

Zermelo Fraenkel Set Theory in Higher-Order Logic

Lawrence C. Paulson
Computer Laboratory
University of Cambridge

March 19, 2025

Abstract

This entry is a new formalisation of ZFC set theory in Isabelle/HOL. It is logically equivalent to Obua's HOLZF [2]; the point is to have the closest possible integration with the rest of Isabelle/HOL, minimising the amount of new notations and exploiting type classes.

There is a type V of sets and a function $elts :: V \Rightarrow V\ set$ mapping a set to its elements. Classes simply have type $V\ set$, and the predicate *small* identifies those classes that correspond to actual sets. Type classes connected with orders and lattices are used to minimise the amount of new notation for concepts such as the subset relation, union and intersection. Basic concepts are formalised: Cartesian products, disjoint sums, natural numbers, functions, etc.

More advanced set-theoretic concepts, such as transfinite induction, ordinals, cardinals and the transitive closure of a set, are also provided. The definition of addition and multiplication for general sets (not just ordinals) follows Kirby [1]. The development includes essential results about cardinal arithmetic. It also develops ordinal exponentiation, Cantor normal form and the concept of indecomposable ordinals. There are numerous results about order types.

The theory provides two type classes with the aim of facilitating developments that combine V with other Isabelle/HOL types: *embeddable*, the class of types that can be injected into V (including V itself as well as V^*V , $V\ list$, etc.), and *small*, the class of types that correspond to some ZF set.

Contents

1	The ZF Axioms, Ordinals and Transfinite Recursion	5
1.1	Syntax and axioms	5
1.2	Type classes and other basic setup	10
1.3	Successor function	17
1.4	Ordinals	17
1.4.1	Transitive sets	18
1.4.2	Zero, successor, sups	18
1.4.3	Induction, Linearity, etc.	20
1.4.4	The natural numbers	22
1.4.5	Limit ordinals	24
1.4.6	Properties of LEAST for ordinals	26
1.5	Transfinite Recursion and the V-levels	29
2	Cartesian products, Disjoint Sums, Ranks, Cardinals	32
2.1	Ordered Pairs	32
2.2	Generalized Cartesian product	34
2.3	Disjoint Sum	36
2.3.1	Equivalences for the injections and an elimination rule	36
2.3.2	Injection and freeness equivalences, for rewriting . . .	36
2.3.3	Applications of disjoint sums and pairs: general union theorems for small sets	38
2.4	Generalised function space and lambda	39
2.5	Transitive closure of a set	41
2.6	Rank of a set	45
2.7	Cardinal Numbers	49
2.7.1	Transitive Closure and VWO	49
2.7.2	Relation VWF	51
2.8	Order types	52
2.8.1	<i>ordermap</i> preserves the orderings in both directions .	53
2.9	More advanced <i>ordertype</i> and <i>ordermap</i> results	55
2.10	Cardinality of an arbitrary HOL set	71
2.11	Cardinality of a set	71
2.12	Cardinality of a set	74
2.13	Transfinite recursion for definitions based on the three cases of ordinals	75
2.14	Cardinal Addition	76
2.14.1	Cardinal addition is commutative	76
2.14.2	Cardinal addition is associative	76
2.15	Cardinal multiplication	78
2.15.1	Cardinal multiplication is commutative	78
2.15.2	Cardinal multiplication is associative	78
2.15.3	Cardinal multiplication distributes over addition . . .	79

2.16	Some inequalities for multiplication	79
2.17	The finite cardinals	81
2.18	Infinite cardinals	82
2.19	Toward's Kunen's Corollary 10.13 (1)	86
2.20	The Aleph-sequence	87
2.21	The ordinal $\omega 1$	90
3	Addition and Multiplication of Sets	91
3.1	Generalised Addition	91
3.1.1	Addition is a monoid	91
3.1.2	Deeper properties of addition	95
3.1.3	Cancellation / set subtraction	99
3.2	Generalised Difference	103
3.3	Generalised Multiplication	105
3.3.1	Proposition 4.3	106
3.3.2	Proposition 4.4-5	109
3.3.3	Theorem 4.6	111
3.3.4	Theorem 4.7	115
3.4	Ordertype properties	123
4	Exponentiation of ordinals	126
5	Cantor Normal Form	138
5.1	Cantor normal form	138
5.2	Simplified Cantor normal form	148
5.3	Indecomposable ordinals	154
5.4	From ordinals to order types	167
6	Type Classes for ZFC	169
6.1	The class of embeddable types	169
6.2	The class of small types	172
7	ZF sets corresponding to \mathbb{R} and \mathbb{C} and the cardinality of the continuum	175
7.1	Making the embedding from the type class explicit	176
7.2	The cardinality of the continuum	176
7.3	Countable and uncountable sets	178
8	Acknowledgements	180

```

theory ZFC-Library
  imports HOL-Library.Countable-Set HOL-Library.Equipollence HOL-Cardinals.Cardinals

begin

Equipollence and Lists.

lemma countable-iff-lepoll: countable  $A \longleftrightarrow A \lesssim (UNIV :: \text{nat set})$ 
  by (auto simp: countable-def lepoll-def)

lemma infinite-times-epoll-self:
  assumes infinite  $A$  shows  $A \times A \approx A$ 
  by (simp add: Times-same-infinite-bij-betw assms eqpoll-def)

lemma infinite-finite-times-lepoll-self:
  assumes infinite  $A$  finite  $B$  shows  $A \times B \lesssim A$ 
proof -
  have  $B \lesssim A$ 
    by (simp add: assms finite-lepoll-infinite)
  then have  $A \times B \lesssim A \times A$ 
    by (simp add: subset-imp-lepoll times-lepoll-mono)
  also have  $\dots \approx A$ 
    by (simp add: <infinite  $A$ > infinite-times-epoll-self)
  finally show ?thesis .
qed

lemma lists-n-lepoll-self:
  assumes infinite  $A$  shows  $\{l \in \text{lists } A. \text{length } l = n\} \lesssim A$ 
proof (induction  $n$ )
  case 0
  have  $\{l \in \text{lists } A. \text{length } l = 0\} = \{\}\}
    by auto
  then show ?case
    by (metis Set.set-insert assms ex-in-conv finite.emptyI singleton-lepoll)
next
  case (Suc  $n$ )
  have  $\{l \in \text{lists } A. \text{length } l = \text{Suc } n\} = (\bigcup_{x \in A}. \bigcup l \in \{l \in \text{lists } A. \text{length } l = n\}.$ 
 $\{x\ \#\ l\})$ 
    by (auto simp: length-Suc-conv)
  also have  $\dots \lesssim A \times \{l \in \text{lists } A. \text{length } l = n\}$ 
    unfolding lepoll-iff
    by (rule-tac  $x = \lambda(x, l). \text{Cons } x \ l$  in exI) auto
  also have  $\dots \lesssim A$ 
  proof (cases finite  $\{l \in \text{lists } A. \text{length } l = n\}$ )
  case True
  then show ?thesis
    using assms infinite-finite-times-lepoll-self by blast
next
  case False
  have  $A \times \{l \in \text{lists } A. \text{length } l = n\} \lesssim A \times A$$ 
```

```

    by (simp add: Suc.IH subset-imp-lepoll times-lepoll-mono)
  also have ...  $\approx$  A
    by (simp add: assms infinite-times-epoll-self)
  finally show ?thesis .
qed
finally show ?case .
qed

lemma infinite-epoll-lists:
  assumes infinite A shows lists A  $\approx$  A
proof -
  have lists A  $\lesssim$  Sigma UNIV ( $\lambda n. \{l \in \text{lists } A. \text{length } l = n\}$ )
    unfolding lepoll-iff
    by (rule-tac x=snd in exI) (auto simp: in-listsI snd-image-Sigma)
  also have ...  $\lesssim$  (UNIV::nat set)  $\times$  A
    by (rule Sigma-lepoll-mono) (auto simp: lists-n-lepoll-self assms)
  also have ...  $\lesssim$  A  $\times$  A
    by (metis assms infinite-le-lepoll order-refl subset-imp-lepoll times-lepoll-mono)
  also have ...  $\approx$  A
    by (simp add: assms infinite-times-epoll-self)
  finally show ?thesis
    by (simp add: lepoll-antisym lepoll-lists)
qed

end

```

1 The ZF Axioms, Ordinals and Transfinite Recursion

```

theory ZFC-in-HOL
  imports ZFC-Library

```

```
begin
```

1.1 Syntax and axioms

```
hide-const (open) list.set Sum subset
```

```
unbundle lattice-syntax
```

```
typedecl V
```

Presentation refined by Dmitriy Traytel

```
axiomatization elts :: V  $\Rightarrow$  V set
```

```
where ext [intro?]: elts x = elts y  $\implies$  x=y
```

```
and down-raw: Y  $\subseteq$  elts x  $\implies$  Y  $\in$  range elts
```

```
and Union-raw: X  $\in$  range elts  $\implies$  Union (elts ' X)  $\in$  range elts
```

```
and Pow-raw: X  $\in$  range elts  $\implies$  inv elts ' Pow X  $\in$  range elts
```

and *replacement-raw*: $X \in \text{range } \text{elts} \implies f \text{ ' } X \in \text{range } \text{elts}$
and *inf-raw*: $\text{range } (g :: \text{nat} \Rightarrow V) \in \text{range } \text{elts}$
and *foundation*: $\text{wf } \{(x,y). x \in \text{elts } y\}$

lemma *mem-not-refl* [*simp*]: $i \notin \text{elts } i$
using *wf-not-refl* [*OF foundation*] **by** *force*

lemma *mem-not-sym*: $\neg (x \in \text{elts } y \wedge y \in \text{elts } x)$
using *wf-not-sym* [*OF foundation*] **by** *force*

A set is small if it can be injected into the extension of a V-set.

definition *small* :: 'a set \Rightarrow bool
where *small* $X \equiv \exists V\text{-of} :: 'a \Rightarrow V. \text{inj-on } V\text{-of } X \wedge V\text{-of ' } X \in \text{range } \text{elts}$

lemma *small-empty* [*iff*]: *small* $\{\}$
by (*simp add: small-def down-raw*)

lemma *small-iff-range*: *small* $X \longleftrightarrow X \in \text{range } \text{elts}$
apply (*simp add: small-def*)
by (*metis inj-on-id2 replacement-raw the-inv-into-onto*)

lemma *small-epoll*: *small* $A \longleftrightarrow (\exists x. \text{elts } x \approx A)$
unfolding *small-def* **by** (*metis UNIV-I bij-betw-def eqpoll-def eqpoll-sym imageE image-eqI*)

