
  fix  $i\ j$  assume  $ij: i \in Field\ r\ j \in Field\ r$  and  $fij: f\ i = f\ j$ 
  show  $i = j$ 
  proof(cases rule:  $wo\text{-}rel.\text{cases}\text{-}Total3[OF\ 0]$ , safe)
    assume  $(i, j) \in r$  and  $ijd: i \neq j$ 
    hence  $rij: (succ\ r\ i, j) \in r$  by (metis  $0\ wo\text{-}rel.\text{succ}\text{-}smallest$ )
    hence  $As\ (succ\ r\ i) \subseteq As\ j$  using  $rc$  unfolding  $relChain\text{-}def$  by auto
    thus  $i = j$  using  $ij\ ijd\ fij\ f$  by auto
  next
    assume  $(j, i) \in r$  and  $ijd: i \neq j$ 
    hence  $rij: (succ\ r\ j, i) \in r$  by (metis  $0\ wo\text{-}rel.\text{succ}\text{-}smallest$ )
    hence  $As\ (succ\ r\ j) \subseteq As\ i$  using  $rc$  unfolding  $relChain\text{-}def$  by auto
    thus  $j = i$  using  $ij\ ijd\ fij\ f$  by fastforce
  qed(insert  $ij$ , auto)
qed
  moreover have  $f' (Field\ r) \subseteq B$  using  $f\ AsBs$  by auto
  moreover have  $|B| < o\ |Field\ r|$  using  $Br\ cr$  by (metis  $card\text{-}of\text{-}unique\ ord\text{-}Less\text{-}ordIso\text{-}trans$ )
  ultimately show False unfolding  $card\text{-}of\text{-}ordLess[symmetric]$  by auto
qed

```

10.10 Regular vs. stable cardinals

lemma *stable-cardSuc*:

```

assumes  $CARD: Card\text{-}order\ r$  and  $INF: \neg finite\ (Field\ r)$ 
shows  $stable(cardSuc\ r)$ 
using  $infinite\text{-}cardSuc\text{-}regularCard\ regularCard\text{-}stable$ 
by (metis  $CARD\ INF\ cardSuc\text{-}Card\text{-}order\ cardSuc\text{-}finite$ )

```

```

lemma stable-ordIso:
  assumes  $r =_o r'$ 
  shows  $\text{stable } r = \text{stable } r'$ 
  by (metis assms ordIso-symmetric stable-ordIso1)

```

```

lemma stable-nat:  $\text{stable } |\text{UNIV}::\text{nat set}|$ 
  using stable-natLeq card-of-nat stable-ordIso by auto

```

Below, the type of "A" is not important – we just had to choose an appropriate type to make "A" possible. What is important is that arbitrarily large infinite sets of stable cardinality exist.

```

lemma infinite-stable-exists:
  assumes CARD:  $\forall r \in R. \text{Card-order } (r::'a \text{ rel})$ 
  shows  $\exists (A :: (\text{nat} + 'a \text{ set}) \text{ set}).$ 
     $\neg \text{finite } A \wedge \text{stable } |A| \wedge (\forall r \in R. r <_o |A|)$ 
proof –
  have  $\exists (A :: (\text{nat} + 'a \text{ set}) \text{ set}).$ 
     $\neg \text{finite } A \wedge \text{stable } |A| \wedge |\text{UNIV}::'a \text{ set}| <_o |A|$ 
proof (cases finite (UNIV::'a set))
  case True
  let  $?B = \text{UNIV}::\text{nat set}$ 
  have  $\neg \text{finite} (?B <+> \{\})$  using finite-Plus-iff by blast
moreover
  have  $\text{stable } |?B|$  using stable-natLeq card-of-nat stable-ordIso1 by blast
  hence  $\text{stable } |?B <+> \{\}|$  using stable-ordIso card-of-Plus-empty1 by blast
moreover
  have  $\neg \text{finite} (\text{Field } |?B|) \wedge \text{finite} (\text{Field } |\text{UNIV}::'a \text{ set}|)$  using True by simp
  hence  $|\text{UNIV}::'a \text{ set}| <_o |?B|$  by (simp add: finite-ordLess-infinite)
  hence  $|\text{UNIV}::'a \text{ set}| <_o |?B <+> \{\}|$  using card-of-Plus-empty1 ordLess-ordIso-trans
by blast
  ultimately show ?thesis by blast
next
  case False
  hence  $\neg \text{finite} (\text{Field } |\text{UNIV}::'a \text{ set}|)$  by simp
  let  $?B = \text{Field}(\text{cardSuc } |\text{UNIV}::'a \text{ set}|)$ 
  have  $0: |?B| =_o |\{\}| <+> ?B$  using card-of-Plus-empty2 by blast
  have  $1: \neg \text{finite } ?B$  using False card-of-cardSuc-finite by blast
  hence  $2: \neg \text{finite} (\{\} <+> ?B)$  using 0 card-of-ordIso-finite by blast
  have  $|?B| =_o \text{cardSuc } |\text{UNIV}::'a \text{ set}|$ 
    using card-of-Card-order cardSuc-Card-order card-of-Field-ordIso by blast
moreover have  $\text{stable}(\text{cardSuc } |\text{UNIV}::'a \text{ set}|)$ 
    using stable-cardSuc card-of-Card-order by blast
ultimately have  $\text{stable } |?B|$  using stable-ordIso by blast
  hence  $3: \text{stable } |\{\} <+> ?B|$  using stable-ordIso 0 by blast
  have  $|\text{UNIV}::'a \text{ set}| <_o \text{cardSuc } |\text{UNIV}::'a \text{ set}|$ 
    using card-of-Card-order cardSuc-greater by blast
moreover have  $|?B| =_o \text{cardSuc } |\text{UNIV}::'a \text{ set}|$ 
    using card-of-Card-order cardSuc-Card-order card-of-Field-ordIso by blast
ultimately have  $|\text{UNIV}::'a \text{ set}| <_o |?B|$ 

```

```

    using ordIso-symmetric ordLess-ordIso-trans by blast
  hence  $|UNIV::'a\ set| <_o |\{\} <+> ?B|$  using 0 ordLess-ordIso-trans by blast
  thus ?thesis using 2 3 by blast
qed
thus ?thesis using CARD card-of-UNIV2 ordLeq-ordLess-trans by blast
qed

corollary infinite-regularCard-exists:
  assumes CARD:  $\forall r \in R. \text{Card-order } (r::'a\ rel)$ 
  shows  $\exists (A :: (nat + 'a\ set)\ set).$ 
     $\neg \text{finite } A \wedge \text{regularCard } |A| \wedge (\forall r \in R. r <_o |A|)$ 
  using infinite-stable-exists[OF CARD] stable-regularCard by (metis Field-card-of
card-of-card-order-on)

end

```

11 Cardinal Arithmetic

```

theory Cardinal-Arithmetic
  imports Cardinal-Order-Relation
begin

```

11.1 Binary sum