Small classes can be mapped to sets.

definition *set* :: $V \text{ set} \Rightarrow V$
where *set* $X \equiv (\text{if } \text{small } X \text{ then } \text{inv } \text{elts } X \text{ else } \text{inv } \text{elts } \{\})$

lemma *set-of-elts* [*simp*]: *set* ($\text{elts } x$) = x
by (*force simp add: ext set-def f-inv-into-f small-def*)

lemma *elts-of-set* [*simp*]: $\text{elts } (\text{set } X) = (\text{if } \text{small } X \text{ then } X \text{ else } \{\})$
by (*simp add: ZFC-in-HOL.set-def down-raw f-inv-into-f small-iff-range*)

lemma *down*: $Y \subseteq \text{elts } x \implies \text{small } Y$
by (*simp add: down-raw small-iff-range*)

lemma *Union* [*intro*]: *small* $X \implies \text{small } (\text{Union } (\text{elts ' } X))$
by (*simp add: Union-raw small-iff-range*)

lemma *Pow*: *small* $X \implies \text{small } (\text{set ' } \text{Pow } X)$
unfolding *small-iff-range* **using** *Pow-raw set-def down* **by** *force*

declare *replacement-raw* [*intro,simp*]

lemma *replacement* [*intro,simp*]:
assumes *small* X
shows *small* ($f \text{ ' } X$)

```

proof –
  let ?A = inv-into X f ‘ (f ‘ X)
  have AX: ?A ⊆ X
    by (simp add: image-subsetI inv-into-into)
  have inj: inj-on f ?A
    by (simp add: f-inv-into-f inj-on-def)
  have injo: inj-on (inv-into X f) (f ‘ X)
    using inj-on-inv-into by blast
  have ∃ V-of. inj-on V-of (f ‘ X) ∧ V-of ‘ f ‘ X ∈ range elts
    if inj-on V-of X and V-of ‘ X = elts x
    for V-of :: ‘a ⇒ V and x
  proof (intro exI conjI)
    show inj-on (V-of ◦ inv-into X f) (f ‘ X)
      by (meson ⟨inv-into X f ‘ f ‘ X ⊆ X⟩ comp-inj-on inj-on-subset injo that)
    have (λx. V-of (inv-into X f (f x))) ‘ X = elts (set (V-of ‘ ?A))
      by (metis AX down elts-of-set image-image image-mono that(2))
    then show (V-of ◦ inv-into X f) ‘ f ‘ X ∈ range elts
      by (metis image-comp image-image rangeI)
    qed
  then show ?thesis
    using assms by (auto simp: small-def)
qed

```

```

lemma small-image-iff [simp]: inj-on f A ⇒ small (f ‘ A) ↔ small A
  by (metis replacement the-inv-into-onto)

```

A little bootstrapping is needed to characterise *small* for sets of arbitrary type.

```

lemma inf: small (range (g :: nat ⇒ V))
  by (simp add: inf-raw small-iff-range)

```

```

lemma small-image-nat-V [simp]: small (g ‘ N) for g :: nat ⇒ V
  by (metis (mono-tags, opaque-lifting) down elts-of-set image-iff inf rangeI subsetI)

```

```

lemma Finite-V:
  fixes X :: V set
  assumes finite X shows small X
  using ex-bij-betw-nat-finite [OF assms] unfolding bij-betw-def by (metis small-image-nat-V)

```

```

lemma small-insert-V:
  fixes X :: V set
  assumes small X
  shows small (insert a X)
proof (cases finite X)
  case True
    then show ?thesis
      by (simp add: Finite-V)
next
  case False

```

```

show ?thesis
  using infinite-imp-bij-betw2 [OF False]
  by (metis replacement Un-insert-right assms bij-betw-imp-surj-on sup-bot.right-neutral)
qed

```

```

lemma small-UN-V [simp,intro]:
  fixes B :: 'a  $\Rightarrow$  V set
  assumes X: small X and B:  $\bigwedge x. x \in X \implies \text{small } (B x)$ 
  shows small ( $\bigcup_{x \in X}. B x$ )
proof -
  have ( $\bigcup (\text{elts } '(\lambda x. \text{ZFC-in-HOL.set } (B x)) ' X)$ ) = ( $\bigcup (B ' X)$ )
    using B by force
  then show ?thesis
    using Union [OF replacement [OF X, of  $\lambda x. \text{ZFC-in-HOL.set } (B x)$ ]] by simp
qed

```

```

definition vinsert where vinsert x y  $\equiv$  set (insert x (elts y))

```

```

lemma elts-vinsert [simp]: elts (vinsert x y) = insert x (elts y)
  using down small-insert-V vinsert-def by auto

```

```

definition succ where succ x  $\equiv$  vinsert x x

```

```

lemma elts-succ [simp]: elts (succ x) = insert x (elts x)
  by (simp add: succ-def)

```

```

lemma finite-imp-small:
  assumes finite X shows small X
  using assms
proof induction
  case empty
  then show ?case
    by simp
next
  case (insert a X)
  then obtain V-of u where u: inj-on V-of X V-of ' X = elts u
    by (meson small-def image-iff)
  show ?case
    unfolding small-def
  proof (intro exI conjI)
    show inj-on (V-of(a:=u)) (insert a X)
      using u
      apply (clarsimp simp add: inj-on-def)
      by (metis image-eqI mem-not-refl)
    have (V-of(a:=u)) ' insert a X = elts (vinsert u u)
      using insert.hyps(2) u(2) by auto
    then show (V-of(a:=u)) ' insert a X  $\in$  range elts
      by (blast intro: elim:)
  qed

```


qed

lemma *small-insert*:

assumes *small X*

shows *small (insert a X)*

proof (*cases finite X*)

case *True*

then show *?thesis*

by (*simp add: finite-imp-small*)

next

case *False*

show *?thesis*

using *infinite-imp-bij-betw2 [OF False]*

by (*metis replacement Un-insert-right assms bij-betw-imp-surj-on sup-bot.right-neutral*)

qed

lemma *smaller-than-small*:

assumes *small A B \subseteq A* **shows** *small B*

using *assms*

by (*metis down elts-of-set image-mono small-def small-iff-range subset-inj-on*)

lemma *small-insert-iff [iff]*: *small (insert a X) \longleftrightarrow small X*

by (*meson small-insert smaller-than-small subset-insertI*)

lemma *small-iff*: *small X \longleftrightarrow ($\exists x. X = \text{elts } x$)*

by (*metis down elts-of-set subset-refl*)

lemma *small-elts [iff]*: *small (elts x)*

by (*auto simp: small-iff*)

lemma *small-diff [iff]*: *small (elts a - X)*

by (*meson Diff-subset down*)

lemma *small-set [simp]*: *small (list.set xs)*

by (*simp add: ZFC-in-HOL.finite-imp-small*)

lemma *small-upair*: *small {x,y}*

by *simp*

lemma *small-Un-elts*: *small (elts x \cup elts y)*

using *Union [OF small-upair]* **by** *auto*

lemma *small-eqcong*: $\llbracket \text{small } X; X \approx Y \rrbracket \implies \text{small } Y$

by (*metis bij-betw-imp-surj-on eqpoll-def replacement*)

lemma *lepoll-small*: $\llbracket \text{small } Y; X \lesssim Y \rrbracket \implies \text{small } X$

by (*meson lepoll-iff replacement smaller-than-small*)

lemma *big-UNIV [simp]*: $\neg \text{small } (\text{UNIV}::V \text{ set})$ (**is** $\neg \text{small } ?U$)

```

proof
  assume small ?U
  then have small A for A :: V set
    by (metis (full-types) UNIV-I down small-iff subsetI)
  then have range elts = UNIV
    by (meson small-iff surj-def)
  then show False
    by (metis Cantors-theorem Pow-UNIV)
qed

```

```

lemma inj-on-set: inj-on set (Collect small)
  by (metis elts-of-set inj-onI mem-Collect-eq)

```

```

lemma set-injective [simp]:  $\llbracket \text{small } X; \text{small } Y \rrbracket \implies \text{set } X = \text{set } Y \longleftrightarrow X=Y$ 
  by (metis elts-of-set)

```

1.2 Type classes and other basic setup

```

instantiation V :: zero
begin
  definition zero-V where 0  $\equiv$  set {}
  instance ..
end

```

```

lemma elts-0 [simp]:  $\text{elts } 0 = \{\}$ 
  by (simp add: zero-V-def)

```

```

lemma set-empty [simp]:  $\text{set } \{\} = 0$ 
  by (simp add: zero-V-def)

```

```

instantiation V :: one
begin
  definition one-V where 1  $\equiv$  succ 0
  instance ..
end

```

```

lemma elts-1 [simp]:  $\text{elts } 1 = \{0\}$ 
  by (simp add: one-V-def)

```

```

lemma insert-neq-0 [simp]:  $\text{set } (\text{insert } a \ X) = 0 \longleftrightarrow \neg \text{small } X$ 
  unfolding zero-V-def
  by (metis elts-of-set empty-not-insert set-of-elts small-insert-iff)

```

```

lemma elts-eq-empty-iff [simp]:  $\text{elts } x = \{\} \longleftrightarrow x=0$ 
  by (auto simp: ZFC-in-HOL.ext)

```

```

instantiation V :: distrib-lattice
begin

```

definition *inf-V* **where** $\text{inf-V } x \ y \equiv \text{set } (\text{elts } x \cap \text{elts } y)$

definition *sup-V* **where** $\text{sup-V } x \ y \equiv \text{set } (\text{elts } x \cup \text{elts } y)$

definition *less-eq-V* **where** $\text{less-eq-V } x \ y \equiv \text{elts } x \subseteq \text{elts } y$

definition *less-V* **where** $\text{less-V } x \ y \equiv \text{less-eq } x \ y \wedge x \neq (y::V)$

instance

proof

show $(x < y) = (x \leq y \wedge \neg y \leq x)$ **for** $x :: V$ **and** $y :: V$
using *ext less-V-def less-eq-V-def* **by** *auto*

show $x \leq x$ **for** $x :: V$

by (*simp add: less-eq-V-def*)

show $x \leq z$ **if** $x \leq y$ $y \leq z$ **for** $x \ y \ z :: V$

using *that by (auto simp: less-eq-V-def)*

show $x = y$ **if** $x \leq y$ $y \leq x$ **for** $x \ y :: V$

using *that by (simp add: less-eq-V-def)*

show $\text{inf } x \ y \leq x$ **for** $x \ y :: V$

by (*metis down elts-of-set inf-V-def inf-sup-ord(1) less-eq-V-def*)

show $\text{inf } x \ y \leq y$ **for** $x \ y :: V$

by (*metis Int-lower2 down elts-of-set inf-V-def less-eq-V-def*)

show $x \leq \text{inf } y \ z$ **if** $x \leq y$ $x \leq z$ **for** $x \ y \ z :: V$

proof –

have *small* $(\text{elts } y \cap \text{elts } z)$

by (*meson down inf.cobounded1*)

then show *?thesis*

using *elts-of-set inf-V-def less-eq-V-def that by auto*

qed

show $x \leq x \sqcup y$ $y \leq x \sqcup y$ **for** $x \ y :: V$

by (*simp-all add: less-eq-V-def small-Un-elts sup-V-def*)

show $\text{sup } y \ z \leq x$ **if** $y \leq x$ $z \leq x$ **for** $x \ y \ z :: V$

using *less-eq-V-def sup-V-def that by auto*

show $\text{sup } x \ (\text{inf } y \ z) = \text{inf } (x \sqcup y) \ (\text{sup } x \ z)$ **for** $x \ y \ z :: V$

proof –

have *small* $(\text{elts } y \cap \text{elts } z)$

by (*meson down inf.cobounded2*)

then show *?thesis*

by (*simp add: Un-Int-distrib inf-V-def small-Un-elts sup-V-def*)

qed

qed

end

lemma *V-equalityI* [*intro*]: $(\bigwedge x. x \in \text{elts } a \longleftrightarrow x \in \text{elts } b) \implies a = b$

by (*meson dual-order.antisym less-eq-V-def subsetI*)

lemma *vsubsetI* [*intro*]: $(\bigwedge x. x \in \text{elts } a \implies x \in \text{elts } b) \implies a \leq b$

by (*simp add: less-eq-V-def subsetI*)

lemma *vsubsetD* [*elim, intro?*]: $a \leq b \implies c \in \text{elts } a \implies c \in \text{elts } b$
using *less-eq-V-def* **by** *auto*

lemma *rev-vsubsetD*: $c \in \text{elts } a \implies a \leq b \implies c \in \text{elts } b$
— The same, with reversed premises for use with *erule* – cf. $\llbracket ?P; ?P \longrightarrow ?Q \rrbracket \implies ?Q$.
by (*rule vsubsetD*)

lemma *vsubsetCE* [*elim, no-atp*]: $a \leq b \implies (c \notin \text{elts } a \implies P) \implies (c \in \text{elts } b \implies P) \implies P$
— Classical elimination rule.
using *vsubsetD* **by** *blast*

lemma *set-image-le-iff*: $\text{small } A \implies \text{set } (f \text{ ‘ } A) \leq B \iff (\forall x \in A. f x \in \text{elts } B)$
by *auto*

lemma *eq0-iff*: $x = 0 \iff (\forall y. y \notin \text{elts } x)$
by *auto*

lemma *less-eq-V-0-iff* [*simp*]: $x \leq 0 \iff x = 0$ **for** $x::V$
by *auto*

lemma *subset-iff-less-eq-V*:
assumes *small B* **shows** $A \subseteq B \iff \text{set } A \leq \text{set } B \wedge \text{small } A$
using *assms down small-iff* **by** *auto*

lemma *small-Collect* [*simp*]: $\text{small } A \implies \text{small } \{x \in A. P x\}$
by (*simp add: smaller-than-small*)

lemma *small-Union-iff*: $\text{small } (\bigcup (\text{elts ‘ } X)) \iff \text{small } X$
proof
show *small X*
if *small* $(\bigcup (\text{elts ‘ } X))$
proof –
have $X \subseteq \text{set ‘ } \text{Pow } (\bigcup (\text{elts ‘ } X))$
by *fastforce*
then show *?thesis*
using *Pow subset-iff-less-eq-V* **that** **by** *auto*
qed
qed *auto*

lemma *not-less-0* [*iff*]:
fixes $x::V$ **shows** $\neg x < 0$
by (*simp add: less-eq-V-def less-le-not-le*)

lemma *le-0* [*iff*]:
fixes $x::V$ **shows** $0 \leq x$
by *auto*

lemma *min-0L* [*simp*]: $\min 0 n = 0$ **for** $n :: V$
by (*simp add: min-absorb1*)

lemma *min-0R* [*simp*]: $\min n 0 = 0$ **for** $n :: V$
by (*simp add: min-absorb2*)

lemma *neq0-conv*: $\bigwedge n::V. n \neq 0 \longleftrightarrow 0 < n$
by (*simp add: less-V-def*)

definition *VPow* :: $V \Rightarrow V$
where $VPow\ x \equiv set\ (set\ ' Pow\ (elts\ x))$

lemma *VPow-iff* [*iff*]: $y \in elts\ (VPow\ x) \longleftrightarrow y \leq x$
using *down Pow*
apply (*auto simp: VPow-def less-eq-V-def*)
using *less-eq-V-def apply fastforce*
done

lemma *VPow-le-VPow-iff* [*simp*]: $VPow\ a \leq VPow\ b \longleftrightarrow a \leq b$
by *auto*

lemma *elts-VPow*: $elts\ (VPow\ x) = set\ ' Pow\ (elts\ x)$
by (*auto simp: VPow-def Pow*)

lemma *small-sup-iff* [*simp*]: $small\ (X \cup Y) \longleftrightarrow small\ X \wedge small\ Y$ **for** $X::V\ set$
by (*metis down elts-of-set small-Un-elts sup-ge1 sup-ge2*)

lemma *elts-sup-iff* [*simp*]: $elts\ (x \sqcup y) = elts\ x \cup elts\ y$
by (*simp add: sup-V-def*)

lemma *trad-foundation*:
assumes $z: z \neq 0$ **shows** $\exists w. w \in elts\ z \wedge w \sqcap z = 0$
using *foundation assms*
by (*simp add: wf-eq-minimal*) (*metis Int-emptyI equals0I inf-V-def set-of-elts zero-V-def*)

instantiation $V :: Sup$

begin

definition *Sup-V* **where** $Sup-V\ X \equiv if\ small\ X\ then\ set\ (Union\ (elts\ ' X))\ else\ 0$

instance ..

end

instantiation $V :: Inf$

begin

definition *Inf-V* **where** $Inf-V\ X \equiv if\ X = \{\}\ then\ 0\ else\ set\ (Inter\ (elts\ ' X))$

instance ..

end

lemma *V-disjoint-iff*: $x \sqcap y = 0 \longleftrightarrow \text{elts } x \cap \text{elts } y = \{\}$
by (*metis down elts-of-set inf-V-def inf-le1 zero-V-def*)

I've no idea why *bdd-above* is treated differently from *bdd-below*, but anyway

lemma *bdd-above-iff-small* [*simp*]: *bdd-above* $X = \text{small } X$ **for** $X :: V \text{ set}$

proof

show *small* X **if** *bdd-above* X

proof –

obtain a **where** $\forall x \in X. x \leq a$

using *that* $\langle \text{bdd-above } X \rangle$ *bdd-above-def* **by** *blast*

then show *small* X

by (*meson VPow-iff* $\langle \forall x \in X. x \leq a \rangle$ *down subsetI*)

qed

show *bdd-above* X

if *small* X

proof –

have $\forall x \in X. x \leq \bigsqcup X$

by (*simp add: SUP-upper Sup-V-def Union less-eq-V-def that*)

then show *?thesis*

by (*meson bdd-above-def*)

qed

qed

instantiation $V :: \text{conditionally-complete-lattice}$

begin

definition *bdd-below-V* **where** *bdd-below-V* $X \equiv X \neq \{\}$

instance

proof

show $\bigsqcap X \leq x$ **if** $x \in X$ *bdd-below* X

for $x :: V$ **and** $X :: V \text{ set}$

using *that* **by** (*auto simp: bdd-below-V-def Inf-V-def split: if-split-asm*)

show $z \leq \bigsqcap X$

if $X \neq \{\} \wedge x. x \in X \implies z \leq x$

for $X :: V \text{ set}$ **and** $z :: V$

using *that*

apply (*clarsimp simp add: bdd-below-V-def Inf-V-def less-eq-V-def split: if-split-asm*)

by (*meson INT-subset-iff down eq-refl equals0I*)

show $x \leq \bigsqcup X$ **if** $x \in X$ **and** *bdd-above* X **for** $x :: V$ **and** $X :: V \text{ set}$

using *that* *Sup-V-def* **by** *auto*

show $\bigsqcup X \leq (z :: V)$ **if** $X \neq \{\} \wedge x. x \in X \implies x \leq z$ **for** $X :: V \text{ set}$ **and** $z :: V$

using *that* **by** (*simp add: SUP-least Sup-V-def less-eq-V-def*)

qed

end

lemma *Sup-upper*: $\llbracket x \in A; \text{small } A \rrbracket \implies x \leq \bigsqcup A$ **for** $A :: V \text{ set}$

by (auto simp: Sup-V-def SUP-upper Union less-eq-V-def)

lemma *Sup-least*:

fixes $z::V$ shows $(\bigwedge x. x \in A \implies x \leq z) \implies \bigsqcup A \leq z$
 by (auto simp: Sup-V-def SUP-least less-eq-V-def)

lemma *Sup-empty* [simp]: $\bigsqcup \{\} = (0::V)$

using *Sup-V-def* by auto

lemma *elts-Sup* [simp]: $small\ X \implies elts\ (\bigsqcup\ X) = \bigcup (elts\ 'X)$

by (auto simp: Sup-V-def)

lemma *sup-V-0-left* [simp]: $0 \sqcup a = a$ and *sup-V-0-right* [simp]: $a \sqcup 0 = a$ for $a::V$

by auto

lemma *Sup-V-insert*:

fixes $x::V$ assumes *small A* shows $\bigsqcup (insert\ x\ A) = x \sqcup \bigsqcup A$
 by (simp add: assms cSup-insert-If)

lemma *Sup-Un-distrib*: $\llbracket small\ A; small\ B \rrbracket \implies \bigsqcup (A \cup B) = \bigsqcup A \sqcup \bigsqcup B$ for $A::V$ set