```

lemma csum-Cnotzero2:
  Cnotzero r2  $\implies$  Cnotzero (r1 +c r2)
  unfolding csum-def
  by (metis Cnotzero-imp-not-empty Field-card-of Plus-eq-empty-conv card-of-card-order-on
czeroE)

```

```

lemma single-cone:
   $|\{x\}| =_o \text{cone}$ 
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 cone-Cnotzero: Cnotzero cone
  by (simp add: cone-not-czero Card-order-cone)

```

```

lemma cone-ordLeq-ctwo: cone  $\leq_o$  ctwo
  unfolding cone-def ctwo-def card-of-ordLeq[symmetric] by auto

```

```

lemma csum-czero1: Card-order r  $\implies$  r +c czero =o r
  unfolding czero-def csum-def Field-card-of
  by (rule ordIso-transitive[OF ordIso-symmetric[OF card-of-Plus-empty1] card-of-Field-ordIso])

```

```

lemma csum-czero2: Card-order r  $\implies$  czero +c r =o r

```

unfolding *czero-def csum-def Field-card-of*
by (*rule ordIso-transitive*[*OF ordIso-symmetric*[*OF card-of-Plus-empty2*] *card-of-Field-ordIso*])

11.2 Product

lemma *Times-cprod*: $|A \times B| =_o |A| *c |B|$
by (*simp only: cprod-def Field-card-of card-of-refl*)

lemma *card-of-Times-singleton*:

fixes $A :: 'a \text{ set}$

shows $|A \times \{x\}| =_o |A|$

proof –

define $f :: 'a \times 'b \Rightarrow 'a$ **where** $f = (\lambda(a, b). a)$

have $A \subseteq f^{-1}(A \times \{x\})$ **unfolding** *f-def* **by** (*auto simp: image-iff*)

hence *bij-betw* f $(A \times \{x\})$ A **unfolding** *bij-betw-def inj-on-def f-def* **by** *fastforce*

thus *?thesis* **using** *card-of-ordIso* **by** *blast*

qed

lemma *cprod-assoc*: $(r *c s) *c t =_o r *c s *c t$
unfolding *cprod-def Field-card-of* **by** (*rule card-of-Times-assoc*)

lemma *cprod-czero*: $r *c \text{czero} =_o \text{czero}$
unfolding *cprod-def czero-def Field-card-of* **by** (*simp add: card-of-empty-ordIso*)

lemma *cprod-cone*: $\text{Card-order } r \Longrightarrow r *c \text{cone} =_o r$
unfolding *cprod-def cone-def Field-card-of*
by (*metis (no-types) card-of-Field-ordIso card-of-Times-singleton ordIso-transitive*)

lemma *ordLeq-cprod1*: $\llbracket \text{Card-order } p1; \text{Cnotzero } p2 \rrbracket \Longrightarrow p1 \leq_o p1 *c p2$
unfolding *cprod-def* **by** (*metis Card-order-Times1 czeroI*)

11.3 Exponentiation

lemma *cexp-czero*: $r \hat{^c} \text{czero} =_o \text{cone}$
unfolding *cexp-def czero-def Field-card-of Func-empty* **by** (*rule single-cone*)

lemma *Pow-cexp-ctwo*:

$|Pow A| =_o \text{ctwo} \hat{^c} |A|$

by (*simp add: card-of-Pow-Func cexp-def ctwo-def*)

lemma *Cnotzero-cexp*:

assumes *Cnotzero* q

shows *Cnotzero* $(q \hat{^c} r)$

proof –

have *Field* $q \neq \{\}$

by (*metis Card-order-iff-ordIso-card-of assms(1) czero-def*)

then show *?thesis*

by (*simp add: card-of-ordIso-czero-iff-empty cexp-def*)

qed

lemma *Cinfinite-ctwo-cexp*:
 $Cinfinite\ r \implies Cinfinite\ (ctwo \hat{c}\ r)$
unfolding *ctwo-def cexp-def cinfinite-def Field-card-of*
by (*rule conjI, rule infinite-Func, auto*)

lemma *cone-ordLeq-iff-Field*:
assumes $cone \leq_o r$
shows $Field\ r \neq \{\}$
by (*metis assms card-of-empty3 card-of-mono2 cone-Cnotzero czeroI*)

lemma *cone-ordLeq-cexp*: $cone \leq_o r1 \implies cone \leq_o r1 \hat{c}\ r2$
by (*simp add: cexp-def cone-def Func-non-emp cone-ordLeq-iff-Field*)

lemma *Card-order-czero*: $Card\text{-}order\ czero$
by (*simp only: card-of-Card-order czero-def*)

lemma *cexp-mono2''*:
assumes $2: p2 \leq_o r2$
and $n1: Cnotzero\ q$
and $n2: Card\text{-}order\ p2$
shows $q \hat{c}\ p2 \leq_o q \hat{c}\ r2$
proof (*cases p2 =o (czero :: 'a rel)*)
case *True*
hence $q \hat{c}\ p2 =o q \hat{c}\ (czero :: 'a\ rel)$ **using** $n1\ n2\ cexp\text{-}cong2\ Card\text{-}order\text{-}czero$
by *blast*
also have $q \hat{c}\ (czero :: 'a\ rel) =o\ cone$ **using** *cexp-czero* **by** *blast*
also have $cone \leq_o q \hat{c}\ r2$ **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

lemma *csum-cexp*: $\llbracket Cinfinite\ r1; Cinfinite\ r2; Card\text{-}order\ q; ctwo \leq_o q \rrbracket \implies$
 $q \hat{c}\ r1 +_c q \hat{c}\ r2 \leq_o q \hat{c}\ (r1 +_c r2)$
apply (*rule csum-cinfinite-bound*)
apply (*metis cexp-mono2' cinfinite-def finite.emptyI ordLeq-csum1*)
apply (*metis cexp-mono2' cinfinite-def finite.emptyI ordLeq-csum2*)
by (*simp-all add: Card-order-cexp Cinfinite-csum1 Cinfinite-cexp cinfinite-cexp*)

lemma *csum-cexp'*: $\llbracket Cinfinite\ r; Card\text{-}order\ q; ctwo \leq_o q \rrbracket \implies q +_c r \leq_o q \hat{c}\ r$
apply (*rule csum-cinfinite-bound*)
apply (*metis Cinfinite-Cnotzero ordLeq-cexp1*)
apply (*metis ordLeq-cexp2*)
apply *blast+*
by (*metis Cinfinite-cexp*)

lemma *card-of-Sigma-ordLeq-Cinfinite*:
 $\llbracket Cinfinite\ r; |I| \leq_o r; \forall i \in I. |A\ i| \leq_o r \rrbracket \implies |SIGMA\ i : I. A\ i| \leq_o r$

unfolding *cinfinite-def* **by** (*blast intro: card-of-Sigma-ordLeq-infinite-Field*)

lemma *Cinfinite-ordLess-cexp*:
assumes r : *Cinfinite* r
shows $r <_o r \hat{^c} r$
proof –
have $r <_o \text{ctwo} \hat{^c} r$ **using** r **by** (*simp only: ordLess-ctwo-cexp*)
also have $\text{ctwo} \hat{^c} r \leq_o r \hat{^c} r$
by (*rule cexp-mono1[OF ctwo-ordLeq-Cinfinite]*) (*auto simp: r ctwo-not-czero Card-order-ctwo*)
finally show ?thesis .
qed

lemma *infinite-ordLeq-cexp*:
assumes *Cinfinite* r
shows $r \leq_o r \hat{^c} r$
by (*rule ordLess-imp-ordLeq[OF Cinfinite-ordLess-cexp[OF assms]]*)

lemma *czero-cexp*: $\text{Cnotzero } r \implies \text{czero} \hat{^c} r =_o \text{czero}$
by (*metis Cnotzero-imp-not-empty cexp-def czero-def card-of-empty-ordIso Field-card-of Func-is-emp*)

lemma *Func-singleton*:
fixes $x :: 'b$ **and** $A :: 'a \text{ set}$
shows $|\text{Func } A \{x\}| =_o |\{x\}|$
proof (*rule ordIso-symmetric*)
define f **where** [*abs-def*]: $f \ y \ a = (\text{if } y = x \wedge a \in A \text{ then } x \text{ else undefined})$ **for** $y \ a$
have $\text{Func } A \{x\} \subseteq f \ ' \{x\}$ **unfolding** f -def *Func-def* **by** (*force simp: fun-eq-iff*)
hence *bij-betw* $f \{x\}$ (*Func* $A \{x\}$)
unfolding *bij-betw-def inj-on-def f-def Func-def* **by** (*auto split: if-split-asm*)
thus $|\{x\}| =_o |\text{Func } A \{x\}|$ **using** *card-of-ordIso* **by** *blast*