by auto

lemma *SUP-sup-distrib*:

fixes $f::V \Rightarrow V$
 shows $small\ A \implies (\bigsqcup_{x \in A}. f\ x \sqcup g\ x) = \bigsqcup (f\ 'A) \sqcup \bigsqcup (g\ 'A)$
 by (force simp:)

lemma *SUP-const* [simp]: $(\bigsqcup y \in A. a) = (if\ A = \{\} then\ (0::V) else\ a)$

by simp

lemma *cSUP-subset-mono*:

fixes $f::'a \Rightarrow V$ set and $g::'a \Rightarrow V$ set
 shows $\llbracket A \subseteq B; \bigwedge x. x \in A \implies f\ x \leq g\ x \rrbracket \implies \bigsqcup (f\ 'A) \leq \bigsqcup (g\ 'B)$
 by (simp add: SUP-subset-mono)

lemma *mem-Sup-iff* [iff]: $x \in elts\ (\bigsqcup X) \iff x \in \bigcup (elts\ 'X) \wedge small\ X$

using *Sup-V-def* by auto

lemma *cSUP-UNION*:

fixes $B::V \Rightarrow V$ set and $f::V \Rightarrow V$
 assumes *ne: small A* and *bdd-UN: small* $(\bigcup_{x \in A}. f\ 'B\ x)$
 shows $\bigsqcup (f\ '(\bigcup_{x \in A}. B\ x)) = \bigsqcup ((\lambda x. \bigsqcup (f\ 'B\ x))\ 'A)$

proof –

have *bdd*: $\bigwedge x. x \in A \implies small\ (f\ 'B\ x)$

using *bdd-UN subset-iff-less-eq-V*

by (meson *SUP-upper smaller-than-small*)

then have *bdd2*: $small\ ((\lambda x. \bigsqcup (f\ 'B\ x))\ 'A)$

using $ne(1)$ **by** *blast*
have $\sqcup (f \text{ ' } (\bigcup_{x \in A}. B \ x)) \leq \sqcup ((\lambda x. \sqcup (f \text{ ' } B \ x)) \text{ ' } A)$
using *assms* **by** (*fastforce simp add: intro!: cSUP-least intro: cSUP-upper2 simp: bdd2 bdd*)
moreover **have** $\sqcup ((\lambda x. \sqcup (f \text{ ' } B \ x)) \text{ ' } A) \leq \sqcup (f \text{ ' } (\bigcup_{x \in A}. B \ x))$
using *assms* **by** (*fastforce simp add: intro!: cSUP-least intro: cSUP-upper simp: image-UN bdd-UN*)
ultimately show *?thesis*
by (*rule order-antisym*)
qed

lemma *Sup-subset-mono*: $small \ B \implies A \subseteq B \implies Sup \ A \leq Sup \ B$ **for** $A :: V \ set$
by *auto*

lemma *Sup-le-iff*: $small \ A \implies Sup \ A \leq a \longleftrightarrow (\forall x \in A. x \leq a)$ **for** $A :: V \ set$
by *auto*

lemma *SUP-le-iff*: $small \ (f \text{ ' } A) \implies \sqcup (f \text{ ' } A) \leq u \longleftrightarrow (\forall x \in A. f \ x \leq u)$ **for** $f :: V \Rightarrow V$
by *blast*

lemma *Sup-eq-0-iff* [*simp*]: $\sqcup A = 0 \longleftrightarrow A \subseteq \{0\} \vee \neg \ small \ A$ **for** $A :: V \ set$
using *Sup-upper* **by** *fastforce*

lemma *Sup-Union-commute*:
fixes $f :: V \Rightarrow V \ set$
assumes $small \ A \ \wedge \ x. x \in A \implies small \ (f \ x)$
shows $\sqcup (\bigcup_{x \in A}. f \ x) = (\sqcup_{x \in A}. \sqcup (f \ x))$
using *assms*
by (*force simp: subset-iff-less-eq-V intro!: antisym*)

lemma *Sup-eq-Sup*:
fixes $B :: V \ set$
assumes $B \subseteq A$ *small* A **and** $*$: $\wedge x. x \in A \implies \exists y \in B. x \leq y$
shows $Sup \ A = Sup \ B$

proof –
have *small* B
using *assms subset-iff-less-eq-V* **by** *auto*
moreover **have** $\exists y \in B. u \in \text{elts } y$
if $x \in A \ u \in \text{elts } x$ **for** $u \ x$
using *that ** **by** *blast*
moreover **have** $\exists x \in A. v \in \text{elts } x$
if $y \in B \ v \in \text{elts } y$ **for** $v \ y$
using *that* $\langle B \subseteq A \rangle$ **by** *blast*
ultimately show *?thesis*
using *assms* **by** *auto*

qed

1.3 Successor function

lemma *vinsert-not-empty* [*simp*]: $vinsert\ a\ A \neq 0$
 and *empty-not-vinsert* [*simp*]: $0 \neq vinsert\ a\ A$
 by (*auto simp: vinsert-def*)

lemma *succ-not-0* [*simp*]: $succ\ n \neq 0$ **and** *zero-not-succ* [*simp*]: $0 \neq succ\ n$
 by (*auto simp: succ-def*)

instantiation $V :: zero-neq-one$

begin

instance

by *intro-classes (metis elts-0 elts-succ empty-iff insert-iff one-V-def set-of-elts)*
end

instantiation $V :: zero-less-one$

begin

instance

by *intro-classes (simp add: less-V-def)*
end

lemma *succ-ne-self* [*simp*]: $i \neq succ\ i$
 by (*metis elts-succ insertI1 mem-not-refl*)

lemma *succ-notin-self*: $succ\ i \notin elts\ i$
 using *elts-succ mem-not-refl* **by** *blast*

lemma *le-succE*: $succ\ i \leq succ\ j \implies i \leq j$
 using *less-eq-V-def mem-not-sym* **by** *auto*

lemma *succ-inject-iff* [*iff*]: $succ\ i = succ\ j \iff i = j$
 by (*simp add: dual-order.antisym le-succE*)

lemma *inj-succ*: *inj succ*
 by (*simp add: inj-def*)

lemma *succ-neq-zero*: $succ\ x \neq 0$
 by (*metis elts-0 elts-succ insert-not-empty*)

definition *pred* **where** $pred\ i \equiv THE\ j.\ i = succ\ j$

lemma *pred-succ* [*simp*]: $pred\ (succ\ i) = i$
 by (*simp add: pred-def*)

1.4 Ordinals

definition *Transset* **where** $Transset\ x \equiv \forall y \in elts\ x.\ y \leq x$

definition *Ord* **where** $Ord\ x \equiv Transset\ x \wedge (\forall y \in elts\ x.\ Transset\ y)$

abbreviation ON where $ON \equiv Collect\ Ord$

1.4.1 Transitive sets

lemma *Transset-0* [*iff*]: $Transset\ 0$
by (*auto simp: Transset-def*)

lemma *Transset-succ* [*intro*]:
assumes $Transset\ x$ **shows** $Transset\ (succ\ x)$
using *assms*
by (*auto simp: Transset-def succ-def less-eq-V-def*)

lemma *Transset-Sup*:
assumes $\bigwedge x. x \in X \implies Transset\ x$ **shows** $Transset\ (\bigsqcup X)$
proof (*cases small X*)
case *True*
with *assms* **show** *?thesis*
by (*simp add: Transset-def*) (*meson Sup-upper assms dual-order.trans*)
qed (*simp add: Sup-V-def*)

lemma *Transset-sup*:
assumes $Transset\ x\ Transset\ y$ **shows** $Transset\ (x \sqcup y)$
using *Transset-def assms* **by** *fastforce*

lemma *Transset-inf*: $\llbracket Transset\ i; Transset\ j \rrbracket \implies Transset\ (i \sqcap j)$
by (*simp add: Transset-def rev-usubsetD*)

lemma *Transset-VPow*: $Transset(i) \implies Transset(VPow(i))$
by (*auto simp: Transset-def*)

lemma *Transset-Inf*: $(\bigwedge i. i \in A \implies Transset\ i) \implies Transset\ (\prod A)$
by (*force simp: Transset-def Inf-V-def*)

lemma *Transset-SUP*: $(\bigwedge x. x \in A \implies Transset\ (B\ x)) \implies Transset\ (\bigsqcup (B\ ` A))$
by (*metis Transset-Sup imageE*)

lemma *Transset-INT*: $(\bigwedge x. x \in A \implies Transset\ (B\ x)) \implies Transset\ (\prod (B\ ` A))$
by (*metis Transset-Inf imageE*)

1.4.2 Zero, successor, sups

lemma *Ord-0* [*iff*]: $Ord\ 0$
by (*auto simp: Ord-def*)

lemma *Ord-succ* [*intro*]:
assumes $Ord\ x$ **shows** $Ord\ (succ\ x)$
using *assms* **by** (*auto simp: Ord-def*)

lemma *Ord-Sup*:
assumes $\bigwedge x. x \in X \implies Ord\ x$ **shows** $Ord\ (\bigsqcup X)$

proof (*cases small X*)
case *True*
with *assms show ?thesis*
by (*auto simp: Ord-def Transset-Sup*)
qed (*simp add: Sup-V-def*)

lemma *Ord-Union*:
assumes $\bigwedge x. x \in X \implies \text{Ord } x \text{ small } X$ **shows** $\text{Ord } (\text{set } (\bigcup (\text{elts } 'X)))$
by (*metis Ord-Sup Sup-V-def assms*)

lemma *Ord-sup*:
assumes $\text{Ord } x \text{ Ord } y$ **shows** $\text{Ord } (x \sqcup y)$
using *assms*
proof (*clarsimp simp: Ord-def*)
show $\text{Transset } (x \sqcup y) \wedge (\forall y \in \text{elts } x \cup \text{elts } y. \text{Transset } y)$
if $\text{Transset } x \text{ Transset } y \forall y \in \text{elts } x. \text{Transset } y \forall y \in \text{elts } y. \text{Transset } y$
using *Ord-def Transset-sup assms* **by** *auto*
qed

lemma *big-ON [simp]: \neg small ON*
proof
assume *small ON*
then have $\text{set } ON \in ON$
by (*metis Ord-Union Ord-succ Sup-upper elts-Sup elts-succ insertI1 mem-Collect-eq mem-not-refl set-of-elts vsubsetD*)
then show *False*
by (*metis \langle small ON \rangle elts-of-set mem-not-refl*)
qed

lemma *Ord-1 [iff]: Ord 1*
using *Ord-succ one-V-def succ-def vinsert-def* **by** *fastforce*

lemma *OrdmemD: Ord k \implies j \in elts k \implies j < k*
using *Ord-def Transset-def less-V-def* **by** *auto*

lemma *Ord-trans: $\llbracket i \in \text{elts } j; j \in \text{elts } k; \text{Ord } k \rrbracket \implies i \in \text{elts } k$*
using *Ord-def Transset-def* **by** *blast*

lemma *mem-0-Ord*:
assumes $k: \text{Ord } k$ **and** $\text{knz}: k \neq 0$ **shows** $0 \in \text{elts } k$
by (*metis Ord-def Transset-def inf.orderE k knz trad-foundation*)

lemma *Ord-in-Ord: $\llbracket \text{Ord } k; m \in \text{elts } k \rrbracket \implies \text{Ord } m$*
using *Ord-def Ord-trans* **by** *blast*

lemma *OrdI: $\llbracket \text{Transset } i; \bigwedge x. x \in \text{elts } i \implies \text{Transset } x \rrbracket \implies \text{Ord } i$*
by (*simp add: Ord-def*)

lemma *Ord-is-Transset: Ord i \implies Transset i*

by (*simp add: Ord-def*)

lemma *Ord-contains-Transset*: $\llbracket \text{Ord } i; j \in \text{elts } i \rrbracket \implies \text{Transset } j$
using *Ord-def* **by** *blast*

lemma *ON-imp-Ord*:
assumes $H \subseteq ON$ $x \in H$
shows *Ord* x
using *assms* **by** *blast*

lemma *elts-subset-ON*: $\text{Ord } \alpha \implies \text{elts } \alpha \subseteq ON$
using *Ord-in-Ord* **by** *blast*

lemma *Transset-pred* [*simp*]: $\text{Transset } x \implies \bigsqcup (\text{elts } (\text{succ } x)) = x$
by (*fastforce simp: Transset-def*)

lemma *Ord-pred* [*simp*]: $\text{Ord } \beta \implies \bigsqcup (\text{insert } \beta (\text{elts } \beta)) = \beta$
using *Ord-def Transset-pred* **by** *auto*

1.4.3 Induction, Linearity, etc.

lemma *Ord-induct* [*consumes 1, case-names step*]:
assumes $k: \text{Ord } k$
and *step*: $\bigwedge x. \llbracket \text{Ord } x; \bigwedge y. y \in \text{elts } x \implies P y \rrbracket \implies P x$
shows $P k$
using *foundation k*
proof (*induction k rule: wf-induct-rule*)
case (*less x*)
then show *?case*
using *Ord-in-Ord local.step* **by** *auto*
qed

Comparability of ordinals

lemma *Ord-linear*: $\text{Ord } k \implies \text{Ord } l \implies k \in \text{elts } l \vee k=l \vee l \in \text{elts } k$
proof (*induct k arbitrary: l rule: Ord-induct*)
case (*step k*)
note *step-k = step*
show *?case* **using** $\langle \text{Ord } l \rangle$
proof (*induct l rule: Ord-induct*)
case (*step l*)
thus *?case* **using** *step-k*
by (*metis Ord-trans V-equalityI*)
qed
qed

The trichotomy law for ordinals

lemma *Ord-linear-lt*:
assumes $\text{Ord } k$ $\text{Ord } l$
obtains (*lt*) $k < l$ | (*eq*) $k=l$ | (*gt*) $l < k$

using *Ord-linear OrdmemD assms* **by** *blast*

lemma *Ord-linear2*:
assumes *Ord k Ord l*
obtains $(lt) k < l \mid (ge) l \leq k$
by *(metis Ord-linear-lt eq-refl assms order.strict-implies-order)*

lemma *Ord-linear-le*:
assumes *Ord k Ord l*
obtains $(le) k \leq l \mid (ge) l \leq k$
by *(meson Ord-linear2 le-less assms)*

lemma *union-less-iff* [*simp*]: $\llbracket Ord\ i; Ord\ j \rrbracket \implies i \sqcup j < k \longleftrightarrow i < k \wedge j < k$
by *(metis Ord-linear-le le-iff-sup sup.order-iff sup.strict-boundedE)*

lemma *Ord-mem-iff-lt*: $Ord\ k \implies Ord\ l \implies k \in elts\ l \longleftrightarrow k < l$
by *(metis Ord-linear OrdmemD less-le-not-le)*

lemma *Ord-Collect-lt*: $Ord\ \alpha \implies \{\xi. Ord\ \xi \wedge \xi < \alpha\} = elts\ \alpha$
by *(auto simp flip: Ord-mem-iff-lt elim: Ord-in-Ord OrdmemD)*

lemma *Ord-not-less*: $\llbracket Ord\ x; Ord\ y \rrbracket \implies \neg x < y \longleftrightarrow y \leq x$
by *(metis (no-types) Ord-linear2 leD)*

lemma *Ord-not-le*: $\llbracket Ord\ x; Ord\ y \rrbracket \implies \neg x \leq y \longleftrightarrow y < x$
by *(metis (no-types) Ord-linear2 leD)*

lemma *le-succ-iff*: $Ord\ i \implies Ord\ j \implies succ\ i \leq succ\ j \longleftrightarrow i \leq j$
by *(metis Ord-linear-le Ord-succ le-succE order-antisym)*

lemma *succ-le-iff*: $Ord\ i \implies Ord\ j \implies succ\ i \leq j \longleftrightarrow i < j$
using *Ord-mem-iff-lt dual-order.strict-implies-order less-eq-V-def* **by** *fastforce*

lemma *succ-in-Sup-Ord*:
assumes *eq: succ $\beta = \sqcup A$ and small A $A \subseteq ON$ Ord β*
shows *succ $\beta \in A$*
proof –
have $\neg \sqcup A \leq \beta$
using *eq $\langle Ord\ \beta \rangle succ-le-iff$ by fastforce*
then show *?thesis*
using *assms by (metis Ord-linear2 Sup-least Sup-upper eq-iff mem-Collect-eq subsetD succ-le-iff)*
qed

lemma *in-succ-iff*: $Ord\ i \implies j \in elts\ (ZFC-in-HOL.succ\ i) \longleftrightarrow Ord\ j \wedge j \leq i$
by *(metis Ord-in-Ord Ord-mem-iff-lt Ord-not-le Ord-succ succ-le-iff)*

lemma *zero-in-succ* [*simp,intro*]: $Ord\ i \implies 0 \in elts\ (succ\ i)$
using *mem-0-Ord* **by** *auto*

lemma *less-succ-self*: $x < \text{succ } x$
by (*simp add: less-eq-V-def order-neq-le-trans subset-insertI*)

lemma *Ord-finite-Sup*: $\llbracket \text{finite } A; A \subseteq \text{ON}; A \neq \{\} \rrbracket \implies \bigsqcup A \in A$
proof (*induction A rule: finite-induct*)
case (*insert x A*)
then have *: *small A A* \subseteq *ON Ord x*
by (*auto simp add: ZFC-in-HOL.finite-imp-small insert.hyps*)
show ?*case*
proof (*cases A = \{\}*)
case *False*
then have $\bigsqcup A \in A$
using *insert by blast*
then have $\bigsqcup A \leq x$ **if** $x \sqcup \bigsqcup A \notin A$
using * **by** (*metis ON-imp-Ord Ord-linear-le sup.absorb2 that*)
then show ?*thesis*
by (*fastforce simp: \langle small A \rangle Sup-V-insert*)
qed auto
qed auto

1.4.4 The natural numbers

primrec *ord-of-nat* :: $\text{nat} \Rightarrow V$ **where**
ord-of-nat 0 = 0
| *ord-of-nat (Suc n) = succ (ord-of-nat n)*

lemma *ord-of-nat-eq-initial*: $\text{ord-of-nat } n = \text{set } (\text{ord-of-nat } \{..
by (*induction n (auto simp: lessThan-Suc succ-def)*)$

lemma *mem-ord-of-nat-iff* [*simp*]: $x \in \text{elts } (\text{ord-of-nat } n) \iff (\exists m < n. x = \text{ord-of-nat } m)$
by (*subst ord-of-nat-eq-initial auto*)

lemma *elts-ord-of-nat*: $\text{elts } (\text{ord-of-nat } k) = \text{ord-of-nat } \{..
by auto$

lemma *Ord-equality*: $\text{Ord } i \implies i = \bigsqcup (\text{succ } \{ \text{elts } i \})$
by (*force intro: Ord-trans*)

lemma *Ord-ord-of-nat* [*simp*]: $\text{Ord } (\text{ord-of-nat } k)$
by (*induct k, auto*)

lemma *ord-of-nat-equality*: $\text{ord-of-nat } n = \bigsqcup ((\text{succ } \circ \text{ord-of-nat}) \{..
by (*metis Ord-equality Ord-ord-of-nat elts-of-set image-comp small-image-nat-V ord-of-nat-eq-initial*)$

definition $\omega :: V$ **where** $\omega \equiv \text{set } (\text{range } \text{ord-of-nat})$

lemma *elts- ω* : $\text{elts } \omega = \{\alpha. \exists n. \alpha = \text{ord-of-nat } n\}$
by (*auto simp: ω -def image-iff*)

lemma *nat-into-Ord* [*simp*]: $n \in \text{elts } \omega \implies \text{Ord } n$
by (*metis Ord-ord-of-nat ω -def elts-of-set image-iff inf*)

lemma *Sup- ω* : $\bigsqcup (\text{elts } \omega) = \omega$
unfolding *ω -def* **by** *force*

lemma *Ord- ω* [*iff*]: $\text{Ord } \omega$
by (*metis Ord-Sup Sup- ω nat-into-Ord*)

lemma *zero-in-omega* [*iff*]: $0 \in \text{elts } \omega$
by (*metis ω -def elts-of-set inf ord-of-nat.simps(1) rangeI*)

lemma *succ-in-omega* [*simp*]: $n \in \text{elts } \omega \implies \text{succ } n \in \text{elts } \omega$
by (*metis ω -def elts-of-set image-iff small-image-nat-V ord-of-nat.simps(2) rangeI*)

lemma *ord-of-eq-0*: $\text{ord-of-nat } j = 0 \implies j = 0$
by (*induct j*) (*auto simp: succ-neq-zero*)