qed

lemma *cone-cexp*: $\text{cone} \hat{^c} r =_o \text{cone}$
unfolding *cexp-def cone-def Field-card-of* **by** (*rule Func-singleton*)

lemma *card-of-Func-squared*:
fixes $A :: 'a \text{ set}$
shows $|\text{Func } (\text{UNIV} :: \text{bool set}) \ A| =_o |A \times A|$
proof (*rule ordIso-symmetric*)
define f **where** $f = (\lambda(x::'a,y) \ b. \text{if } A = \{\} \text{ then undefined else if } b \text{ then } x \text{ else } y)$
have $\text{Func } (\text{UNIV} :: \text{bool set}) \ A \subseteq f \ ' (A \times A)$ **unfolding** f -def *Func-def*
by (*auto simp: image-iff fun-eq-iff split: option.splits if-split-asm*) *blast*
hence *bij-betw* $f \ (A \times A)$ (*Func* $(\text{UNIV} :: \text{bool set}) \ A$)
unfolding *bij-betw-def inj-on-def f-def Func-def* **by** (*auto simp: fun-eq-iff*)
thus $|A \times A| =_o |\text{Func } (\text{UNIV} :: \text{bool set}) \ A|$ **using** *card-of-ordIso* **by** *blast*
qed

lemma *cexp-ctwo*: $r \hat{=} c \text{ two} =_o r * c \text{ r}$
unfolding *cexp-def ctwo-def cprod-def Field-card-of* **by** (*rule card-of-Func-squared*)

lemma *card-of-Func-Plus*:
fixes $A :: 'a \text{ set}$ **and** $B :: 'b \text{ set}$ **and** $C :: 'c \text{ set}$
shows $|Func (A <+> B) C| =_o |Func A C \times Func B C|$
proof (*rule ordIso-symmetric*)
define f **where** $f = (\lambda(g :: 'a \Rightarrow 'c, h :: 'b \Rightarrow 'c) \text{ ab. case ab of } Inl\ a \Rightarrow g\ a \mid Inr\ b \Rightarrow h\ b)$
define f' **where** $f' = (\lambda(f :: ('a + 'b) \Rightarrow 'c). (\lambda a. f\ (Inl\ a), \lambda b. f\ (Inr\ b)))$
have $f' \text{ ' } (Func\ A\ C \times Func\ B\ C) \subseteq Func\ (A <+> B)\ C$
unfolding *Func-def f-def* **by** (*force split: sum.splits*)
moreover **have** $f' \text{ ' } Func\ (A <+> B)\ C \subseteq Func\ A\ C \times Func\ B\ C$ **unfolding** *Func-def f'-def* **by** *force*
moreover **have** $\forall a \in Func\ A\ C \times Func\ B\ C. f'\ (f\ a) = a$ **unfolding** *f'-def f-def Func-def* **by** *auto*
moreover **have** $\forall a' \in Func\ (A <+> B)\ C. f\ (f'\ a') = a'$ **unfolding** *f'-def f-def Func-def*
by (*auto split: sum.splits*)
ultimately **have** $bij\text{-}betw\ f\ (Func\ A\ C \times Func\ B\ C)\ (Func\ (A <+> B)\ C)$
by (*intro bij-betw-byWitness[of - f' f]*)
thus $|Func\ A\ C \times Func\ B\ C| =_o |Func\ (A <+> B)\ C|$ **using** *card-of-ordIso*
by *blast*
qed

lemma *cexp-csum*: $r \hat{=} c\ (s + c\ t) =_o r \hat{=} c\ s * c\ r \hat{=} c\ t$
unfolding *cexp-def cprod-def csum-def Field-card-of* **by** (*rule card-of-Func-Plus*)

11.4 Powerset

definition *cpow* **where** $cpow\ r = |Pow\ (Field\ r)|$

lemma *card-order-cpow*: $card\text{-}order\ r \Longrightarrow card\text{-}order\ (cpow\ r)$
by (*simp only: cpow-def Field-card-order Pow-UNIV card-of-card-order-on*)

lemma *cpow-greater-eq*: $Card\text{-}order\ r \Longrightarrow r \leq_o cpow\ r$
by (*rule ordLess-imp-ordLeq*) (*simp only: cpow-def Card-order-Pow*)

lemma *Cinfinite-cpow*: $Cinfinite\ r \Longrightarrow Cinfinite\ (cpow\ r)$
unfolding *cpow-def cinfinite-def* **by** *simp*

lemma *Card-order-cpow*: $Card\text{-}order\ (cpow\ r)$
unfolding *cpow-def* **by** (*rule card-of-Card-order*)

lemma *cardSuc-ordLeq-cpow*: $Card\text{-}order\ r \Longrightarrow cardSuc\ r \leq_o cpow\ r$
unfolding *cpow-def* **by** (*metis Card-order-Pow cardSuc-ordLess-ordLeq card-of-Card-order*)

lemma *cpow-cexp-ctwo*: $cpow\ r =_o ctwo\ \hat{=} c\ r$

unfolding *cpow-def ctwo-def cexp-def Field-card-of* **by** (*rule card-of-Pow-Func*)

11.5 Inverse image

lemma *vimage-ordLeq*:

assumes $|A| \leq_o k$ **and** $\forall a \in A. |vimage\ f\ \{a\}| \leq_o k$ **and** *Cinfinite* k
shows $|vimage\ f\ A| \leq_o k$

proof–

have $vimage\ f\ A = (\bigcup a \in A. vimage\ f\ \{a\})$ **by** *auto*

also have $|\bigcup a \in A. vimage\ f\ \{a\}| \leq_o k$

using *UNION-Cinfinite-bound[OF assms]* .

finally show *?thesis* .

qed

11.6 Maximum

definition *cmax* **where**

cmax $r\ s =$

(*if* *cinfinite* $r \vee$ *cinfinite* s *then* $czero +_c r +_c s$

else $natLeq-on\ (max\ (card\ (Field\ r))\ (card\ (Field\ s))) +_c\ czero$)

lemma *cmax-com*: $cmax\ r\ s =_o\ cmax\ s\ r$

unfolding *cmax-def*

by (*auto simp: max.commute intro: csum-cong2[OF csum-com] csum-cong2[OF czero-ordIso]*)

lemma *cmax1*:

assumes *Card-order* r *Card-order* s $s \leq_o r$

shows $cmax\ r\ s =_o r$

unfolding *cmax-def*

proof (*split if-splits, intro conjI impI*)

assume *cinfinite* $r \vee$ *cinfinite* s

hence *Cinf*: *Cinfinite* r **using** *assms*(1,3) **by** (*metis cinfinite-mono*)

have $czero +_c r +_c s =_o r +_c s$ **by** (*rule csum-czero2[OF Card-order-csum]*)

also have $r +_c s =_o r$ **by** (*rule csum-absorb1[OF Cinf assms(3)]*)

finally show $czero +_c r +_c s =_o r$.

next

assume $\neg (cinfinite\ r \vee cinfinite\ s)$

hence *fin*: *finite* $(Field\ r)$ **and** *finite* $(Field\ s)$ **unfolding** *cinfinite-def* **by** *simp-all*

moreover

{ **from** *assms*(2) **have** $|Field\ s| =_o s$ **by** (*rule card-of-Field-ordIso*)

also from *assms*(3) **have** $s \leq_o r$.

also from *assms*(1) **have** $r =_o |Field\ r|$ **by** (*rule ordIso-symmetric[OF card-of-Field-ordIso]*)

finally have $|Field\ s| \leq_o |Field\ r|$.

}

ultimately have $card\ (Field\ s) \leq card\ (Field\ r)$ **by** (*subst sym[OF finite-card-of-iff-card2]*)

hence $max\ (card\ (Field\ r))\ (card\ (Field\ s)) = card\ (Field\ r)$ **by** (*rule max-absorb1*)

hence $natLeq-on\ (max\ (card\ (Field\ r))\ (card\ (Field\ s))) +_c\ czero =$

$natLeq-on\ (card\ (Field\ r)) +_c\ czero$ **by** *simp*

also have $\dots =_o natLeq-on\ (card\ (Field\ r))$ **by** (*rule csum-czero1[OF natLeq-on-Card-order]*)

also have $\text{natLeq-on } (\text{card } (\text{Field } r)) =_o |\text{Field } r|$
by (*rule ordIso-symmetric* [*OF finite-imp-card-of-natLeq-on* [*OF fin*]])
also from *assms*(1) **have** $|\text{Field } r| =_o r$ **by** (*rule card-of-Field-ordIso*)
finally show $\text{natLeq-on } (\max (\text{card } (\text{Field } r)) (\text{card } (\text{Field } s))) +_c \text{czero} =_o r$.
qed

lemma *cmax2*:
assumes *Card-order* *r* *Card-order* *s* $r \leq_o s$