lemma *ord-of-nat-le-omega*: $\text{ord-of-nat } n \leq \omega$
by (*metis Sup- ω ZFC-in-HOL.Sup-upper ω -def elts-of-set inf rangeI*)

lemma *ord-of-eq-0-iff* [*simp*]: $\text{ord-of-nat } n = 0 \longleftrightarrow n=0$
by (*auto simp: ord-of-eq-0*)

lemma *ord-of-nat-inject* [*iff*]: $\text{ord-of-nat } i = \text{ord-of-nat } j \longleftrightarrow i=j$
proof (*induct i arbitrary: j*)
case 0 **show** ?*case*
using *ord-of-eq-0* **by** *auto*
next
case (*Suc i*) **then show** ?*case*
by *auto* (*metis elts-0 elts-succ insert-not-empty not0-implies-Suc ord-of-nat.simps succ-inject-iff*)
qed

corollary *inj-ord-of-nat*: *inj ord-of-nat*
by (*simp add: linorder-injI*)

corollary *countable*:
assumes *countable X* **shows** *small X*
proof –
have $X \subseteq \text{range } (\text{from-nat-into } X)$
by (*simp add: assms subset-range-from-nat-into*)
then show ?*thesis*
by (*meson inf-raw inj-ord-of-nat replacement small-def smaller-than-small*)
qed

corollary *infinite- ω* : *infinite* (elts ω)
using *range-inj-infinite* [of ord-of-nat]
by (*simp add: ω -def inj-ord-of-nat*)

corollary *ord-of-nat-mono-iff* [iff]: *ord-of-nat* $i \leq \text{ord-of-nat } j \iff i \leq j$
by (*metis Ord-def Ord-ord-of-nat Transset-def eq-iff mem-ord-of-nat-iff not-less ord-of-nat-inject*)

corollary *ord-of-nat-strict-mono-iff* [iff]: *ord-of-nat* $i < \text{ord-of-nat } j \iff i < j$
by (*simp add: less-le-not-le*)

lemma *small-image-nat* [simp]:
fixes $N :: \text{nat set}$ **shows** *small* ($g \text{ ' } N$)
by (*simp add: countable*)

lemma *finite-Ord-omega*: $\alpha \in \text{elts } \omega \implies \text{finite} (\text{elts } \alpha)$
proof (*clarsimp simp add: ω -def*)
show *finite* (elts (ord-of-nat n)) **if** $\alpha = \text{ord-of-nat } n$ **for** n
using *that* **by** (*simp add: ord-of-nat-eq-initial [of n]*)
qed

lemma *infinite-Ord-omega*: *Ord* $\alpha \implies \text{infinite} (\text{elts } \alpha) \implies \omega \leq \alpha$
by (*meson Ord- ω Ord-linear2 Ord-mem-iff-lt finite-Ord-omega*)

lemma *ord-of-minus-1*: $n > 0 \implies \text{ord-of-nat } n = \text{succ} (\text{ord-of-nat } (n - 1))$
by (*metis Suc-diff-1 ord-of-nat.simps(2)*)

lemma *card-ord-of-nat* [simp]: *card* (elts (ord-of-nat m)) = m
by (*induction m*) (*auto simp: ω -def finite-Ord-omega*)

lemma *ord-of-nat- ω* [iff]: *ord-of-nat* $n \in \text{elts } \omega$
by (*simp add: ω -def*)

lemma *succ- ω -iff* [iff]: *succ* $n \in \text{elts } \omega \iff n \in \text{elts } \omega$
by (*metis Ord- ω OrdmemD elts-vinsert insert-iff less-V-def succ-def succ-in-omega vsubsetD*)

lemma *ω -gt0* [simp]: $\omega > 0$
by (*simp add: OrdmemD*)

lemma *ω -gt1* [simp]: $\omega > 1$
by (*simp add: OrdmemD one-V-def*)

1.4.5 Limit ordinals

definition *Limit* :: $V \Rightarrow \text{bool}$
where *Limit* $i \equiv \text{Ord } i \wedge 0 \in \text{elts } i \wedge (\forall y. y \in \text{elts } i \longrightarrow \text{succ } y \in \text{elts } i)$

lemma *zero-not-Limit* [iff]: $\neg \text{Limit } 0$

by (simp add: Limit-def)

lemma not-succ-Limit [simp]: $\neg \text{Limit}(\text{succ } i)$
by (metis Limit-def Ord-mem-iff-lt elts-succ insertI1 less-irrefl)

lemma Limit-is-Ord: $\text{Limit } \xi \implies \text{Ord } \xi$
by (simp add: Limit-def)

lemma succ-in-Limit-iff: $\text{Limit } \xi \implies \text{succ } \alpha \in \text{elts } \xi \longleftrightarrow \alpha \in \text{elts } \xi$
by (metis Limit-def OrdmemD elts-succ insertI1 less-V-def vsubsetD)

lemma Limit-eq-Sup-self [simp]: $\text{Limit } i \implies \text{Sup } (\text{elts } i) = i$
apply (rule order-antisym)
apply (simp add: Limit-def Ord-def Transset-def Sup-least)
by (metis Limit-def Ord-equality Sup-V-def SUP-le-iff Sup-upper small-elts)

lemma zero-less-Limit: $\text{Limit } \beta \implies 0 < \beta$
by (simp add: Limit-def OrdmemD)

lemma non-Limit-ord-of-nat [iff]: $\neg \text{Limit } (\text{ord-of-nat } m)$
by (metis Limit-def mem-ord-of-nat-iff not-succ-Limit ord-of-eq-0-iff ord-of-minus-1)

lemma Limit-omega [iff]: $\text{Limit } \omega$
by (simp add: Limit-def)

lemma omega-nonzero [simp]: $\omega \neq 0$
using Limit-omega **by** fastforce

lemma Ord-cases-lemma:
assumes Ord k **shows** $k = 0 \vee (\exists j. k = \text{succ } j) \vee \text{Limit } k$
proof (cases Limit k)
case False
have $\text{succ } j \in \text{elts } k$ **if** $\forall j. k \neq \text{succ } j$ $j \in \text{elts } k$ **for** j
by (metis Ord-in-Ord Ord-linear Ord-succ assms elts-succ insertE mem-not-sym that)
with assms **show** ?thesis
by (auto simp: Limit-def mem-0-Ord)
qed auto

lemma Ord-cases [cases type: V, case-names 0 succ limit]:
assumes Ord k
obtains $k = 0 \mid l$ **where** Ord l $\text{succ } l = k \mid \text{Limit } k$
by (metis assms Ord-cases-lemma Ord-in-Ord elts-succ insertI1)

lemma non-succ-LimitI:
assumes $i \neq 0$ Ord i $\bigwedge y. \text{succ}(y) \neq i$
shows Limit i
using Ord-cases-lemma assms **by** blast

lemma *Ord-induct3* [*consumes 1, case-names 0 succ Limit, induct type: V*]:
assumes α : *Ord* α
and P : $P\ 0 \wedge \alpha. \llbracket \text{Ord } \alpha; P\ \alpha \rrbracket \implies P\ (\text{succ } \alpha)$
 $\bigwedge \alpha. \llbracket \text{Limit } \alpha; \bigwedge \xi. \xi \in \text{elts } \alpha \implies P\ \xi \rrbracket \implies P\ (\bigsqcup \xi \in \text{elts } \alpha. \xi)$
shows $P\ \alpha$
using α
proof (*induction* α *rule: Ord-induct*)
case (*step* α)
then show *?case*
by (*metis Limit-eq-Sup-self Ord-cases P elts-succ image-ident insertI1*)
qed

1.4.6 Properties of LEAST for ordinals

lemma
assumes *Ord* k P k
shows *Ord-LeastI*: $P\ (\text{LEAST } i. \text{Ord } i \wedge P\ i)$ **and** *Ord-Least-le*: $(\text{LEAST } i. \text{Ord } i \wedge P\ i) \leq k$
proof –
have $P\ (\text{LEAST } i. \text{Ord } i \wedge P\ i) \wedge (\text{LEAST } i. \text{Ord } i \wedge P\ i) \leq k$
using *assms*
proof (*induct* k *rule: Ord-induct*)
case (*step* x) **then have** $P\ x$ **by** *simp*
show *?case* **proof** (*rule classical*)
assume *assm*: $\neg (P\ (\text{LEAST } a. \text{Ord } a \wedge P\ a) \wedge (\text{LEAST } a. \text{Ord } a \wedge P\ a) \leq x)$
have $\bigwedge y. \text{Ord } y \wedge P\ y \implies x \leq y$
proof (*rule classical*)
fix y
assume y : $\text{Ord } y \wedge P\ y \neg x \leq y$
with *step* **obtain** $P\ (\text{LEAST } a. \text{Ord } a \wedge P\ a)$ **and** *le*: $(\text{LEAST } a. \text{Ord } a \wedge P\ a) \leq y$
by (*meson Ord-linear2 Ord-mem-iff-lt*)
with *assm* **have** $x < (\text{LEAST } a. \text{Ord } a \wedge P\ a)$
by (*meson Ord-linear-le y order.trans <Ord x>*)
then show $x \leq y$
using *le* **by** *auto*
qed
then have *Least*: $(\text{LEAST } a. \text{Ord } a \wedge P\ a) = x$
by (*simp add: Least-equality <Ord x> step.prem*)
with $\langle P\ x \rangle$ **show** *?thesis* **by** *simp*
qed
qed
then show $P\ (\text{LEAST } i. \text{Ord } i \wedge P\ i)$ **and** $(\text{LEAST } i. \text{Ord } i \wedge P\ i) \leq k$ **by** *auto*
qed

lemma *Ord-Least*:
assumes *Ord* k P k
shows *Ord* $(\text{LEAST } i. \text{Ord } i \wedge P\ i)$

proof –
have $\text{Ord } (\text{LEAST } i. \text{Ord } i \wedge (\text{Ord } i \wedge P i))$
using Ord-LeastI [**where** $P = \lambda i. \text{Ord } i \wedge P i$] **assms by blast**
then show *?thesis*
by simp
qed