shows $\text{cmax } r \text{ } s =_o s$
by (*metis assms cmax1 cmax-com ordIso-transitive*)

context
fixes *r s*
assumes *r*: *Cinfinite* *r*
and *s*: *Cinfinite* *s*
begin

lemma *cmax-csum*: $\text{cmax } r \text{ } s =_o r +_c s$
by (*simp add: Card-order-csum cmax-def csum-czero2 r*)

lemma *cmax-cprod*: $\text{cmax } r \text{ } s =_o r *_c s$
proof (*cases r ≤_o s*)
case *True*
hence $\text{cmax } r \text{ } s =_o s$ **by** (*metis cmax2 r s*)
also have $s =_o r *_c s$ **by** (*metis Cinfinite-Cnotzero True cprod-infinite2' ordIso-symmetric r s*)
finally show ?thesis .
next
case *False*
hence $s \leq_o r$ **by** (*metis ordLeq-total r s card-order-on-def*)
hence $\text{cmax } r \text{ } s =_o r$ **by** (*metis cmax1 r s*)
also have $r =_o r *_c s$ **by** (*metis Cinfinite-Cnotzero ⟨s ≤_o r⟩ cprod-infinite1' ordIso-symmetric r s*)
finally show ?thesis .
qed

end

lemma *Card-order-cmax*:
assumes *r*: *Card-order* *r* **and** *s*: *Card-order* *s*
shows *Card-order* ($\text{cmax } r \text{ } s$)
unfolding *cmax-def* **by** (*auto simp: Card-order-csum*)

lemma *ordLeq-cmax*:
assumes *r*: *Card-order* *r* **and** *s*: *Card-order* *s*
shows $r \leq_o \text{cmax } r \text{ } s \wedge s \leq_o \text{cmax } r \text{ } s$
by (*meson card-order-on-def cmax1 cmax2 ordIso-iff-ordLeq ordLeq-total ordLeq-transitive r s*)

lemmas $\text{ordLeq-cmax1} = \text{ordLeq-cmax}[\text{THEN conjunct1}]$ **and**
 $\text{ordLeq-cmax2} = \text{ordLeq-cmax}[\text{THEN conjunct2}]$

lemma *finite-cmax*:

assumes r : Card-order r **and** s : Card-order s

shows $\text{finite}(\text{Field}(\text{cmax } r \ s)) \longleftrightarrow \text{finite}(\text{Field } r) \wedge \text{finite}(\text{Field } s)$

by (*meson card-order-on-def cmax1 cmax2 ordIso-finite-Field ordLeq-finite-Field ordLeq-total r s*)

end

12 Extending Well-founded Relations to Wellorders

theory *Wellorder-Extension*

imports *Main Order-Union*

begin

12.1 Extending Well-founded Relations to Wellorders

A *downset* (also lower set, decreasing set, initial segment, or downward closed set) is closed w.r.t. smaller elements.

definition *downset-on* **where**

$\text{downset-on } A \ r = (\forall x \ y. (x, y) \in r \wedge y \in A \longrightarrow x \in A)$

lemma *downset-onI*:

$(\bigwedge x \ y. (x, y) \in r \implies y \in A \implies x \in A) \implies \text{downset-on } A \ r$

by (*auto simp: downset-on-def*)

lemma *downset-onD*:

$\text{downset-on } A \ r \implies (x, y) \in r \implies y \in A \implies x \in A$

unfolding *downset-on-def* **by** *blast*

Extensions of relations w.r.t. a given set.

definition *extension-on* **where**

$\text{extension-on } A \ r \ s = (\forall x \in A. \forall y \in A. (x, y) \in s \longrightarrow (x, y) \in r)$

lemma *extension-onI*:

$(\bigwedge x \ y. \llbracket x \in A; y \in A; (x, y) \in s \rrbracket \implies (x, y) \in r) \implies \text{extension-on } A \ r \ s$

by (*auto simp: extension-on-def*)

lemma *extension-onD*:

$\text{extension-on } A \ r \ s \implies x \in A \implies y \in A \implies (x, y) \in s \implies (x, y) \in r$

by (*auto simp: extension-on-def*)

lemma *downset-on-Union*:

assumes $\bigwedge r. r \in R \implies \text{downset-on}(\text{Field } r) \ p$

shows *downset-on* (*Field* ($\bigcup R$)) *p*
using *assms* **by** (*auto intro: downset-onI dest: downset-onD*)

lemma *chain-subset-extension-on-Union*:
assumes *chain* $\subseteq R$ **and** $\bigwedge r. r \in R \implies \text{extension-on } (\text{Field } r) \ r \ p$
shows *extension-on* (*Field* ($\bigcup R$)) ($\bigcup R$) *p*
using *assms*
by (*simp add: chain-subset-def extension-on-def*)
(*metis (no-types) mono-Field subsetD*)

lemma *downset-on-empty* [*simp*]: *downset-on* $\{\}$ *p*
by (*auto simp: downset-on-def*)

lemma *extension-on-empty* [*simp*]: *extension-on* $\{\}$ *p q*
by (*auto simp: extension-on-def*)

Every well-founded relation can be extended to a wellorder.

theorem *well-order-extension*:
assumes *wf* *p*
shows $\exists w. p \subseteq w \wedge \text{Well-order } w$
proof –
let $?K = \{r. \text{Well-order } r \wedge \text{downset-on } (\text{Field } r) \ p \wedge \text{extension-on } (\text{Field } r) \ r \ p\}$
define *I* **where** $I = \text{init-seg-of} \cap ?K \times ?K$
have *I-init*: $I \subseteq \text{init-seg-of}$ **by** (*simp add: I-def*)
then have *subch*: $\bigwedge R. R \in \text{Chains } I \implies \text{chain} \subseteq R$
by (*auto simp: init-seg-of-def chain-subset-def Chains-def*)
have *Chains-wo*: $\bigwedge R \ r. R \in \text{Chains } I \implies r \in R \implies$
 $\text{Well-order } r \wedge \text{downset-on } (\text{Field } r) \ p \wedge \text{extension-on } (\text{Field } r) \ r \ p$
by (*simp add: Chains-def I-def*) *blast*
have *FI*: *Field* *I* = $?K$ **by** (*auto simp: I-def init-seg-of-def Field-def*)
then have *O*: *Partial-order* *I*
by (*auto simp: partial-order-on-def preorder-on-def antisym-def antisym-init-seg-of refl-on-def*
trans-def I-def elim: trans-init-seg-of)
have $\bigcup R \in ?K \wedge (\forall r \in R. (r, \bigcup R) \in I)$ **if** $R \in \text{Chains } I$ **for** *R*
proof –
from *that* **have** *Ris*: $R \in \text{Chains } \text{init-seg-of}$ **using** *mono-Chains* [*OF I-init*]
by *blast*
have *subch*: $\text{chain} \subseteq R$ **using** $\langle R \in \text{Chains } I \rangle$ *I-init*
by (*auto simp: init-seg-of-def chain-subset-def Chains-def*)
have $\forall r \in R. \text{Refl } r$ **and** $\forall r \in R. \text{trans } r$ **and** $\forall r \in R. \text{antisym } r$ **and**
 $\forall r \in R. \text{Total } r$ **and** $\forall r \in R. \text{wf } (r - \text{Id})$ **and**
 $\bigwedge r. r \in R \implies \text{downset-on } (\text{Field } r) \ p$ **and**
 $\bigwedge r. r \in R \implies \text{extension-on } (\text{Field } r) \ r \ p$
using *Chains-wo* [*OF* $\langle R \in \text{Chains } I \rangle$] **by** (*simp-all add: order-on-defs*)
have *Refl* ($\bigcup R$) **using** $\langle \forall r \in R. \text{Refl } r \rangle$ **unfolding** *refl-on-def* **by** *fastforce*
moreover have *trans* ($\bigcup R$)
by (*rule chain-subset-trans-Union* [*OF subch* $\langle \forall r \in R. \text{trans } r \rangle$])

```

moreover have antisym ( $\bigcup R$ )
  by (rule chain-subset-antisym-Union [OF subch  $\langle \forall r \in R. \text{antisym } r \rangle$ ])
moreover have Total ( $\bigcup R$ )
  by (rule chain-subset-Total-Union [OF subch  $\langle \forall r \in R. \text{Total } r \rangle$ ])
moreover have wf ( $(\bigcup R) - \text{Id}$ )
proof –
  have  $(\bigcup R) - \text{Id} = \bigcup \{r - \text{Id} \mid r. r \in R\}$  by blast
  with  $\langle \forall r \in R. \text{wf } (r - \text{Id}) \rangle$  wf-Union-wf-init-segs [OF Chains-inits-DiffI [OF
Ris]]
  show ?thesis by fastforce
qed
ultimately have Well-order ( $\bigcup R$ ) by (simp add: order-on-defs)
moreover have  $\forall r \in R. r$  initial-segment-of  $\bigcup R$  using Ris
  by (simp add: Chains-init-seg-of-Union)
moreover have downset-on (Field ( $\bigcup R$ )) p
  by (rule downset-on-Union [OF  $\langle \bigwedge r. r \in R \implies \text{downset-on } (\text{Field } r) \text{ } p \rangle$ ])
moreover have extension-on (Field ( $\bigcup R$ )) ( $\bigcup R$ ) p
  by (rule chain-subset-extension-on-Union [OF subch  $\langle \bigwedge r. r \in R \implies \text{exten-}$ 
sion-on (Field r) r p ]])
ultimately show ?thesis
  using mono-Chains [OF I-init] and  $\langle R \in \text{Chains } I \rangle$ 
  by (simp (no-asm) add: I-def del: Field-Union) (metis Chains-wo)
qed
then have  $1: \exists u \in \text{Field } I. \forall r \in R. (r, u) \in I$  if  $R \in \text{Chains } I$  for R
  using that by (subst FI) blast

```

Zorn's Lemma yields a maximal wellorder *m*.