— The following 3 lemmas are due to Brian Huffman

lemma $\text{Ord-LeastI-ex: } \exists i. \text{Ord } i \wedge P i \implies P (\text{LEAST } i. \text{Ord } i \wedge P i)$
using Ord-LeastI **by blast**

lemma Ord-LeastI2:
 $\llbracket \text{Ord } a; P a; \bigwedge x. \llbracket \text{Ord } x; P x \rrbracket \implies Q x \rrbracket \implies Q (\text{LEAST } i. \text{Ord } i \wedge P i)$
by ($\text{blast intro: Ord-LeastI Ord-Least}$)

lemma Ord-LeastI2-ex:
 $\exists a. \text{Ord } a \wedge P a \implies (\bigwedge x. \llbracket \text{Ord } x; P x \rrbracket \implies Q x) \implies Q (\text{LEAST } i. \text{Ord } i \wedge P i)$
by ($\text{blast intro: Ord-LeastI-ex Ord-Least}$)

lemma $\text{Ord-LeastI2-wellorder:}$
assumes $\text{Ord } a P a$
and $\bigwedge a. \llbracket P a; \forall b. \text{Ord } b \wedge P b \longrightarrow a \leq b \rrbracket \implies Q a$
shows $Q (\text{LEAST } i. \text{Ord } i \wedge P i)$
proof ($\text{rule LeastI2-order}$)
show $\text{Ord } (\text{LEAST } i. \text{Ord } i \wedge P i) \wedge P (\text{LEAST } i. \text{Ord } i \wedge P i)$
using $\text{Ord-Least Ord-LeastI assms by auto}$
next
fix y **assume** $\text{Ord } y \wedge P y$ **thus** $(\text{LEAST } i. \text{Ord } i \wedge P i) \leq y$
by ($\text{simp add: Ord-Least-le}$)
next
fix x **assume** $\text{Ord } x \wedge P x \forall y. \text{Ord } y \wedge P y \longrightarrow x \leq y$ **thus** $Q x$
by ($\text{simp add: assms(3)}$)
qed

lemma $\text{Ord-LeastI2-wellorder-ex:}$
assumes $\exists x. \text{Ord } x \wedge P x$
and $\bigwedge a. \llbracket P a; \forall b. \text{Ord } b \wedge P b \longrightarrow a \leq b \rrbracket \implies Q a$
shows $Q (\text{LEAST } i. \text{Ord } i \wedge P i)$
using **assms by clarify** ($\text{blast intro!: Ord-LeastI2-wellorder}$)

lemma $\text{not-less-Ord-Least: } \llbracket k < (\text{LEAST } x. \text{Ord } x \wedge P x); \text{Ord } k \rrbracket \implies \neg P k$
using $\text{Ord-Least-le less-le-not-le}$ **by auto**

lemma $\text{exists-Ord-Least-iff: } (\exists \alpha. \text{Ord } \alpha \wedge P \alpha) \longleftrightarrow (\exists \alpha. \text{Ord } \alpha \wedge P \alpha \wedge (\forall \beta < \alpha. \text{Ord } \beta \longrightarrow \neg P \beta))$ (**is** $?lhs \longleftrightarrow ?rhs$)
proof
assume $?rhs$ **thus** $?lhs$ **by blast**
next

assume $H: ?lhs$ **then obtain** α **where** $\alpha: Ord \ \alpha \ P \ \alpha$ **by** *blast*
let $?x = LEAST \ \alpha. \ Ord \ \alpha \wedge P \ \alpha$
have $Ord \ ?x$
by (*metis Ord-Least* α)
moreover
{ **fix** β **assume** $m: \beta < ?x \ Ord \ \beta$
from *not-less-Ord-Least*[*OF* m] **have** $\neg P \ \beta .$ }
ultimately show $?rhs$
using *Ord-LeastI-ex*[*OF* H] **by** *blast*
qed

lemma *Ord-mono-imp-increasing*:
assumes *fun-hD*: $h \in D \rightarrow D$
and *mono-h*: *strict-mono-on* $D \ h$
and $D \subseteq ON$ **and** $\nu: \nu \in D$
shows $\nu \leq h \ \nu$
proof (*rule ccontr*)
assume *non*: $\neg \nu \leq h \ \nu$
define μ **where** $\mu \equiv LEAST \ \mu. \ Ord \ \mu \wedge \neg \mu \leq h \ \mu \wedge \mu \in D$
have $Ord \ \nu$
using $\nu \langle D \subseteq ON \rangle$ **by** *blast*
then have $\mu: \neg \mu \leq h \ \mu \wedge \mu \in D$
unfolding μ -*def* **by** (*rule Ord-LeastI*) (*simp add: \nu non*)
have $Ord \ (h \ \nu)$
using *assms* **by** *auto*
then have $Ord \ (h \ (h \ \nu))$
by (*meson ON-imp-Ord \nu assms funcset-mem*)
have $Ord \ \mu$
using $\mu \langle D \subseteq ON \rangle$ **by** *blast*
then have $h \ \mu < \mu$
by (*metis ON-imp-Ord Ord-linear2 PiE \mu \langle D \subseteq ON \rangle fun-hD*)
then have $\neg h \ \mu \leq h \ (h \ \mu)$
using μ *fun-hD* *mono-h* **by** (*force simp: strict-mono-on-def*)
moreover have $*$: $h \ \mu \in D$
using μ *fun-hD* **by** *auto*
moreover have $Ord \ (h \ \mu)$
using $\langle D \subseteq ON \rangle *$ **by** *blast*
ultimately have $\mu \leq h \ \mu$
by (*simp add: \mu-def Ord-Least-le*)
then show *False*
using μ **by** *blast*
qed

lemma *le-Sup-iff*:
assumes $A \subseteq ON$ $Ord \ x$ *small* A **shows** $x \leq \bigsqcup A \iff (\forall y \in ON. y < x \implies (\exists a \in A. y < a))$
proof (*intro iffI ballI impI*)
show $\exists a \in A. y < a$
if $x \leq \bigsqcup A$ $y \in ON$ $y < x$

```

for  $y$ 
proof –
  have  $\neg \sqcup A \leq y \text{ Ord } y$ 
    using that by auto
  then show ?thesis
    by (metis Ord-linear2 Sup-least ‹A ⊆ ON› mem-Collect-eq subset-eq)
qed
show  $x \leq \sqcup A$ 
  if  $\forall y \in ON. y < x \longrightarrow (\exists a \in A. y < a)$ 
    using that assms
    by (metis Ord-Sup Ord-linear-le Sup-upper less-le-not-le mem-Collect-eq subsetD)
qed

```

lemma *le-SUP-iff*: $\llbracket f \text{ ‘ } A \subseteq ON; \text{ Ord } x; \text{ small } A \rrbracket \implies x \leq \sqcup (f \text{ ‘ } A) \longleftrightarrow (\forall y \in ON. y < x \longrightarrow (\exists i \in A. y < f i))$
by (*simp add: le-Sup-iff*)

1.5 Transfinite Recursion and the V-levels

definition *transrec* :: $((V \Rightarrow 'a) \Rightarrow V \Rightarrow 'a) \Rightarrow V \Rightarrow 'a$
where *transrec* $H a \equiv wfrec \{(x,y). x \in elts y\} H a$

lemma *transrec*: $transrec H a = H (\lambda x \in elts a. transrec H x) a$

```

proof –
  have (cut (wfrec {(x, y). x ∈ elts y} H) {(x, y). x ∈ elts y} a)
    =  $(\lambda x \in elts a. wfrec \{(x, y). x \in elts y\} H x)$ 
    by (force simp: cut-def)
  then show ?thesis
    unfolding transrec-def
    by (simp add: foundation wfrec)
qed

```

Avoids explosions in proofs; resolve it with a meta-level definition

lemma *def-transrec*:
 $\llbracket \bigwedge x. f x \equiv transrec H x \rrbracket \implies f a = H(\lambda x \in elts a. f x) a$
by (*metis restrict-ext transrec*)

lemma *eps-induct* [*case-names step*]:
assumes $\bigwedge x. (\bigwedge y. y \in elts x \implies P y) \implies P x$
shows $P a$
using *wf-induct [OF foundation] assms by auto*

definition *Vfrom* :: $[V, V] \Rightarrow V$
where *Vfrom* $a \equiv transrec (\lambda f x. a \sqcup \sqcup ((\lambda y. VPow(f y)) \text{ ‘ } elts x))$

abbreviation *Vset* :: $V \Rightarrow V$ **where** *Vset* $\equiv Vfrom 0$

lemma *Vfrom*: $Vfrom\ a\ i = a \sqcup \bigsqcup((\lambda j. VPow(Vfrom\ a\ j)) \text{ `elts } i)$
apply (*subst Vfrom-def*)
apply (*subst transrec*)
using *Vfrom-def* **by** *auto*

lemma *Vfrom-0* [*simp*]: $Vfrom\ a\ 0 = a$
by (*subst Vfrom*) *auto*

lemma *Vset*: $Vset\ i = \bigsqcup((\lambda j. VPow(Vset\ j)) \text{ `elts } i)$
by (*subst Vfrom*) *auto*

lemma *Vfrom-mono1*:
assumes $a \leq b$ **shows** $Vfrom\ a\ i \leq Vfrom\ b\ i$
proof (*induction i rule: eps-induct*)
case (*step i*)
then have $a \sqcup (\bigsqcup_{j \in elts\ i}. VPow(Vfrom\ a\ j)) \leq b \sqcup (\bigsqcup_{j \in elts\ i}. VPow(Vfrom\ b\ j))$
by (*intro sup-mono cSUP-subset-mono <a ≤ b>*) *auto*
then show *?case*
by (*metis Vfrom*)
qed

lemma *Vfrom-mono2*: $Vfrom\ a\ i \leq Vfrom\ a\ (i \sqcup j)$
proof (*induction arbitrary: j rule: eps-induct*)
case (*step i*)
then have $a \sqcup (\bigsqcup_{j \in elts\ i}. VPow(Vfrom\ a\ j)) \leq a \sqcup (\bigsqcup_{j \in elts\ (i \sqcup j)}. VPow(Vfrom\ a\ j))$
by (*intro sup-mono cSUP-subset-mono order-refl*) *auto*
then show *?case*
by (*metis Vfrom*)
qed

lemma *Vfrom-mono*: $\llbracket Ord\ i; a \leq b; i \leq j \rrbracket \implies Vfrom\ a\ i \leq Vfrom\ b\ j$
by (*metis (no-types) Vfrom-mono1 Vfrom-mono2 dual-order.trans sup.absorb-iff2*)

lemma *Transset-Vfrom*: $Transset(A) \implies Transset(Vfrom\ A\ i)$
proof (*induction i rule: eps-induct*)
case (*step i*)
then show *?case*
by (*metis Transset-SUP Transset-VPow Transset-sup Vfrom*)
qed