```

from Zorns-po-lemma [OF 0 1] obtain m ::  $('a \times 'a)$  set
  where Well-order m and downset-on (Field m) p and extension-on (Field m)
m p and
  max:  $\forall r. \text{Well-order } r \wedge \text{downset-on } (\text{Field } r) \text{ } p \wedge \text{extension-on } (\text{Field } r) \text{ } r \text{ } p \wedge$ 
 $(m, r) \in I \longrightarrow r = m$ 
  by (auto simp: FI)
have Field p  $\subseteq$  Field m
proof (rule ccontr)
  let ?Q = Field p – Field m
  assume  $\neg (\text{Field } p \subseteq \text{Field } m)$ 
  with assms [unfolded wf-eq-minimal, THEN spec, of ?Q]
    obtain x where  $x \in \text{Field } p$  and  $x \notin \text{Field } m$  and
    min:  $\forall y. (y, x) \in p \longrightarrow y \notin ?Q$  by blast

```

Add *x* as topmost element to *m*.

```

let ?s =  $\{(y, x) \mid y. y \in \text{Field } m\}$ 
let ?m = insert (x, x) m  $\cup$  ?s
have Fm: Field ?m = insert x (Field m) by (auto simp: Field-def)
have Refl m and trans m and antisym m and Total m and wf (m – Id)
  using  $\langle \text{Well-order } m \rangle$  by (simp-all add: order-on-defs)

```

We show that the extension is a wellorder.

have *Refl* ?*m* using $\langle \text{Refl } m \rangle$ *Fm* by (auto simp: refl-on-def)
 moreover have *trans* ?*m* using $\langle \text{trans } m \rangle$ $\langle x \notin \text{Field } m \rangle$
 unfolding *trans-def* *Field-def* *Domain-unfold* *Domain-converse* [symmetric]
 by blast
 moreover have *antisym* ?*m* using $\langle \text{antisym } m \rangle$ $\langle x \notin \text{Field } m \rangle$
 unfolding *antisym-def* *Field-def* *Domain-unfold* *Domain-converse* [symmetric]
 by blast
 moreover have *Total* ?*m* using $\langle \text{Total } m \rangle$ *Fm* by (auto simp: Relation.total-on-def)
 moreover have *wf* (?*m* - *Id*)
 proof -
 have *wf* ?*s* using $\langle x \notin \text{Field } m \rangle$
 by (simp add: wf-eq-minimal *Field-def* *Domain-unfold* *Domain-converse*
 [symmetric]) metis
 thus ?thesis using $\langle \text{wf } (m - \text{Id}) \rangle$ $\langle x \notin \text{Field } m \rangle$
 wf-subset [OF $\langle \text{wf } ?s \rangle$ *Diff-subset*]
 by (fastforce intro!: wf-Un simp add: Un-Diff *Field-def*)
 qed
 ultimately have *Well-order* ?*m* by (simp add: order-on-defs)
 moreover have *extension-on* (*Field* ?*m*) ?*m* *p*
 using $\langle \text{extension-on } (\text{Field } m) \ m \ p \rangle$ $\langle \text{downset-on } (\text{Field } m) \ p \rangle$
 by (subst *Fm*) (auto simp: extension-on-def dest: downset-onD)
 moreover have *downset-on* (*Field* ?*m*) *p*
 apply (subst *Fm*)
 using $\langle \text{downset-on } (\text{Field } m) \ p \rangle$ and *min*
 unfolding *downset-on-def* *Field-def* by blast
 moreover have $(m, ?m) \in I$
 using $\langle \text{Well-order } m \rangle$ and $\langle \text{Well-order } ?m \rangle$ and
 $\langle \text{downset-on } (\text{Field } m) \ p \rangle$ and $\langle \text{downset-on } (\text{Field } ?m) \ p \rangle$ and
 $\langle \text{extension-on } (\text{Field } m) \ m \ p \rangle$ and $\langle \text{extension-on } (\text{Field } ?m) \ ?m \ p \rangle$ and
 $\langle \text{Refl } m \rangle$ and $\langle x \notin \text{Field } m \rangle$
 by (auto simp: I-def init-seg-of-def refl-on-def)
 ultimately
 — This contradicts maximality of *m*:
 show *False* using *max* and $\langle x \notin \text{Field } m \rangle$ unfolding *Field-def* by blast
 qed
 have $p \subseteq m$
 using $\langle \text{Field } p \subseteq \text{Field } m \rangle$ and $\langle \text{extension-on } (\text{Field } m) \ m \ p \rangle$
 unfolding *Field-def* *extension-on-def* by auto fast
 with $\langle \text{Well-order } m \rangle$ show ?thesis by blast
 qed

Every well-founded relation can be extended to a total wellorder.

corollary *total-well-order-extension*:

assumes *wf* *p*
 shows $\exists w. p \subseteq w \wedge \text{Well-order } w \wedge \text{Field } w = \text{UNIV}$
 proof -
 from *well-order-extension* [OF *assms*] obtain *w*
 where $p \subseteq w$ and *wo*: *Well-order* *w* by blast
 let ?*A* = *UNIV* - *Field* *w*


```

from well-order-on [of ?A] obtain w' where wo': well-order-on ?A w' ..
have [simp]: Field w' = ?A using well-order-on-Well-order [OF wo'] by simp
have *: Field w  $\cap$  Field w' = {} by simp
let ?w = w  $\cup$  w'
have p  $\subseteq$  ?w using  $\langle p \subseteq w \rangle$  by (auto simp: Osum-def)
moreover have Well-order ?w using Osum-Well-order [OF * wo] and wo' by
simp
moreover have Field ?w = UNIV by (simp add: Field-Osum)
ultimately show ?thesis by blast
qed

```

corollary *well-order-on-extension*:

```

assumes wf p and Field p  $\subseteq$  A
shows  $\exists w. p \subseteq w \wedge$  well-order-on A w
proof –
from total-well-order-extension [OF  $\langle wf\ p \rangle$ ] obtain r
where p  $\subseteq$  r and wo: Well-order r and univ: Field r = UNIV by blast
let ?r = {(x, y). x  $\in$  A  $\wedge$  y  $\in$  A  $\wedge$  (x, y)  $\in$  r}
from  $\langle p \subseteq r \rangle$  have p  $\subseteq$  ?r using  $\langle Field\ p \subseteq A \rangle$  by (auto simp: Field-def)
have Refl r trans r antisym r Total r wf (r – Id)
using  $\langle Well-order\ r \rangle$  by (simp-all add: order-on-defs)
have refl-on A ?r using  $\langle Refl\ r \rangle$  by (auto simp: refl-on-def univ)
moreover have trans ?r using  $\langle trans\ r \rangle$ 
unfolding trans-def by blast
moreover have antisym ?r using  $\langle antisym\ r \rangle$ 
unfolding antisym-def by blast
moreover have total-on A ?r using  $\langle Total\ r \rangle$  by (simp add: total-on-def univ)
moreover have wf (?r – Id) by (rule wf-subset [OF  $\langle wf(r - Id) \rangle$ ]) blast
ultimately have well-order-on A ?r by (simp add: order-on-defs)
with  $\langle p \subseteq ?r \rangle$  show ?thesis by blast
qed

```

end

13 Theory of Ordinals and Cardinals

theory *Cardinals*

imports *Ordinal-Arithmetic Cardinal-Arithmetic Wellorder-Extension*

begin

end

14 Sets Strictly Bounded by an Infinite Cardinal

theory *Bounded-Set*

imports *Cardinals*

begin

```

typedef ('a, 'k) bset (λ set[-] › [22, 21] 21) =
  {A :: 'a set. |A| <o natLeq + c |UNIV :: 'k set|}
morphisms set-bset Abs-bset
by (rule exI[of - {}]) (auto simp: card-of-empty4 csum-def)

setup-lifting type-definition-bset

lift-definition map-bset ::
  ('a ⇒ 'b) ⇒ 'a set['k] ⇒ 'b set['k] is image
using card-of-image ordLeq-ordLess-trans by blast

lift-definition rel-bset ::
  ('a ⇒ 'b ⇒ bool) ⇒ 'a set['k] ⇒ 'b set['k] ⇒ bool is rel-set
  .

lift-definition bempty :: 'a set['k] is {}
by (auto simp: card-of-empty4 csum-def)

lift-definition binsert :: 'a ⇒ 'a set['k] ⇒ 'a set['k] is insert
using infinite-card-of-insert ordIso-ordLess-trans Field-card-of Field-natLeq UNIV-Plus-UNIV
  csum-def finite-Plus-UNIV-iff finite-insert finite-ordLess-infinite2 infinite-UNIV-nat
by metis

definition bsingleton where
  bsingleton x = binsert x bempty

lemma set-bset-to-set-bset: |A| <o natLeq + c |UNIV :: 'k set| ⇒
  set-bset (the-inv set-bset A :: 'a set['k]) = A
apply (rule f-the-inv-into-f[unfolded inj-on-def])
apply (simp add: set-bset-inject range-eqI Abs-bset-inverse[symmetric])
apply (rule range-eqI Abs-bset-inverse[symmetric] CollectI)+
  .

lemma rel-bset-aux-infinite:
  fixes a :: 'a set['k] and b :: 'b set['k]
  shows (∀ t ∈ set-bset a. ∃ u ∈ set-bset b. R t u) ∧ (∀ u ∈ set-bset b. ∃ t ∈ set-bset
a. R t u) ⟷
  ((BNF-Def.Grp {a. set-bset a ⊆ {(a, b). R a b}} (map-bset fst))-1-1 OO
  BNF-Def.Grp {a. set-bset a ⊆ {(a, b). R a b}} (map-bset snd)) a b (is ?L ⟷
?R)
proof
  assume ?L
  define R' :: ('a × 'b) set['k]
  where R' = the-inv set-bset (Collect (case-prod R) ∩ (set-bset a × set-bset b))
  (is - = the-inv set-bset ?L')
  have |?L'| <o natLeq + c |UNIV :: 'k set|
  unfolding csum-def Field-natLeq
  by (intro ordLeq-ordLess-trans[OF card-of-mono1[OF Int-lower2]]
  card-of-Times-ordLess-infinite)

```

```

    (simp, (transfer, simp add: csum-def Field-natLeq)+)
  hence *: set-bset R' = ?L' unfolding R'-def by (intro set-bset-to-set-bset)
  show ?R unfolding Grp-def relcompp.simps conversep.simps
  proof (intro CollectI case-prodI exI[of - a] exI[of - b] exI[of - R] conjI refl)
    from * show a = map-bset fst R' using conjunct1[OF <?L>]
    by (transfer, auto simp add: image-def Int-def split: prod.splits)
    from * show b = map-bset snd R' using conjunct2[OF <?L>]
    by (transfer, auto simp add: image-def Int-def split: prod.splits)
  qed (auto simp add: *)
next
  assume ?R thus ?L unfolding Grp-def relcompp.simps conversep.simps
  by transfer force
qed

bnf 'a set['k]
  map: map-bset
  sets: set-bset
  bd: natLeq +c card-suc | UNIV :: 'k set|
  wits: bempty
  rel: rel-bset
proof -
  show map-bset id = id by (rule ext, transfer) simp
next
  fix f g
  show map-bset (f o g) = map-bset f o map-bset g by (rule ext, transfer) auto
next
  fix X f g
  assume  $\bigwedge z. z \in \text{set-bset } X \implies f z = g z$ 
  then show map-bset f X = map-bset g X by transfer force
next
  fix f
  show set-bset o map-bset f = (') f o set-bset by (rule ext, transfer) auto
next
  fix X :: 'a set['k]
  have |set-bset X| <o natLeq +c |UNIV :: 'k set| by transfer blast
  then show |set-bset X| <o natLeq +c card-suc |UNIV :: 'k set|
    by (rule ordLess-ordLeq-trans[OF - csum-mono2[OF ordLess-imp-ordLeq[OF
card-suc-greater[OF card-of-card-order-on]]]])
next
  fix R S
  show rel-bset R OO rel-bset S  $\leq$  rel-bset (R OO S)
    by (rule predicate2I, transfer) (auto simp: rel-set-OO[symmetric])
next
  fix R :: 'a  $\Rightarrow$  'b  $\Rightarrow$  bool
  show rel-bset R = (( $\lambda x y. \exists z. \text{set-bset } z \subseteq \{(x, y). R x y\} \wedge$ 
    map-bset fst z = x  $\wedge$  map-bset snd z = y) :: 'a set['k]  $\Rightarrow$  'b set['k]  $\Rightarrow$  bool)
    by (simp add: rel-bset-def map-fun-def o-def rel-set-def
    rel-bset-aux-infinite[unfolded OO-Grp-alt])
next

```

```

fix x
assume  $x \in \text{set-bset } \text{bempty}$ 
then show False by transfer simp
qed (simp-all add: card-order-bd-fun Cinfinites-bd-fun regularCard-bd-fun)

lemma map-bset-bempty[simp]:  $\text{map-bset } f \text{ bempty} = \text{bempty}$ 
by transfer auto

lemma map-bset-binsert[simp]:  $\text{map-bset } f (\text{binsert } x \text{ } X) = \text{binsert } (f \text{ } x) (\text{map-bset } f \text{ } X)$ 
by transfer auto

lemma map-bset-bsingleton:  $\text{map-bset } f (\text{bsingleton } x) = \text{bsingleton } (f \text{ } x)$ 
unfolding bsingleton-def by simp

lemma bempty-not-binsert:  $\text{bempty} \neq \text{binsert } x \text{ } X \text{ } \text{binsert } x \text{ } X \neq \text{bempty}$ 
by (transfer, auto)+

lemma bempty-not-bsingleton[simp]:  $\text{bempty} \neq \text{bsingleton } x \text{ } \text{bsingleton } x \neq \text{bempty}$ 
unfolding bsingleton-def by (simp-all add: bempty-not-binsert)

lemma bsingleton-inj[simp]:  $\text{bsingleton } x = \text{bsingleton } y \longleftrightarrow x = y$ 
unfolding bsingleton-def by transfer auto

lemma rel-bsingleton[simp]:
   $\text{rel-bset } R (\text{bsingleton } x1) (\text{bsingleton } x2) = R \text{ } x1 \text{ } x2$ 
unfolding bsingleton-def
by transfer (auto simp: rel-set-def)

lemma rel-bset-bsingleton[simp]:
   $\text{rel-bset } R (\text{bsingleton } x1) = (\lambda X. X \neq \text{bempty} \wedge (\forall x2 \in \text{set-bset } X. R \text{ } x1 \text{ } x2))$ 
   $\text{rel-bset } R X (\text{bsingleton } x2) = (X \neq \text{bempty} \wedge (\forall x1 \in \text{set-bset } X. R \text{ } x1 \text{ } x2))$ 
unfolding bsingleton-def fun-eq-iff
by (transfer, force simp add: rel-set-def)+

lemma rel-bset-bempty[simp]:
   $\text{rel-bset } R \text{ bempty } X = (X = \text{bempty})$ 
   $\text{rel-bset } R Y \text{ bempty} = (Y = \text{bempty})$ 
by (transfer, simp add: rel-set-def)+

definition bset-of-option where
   $\text{bset-of-option} = \text{case-option } \text{bempty } \text{bsingleton}$ 

lift-definition bgraph ::  $('a \Rightarrow 'b \text{ option}) \Rightarrow ('a \times 'b) \text{ set}['a \text{ set}]$  is
   $\lambda f. \{(a, b). f \text{ } a = \text{Some } b\}$ 
proof –
  fix  $f :: 'a \Rightarrow 'b \text{ option}$ 
  have  $|\{(a, b). f \text{ } a = \text{Some } b\}| \leq o \mid \text{UNIV} :: 'a \text{ set}$ 

```

```

    by (rule surj-imp-ordLeq[of -  $\lambda x. (x, \text{the } (f x))$ ]) auto
  also have  $|UNIV :: 'a \text{ set}| <_o |UNIV :: 'a \text{ set set}|$ 
    by simp
  also have  $|UNIV :: 'a \text{ set set}| \leq_o \text{natLeq} + c |UNIV :: 'a \text{ set set}|$ 
    by (rule ordLeq-csum2) simp
  finally show  $|\{(a, b). f a = \text{Some } b\}| <_o \text{natLeq} + c |UNIV :: 'a \text{ set set}|$  .
qed

```

```

lemma rel-bset-False[simp]: rel-bset ( $\lambda x y. \text{False}$ )  $x y = (x = \text{bempty} \wedge y = \text{bempty})$ 
  by transfer (auto simp: rel-set-def)

```

```

lemma rel-bset-of-option[simp]:
  rel-bset  $R$  (bset-of-option  $x1$ ) (bset-of-option  $x2$ ) = rel-option  $R$   $x1$   $x2$ 
  unfolding bset-of-option-def bsingleton-def[abs-def]
  by transfer (auto simp: rel-set-def split: option.splits)

```

```

lemma rel-bgraph[simp]:
  rel-bset (rel-prod (=)  $R$ ) (bgraph  $f1$ ) (bgraph  $f2$ ) = rel-fun (=) (rel-option  $R$ )  $f1$ 
  f2
  apply transfer
  apply (auto simp: rel-fun-def rel-option-iff rel-set-def split: option.splits)
  using option.collapse apply fastforce+
  done

```

```

lemma set-bset-bsingleton[simp]:
  set-bset (bsingleton  $x$ ) =  $\{x\}$ 
  unfolding bsingleton-def by transfer auto

```

```

lemma binsert-absorb[simp]: binsert  $a$  (binsert  $a$   $x$ ) = binsert  $a$   $x$ 
  by transfer simp

```

```

lemma map-bset-eq-bempty-iff[simp]: map-bset  $f$   $X = \text{bempty} \longleftrightarrow X = \text{bempty}$ 
  by transfer auto

```

```

lemma map-bset-eq-bsingleton-iff[simp]:
  map-bset  $f$   $X = \text{bsingleton } x \longleftrightarrow (\text{set-bset } X \neq \{\}) \wedge (\forall y \in \text{set-bset } X. f y = x)$ 
  unfolding bsingleton-def by transfer auto

```

```

lift-definition bCollect ::  $('a \Rightarrow \text{bool}) \Rightarrow 'a \text{ set} ['a \text{ set}]$  is Collect
  apply (rule ordLeq-ordLess-trans[OF card-of-mono1[OF subset-UNIV]])
  apply (rule ordLess-ordLeq-trans[OF card-of-set-type])
  apply (rule ordLeq-csum2[OF card-of-Card-order])
  done

```

```

lift-definition bmember ::  $'a \Rightarrow 'a \text{ set} ['k] \Rightarrow \text{bool}$  is ( $\in$ ) .

```

```

lemma bmember-bCollect[simp]: bmember  $a$  (bCollect  $P$ ) =  $P a$ 
  by transfer simp

```

```

lemma bset-eq-iff:  $A = B \longleftrightarrow (\forall a. \text{bmember } a \ A = \text{bmember } a \ B)$ 
  by transfer auto

locale bset-lifting
begin

declare bset.rel-eq[relator-eq]
declare bset.rel-mono[relator-mono]
declare bset.rel-compp[symmetric, relator-distr]
declare bset.rel-transfer[transfer-rule]

lemma bset-quot-map[quot-map]:  $\text{Quotient } R \text{ Abs Rep } T \implies$ 
   $\text{Quotient } (\text{rel-bset } R) (\text{map-bset Abs}) (\text{map-bset Rep}) (\text{rel-bset } T)$ 
  unfolding Quotient-alt-def5 bset.rel-Grp[of UNIV, simplified, symmetric]
    bset.rel-conversep[symmetric] bset.rel-compp[symmetric]
  by (auto elim: bset.rel-mono-strong)

lemma set-relator-eq-onp [relator-eq-onp]:
   $\text{rel-bset } (\text{eq-onp } P) = \text{eq-onp } (\lambda A. \text{Ball } (\text{set-bset } A) \ P)$ 
  unfolding fun-eq-iff eq-onp-def by transfer (auto simp: rel-set-def)

end

end

```