lemma *Transset-Vset* [*simp*]: $Transset(Vset\ i)$
by (*simp add: Transset-Vfrom*)

lemma *Vfrom-sup*: $Vfrom\ a\ (i \sqcup j) = Vfrom\ a\ i \sqcup Vfrom\ a\ j$
proof (*rule order-antisym*)
show $Vfrom\ a\ (i \sqcup j) \leq Vfrom\ a\ i \sqcup Vfrom\ a\ j$
by (*simp add: Vfrom [of a i j] Vfrom [of a i] Vfrom [of a j] Sup-Un-distrib*)

image-Un sup.assoc sup.left-commute
show $V\text{from } a \sqcup V\text{from } a \ j \leq V\text{from } a \ (i \sqcup j)$
by (*metis Vfrom-mono2 le-supI sup-commute*)
qed

lemma *Vfrom-succ-Ord*:
assumes *Ord i* **shows** $V\text{from } a \ (\text{succ } i) = a \sqcup VPow(V\text{from } a \ i)$
proof (*cases i = 0*)
case *True*
then show *?thesis*
by (*simp add: Vfrom [of - succ 0]*)
next
case *False*
have *: $(\bigsqcup x \in \text{elts } i. VPow (V\text{from } a \ x)) \leq VPow (V\text{from } a \ i)$
proof (*rule cSup-least*)
show $(\lambda x. VPow (V\text{from } a \ x)) \text{ ' elts } i \neq \{\}$
using *False by auto*
show $x \leq VPow (V\text{from } a \ i)$ **if** $x \in (\lambda x. VPow (V\text{from } a \ x)) \text{ ' elts } i$ **for** x
using *that*
by (*clarsimp (meson Ord-in-Ord Ord-linear-le Vfrom-mono assms mem-not-refl order-refl vsubsetD)*)
qed
show *?thesis*
proof (*rule Vfrom [THEN trans]*)
show $a \sqcup (\bigsqcup j \in \text{elts } (\text{succ } i). VPow (V\text{from } a \ j)) = a \sqcup VPow (V\text{from } a \ i)$
using *assms*
by (*intro sup-mono order-antisym*) (*auto simp: Sup-V-insert **)
qed
qed

lemma *Vset-succ*: $Ord \ i \implies Vset(\text{succ}(i)) = VPow(Vset(i))$
by (*simp add: Vfrom-succ-Ord*)

lemma *Vfrom-Sup*:
assumes $X \neq \{\}$ *small X*
shows $V\text{from } a \ (\text{Sup } X) = (\bigsqcup y \in X. V\text{from } a \ y)$
proof (*rule order-antisym*)
have $V\text{from } a \ (\bigsqcup X) = a \sqcup (\bigsqcup j \in \text{elts } (\bigsqcup X). VPow (V\text{from } a \ j))$
by (*metis Vfrom*)
also have $\dots \leq \bigsqcup (V\text{from } a \ \text{' } X)$
proof –
have $a \leq \bigsqcup (V\text{from } a \ \text{' } X)$
by (*metis Vfrom all-not-in-conv assms bdd-above-iff-small cSUP-upper2 replacement sup-ge1*)
moreover have $(\bigsqcup j \in \text{elts } (\bigsqcup X). VPow (V\text{from } a \ j)) \leq \bigsqcup (V\text{from } a \ \text{' } X)$
proof –
have $VPow (V\text{from } a \ x) \leq \bigsqcup (V\text{from } a \ \text{' } X)$
if $y \in X \ x \in \text{elts } y$ **for** $x \ y$
proof –

```

    have VPow (Vfrom a x) ≤ Vfrom a y
      by (metis Vfrom bdd-above-iff-small cSUP-upper2 le-supI2 order-refl
replacement small-elts that(2))
    also have ... ≤  $\bigsqcup$  (Vfrom a ' X)
      using assms that by (force intro: cSUP-upper)
    finally show ?thesis .
  qed
  then show ?thesis
    by (simp add: SUP-le-iff ‹small X›)
  qed
  ultimately show ?thesis
    by auto
  qed
  finally show Vfrom a ( $\bigsqcup$  X) ≤  $\bigsqcup$  (Vfrom a ' X) .
  have  $\bigwedge x. x \in X \implies$ 
    a  $\sqcup$  ( $\bigsqcup_{j \in \text{elts } x}. \text{VPow } (Vfrom a j)$ )
    ≤ a  $\sqcup$  ( $\bigsqcup_{j \in \text{elts } (\bigsqcup X)}. \text{VPow } (Vfrom a j)$ )
    using cSUP-subset-mono ‹small X› by auto
  then show  $\bigsqcup$  (Vfrom a ' X) ≤ Vfrom a ( $\bigsqcup$  X)
    by (metis Vfrom assms(1) cSUP-least)
  qed

```

```

lemma Limit-Vfrom-eq:
  Limit(i)  $\implies$  Vfrom a i = ( $\bigsqcup y \in \text{elts } i. \text{Vfrom a } y$ )
  by (metis Limit-def Limit-eq-Sup-self Vfrom-Sup ex-in-conv small-elts)
end

```

2 Cartesian products, Disjoint Sums, Ranks, Cardinals

```

theory ZFC-Cardinals
  imports ZFC-in-HOL

```

```

begin

```

```

declare [[coercion-enabled]]
declare [[coercion ord-of-nat :: nat  $\Rightarrow$  V]]

```

2.1 Ordered Pairs

```

lemma singleton-eq-iff [iff]: set {a} = set {b}  $\longleftrightarrow$  a=b
  by simp

```

```

lemma doubleton-eq-iff: set {a,b} = set {c,d}  $\longleftrightarrow$  (a=c  $\wedge$  b=d)  $\vee$  (a=d  $\wedge$  b=c)
  by (simp add: Set.doubleton-eq-iff)

```

```

definition vpair :: V  $\Rightarrow$  V  $\Rightarrow$  V

```


where $vpair\ a\ b = set\ \{set\ \{a\}, set\ \{a,b\}\}$

definition $vfst :: V \Rightarrow V$

where $vfst\ p \equiv THE\ x.\ \exists y.\ p = vpair\ x\ y$

definition $vsnd :: V \Rightarrow V$

where $vsnd\ p \equiv THE\ y.\ \exists x.\ p = vpair\ x\ y$

definition $vsplit :: [[V, V] \Rightarrow 'a, V] \Rightarrow 'a::\{\}$ — for pattern-matching

where $vsplit\ c \equiv \lambda p.\ c\ (vfst\ p)\ (vsnd\ p)$

nonterminal Vs

syntax (ASCII)

-*Tuple* $:: [V, Vs] \Rightarrow V$ ($\langle(-,/ -)\rangle$)

-*hpattern* $:: [pttrn, patterns] \Rightarrow pttrn$ ($\langle(-,/ -)\rangle$)

syntax

$:: V \Rightarrow Vs$ ($\langle-\rangle$)

-*Enum* $:: [V, Vs] \Rightarrow Vs$ ($\langle(-,/ -)\rangle$)

-*Tuple* $:: [V, Vs] \Rightarrow V$ ($\langle\langle(-,/ -)\rangle\rangle$)

-*hpattern* $:: [pttrn, patterns] \Rightarrow pttrn$ ($\langle\langle(-,/ -)\rangle\rangle$)

syntax-consts

-*Enum* -*Tuple* $\equiv vpair\ and$

-*hpattern* $\equiv vsplit$

translations

$\langle x, y, z \rangle \equiv \langle x, \langle y, z \rangle \rangle$

$\langle x, y \rangle \equiv CONST\ vpair\ x\ y$

$\langle x, y, z \rangle \equiv \langle x, \langle y, z \rangle \rangle$

$\lambda \langle x, y, zs \rangle.\ b \equiv CONST\ vsplit(\lambda x\ \langle y, zs \rangle.\ b)$

$\lambda \langle x, y \rangle.\ b \equiv CONST\ vsplit(\lambda x\ y.\ b)$

lemma $vpair-def'$: $vpair\ a\ b = set\ \{set\ \{a,a\}, set\ \{a,b\}\}$

by (*simp add: vpair-def*)

lemma $vpair-iff$ [*simp*]: $vpair\ a\ b = vpair\ a'\ b' \longleftrightarrow a=a' \wedge b=b'$

unfolding $vpair-def'$ *doubleton-eq-iff* **by** *auto*

lemmas $vpair-inject = vpair-iff$ [*THEN iffD1, THEN conjE, elim!*]

lemma $vfst-conv$ [*simp*]: $vfst\ \langle a, b \rangle = a$

by (*simp add: vfst-def*)

lemma $vsnd-conv$ [*simp*]: $vsnd\ \langle a, b \rangle = b$

by (*simp add: vsnd-def*)

lemma $vsplit$ [*simp*]: $vsplit\ c\ \langle a, b \rangle = c\ a\ b$

by (*simp add: vsplit-def*)

lemma $vpair-neq-fst$: $\langle a, b \rangle \neq a$

by (*metis elts-of-set insertI1 mem-not-sym small-upair vpair-def'*)

lemma *vpair-neq-snd*: $\langle a, b \rangle \neq b$

by (*metis elts-of-set insertI1 mem-not-sym small-upair subsetD subset-insertI vpair-def'*)

lemma *vpair-nonzero* [*simp*]: $\langle x, y \rangle \neq 0$

by (*metis elts-0 elts-of-set empty-not-insert small-upair vpair-def*)

lemma *zero-notin-vpair*: $0 \notin \text{elts } \langle x, y \rangle$

by (*auto simp: vpair-def*)

lemma *inj-on-vpair* [*simp*]: *inj-on* $(\lambda(x, y). \langle x, y \rangle)$ *A*

by (*auto simp: inj-on-def*)

2.2 Generalized Cartesian product

definition *VSigma* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V$

where *VSigma* *A B* $\equiv \text{set}(\bigcup x \in \text{elts } A. \bigcup y \in \text{elts } (B x). \{\langle x, y \rangle\})$

abbreviation *vtimes* **where** *vtimes* *A B* $\equiv \text{VSigma } A (\lambda x. B)$

definition *pairs* :: $V \Rightarrow (V * V)\text{set}$

where *pairs* *r* $\equiv \{(x, y). \langle x, y \rangle \in \text{elts } r\}$

lemma *pairs-iff-elts*: $\langle x, y \rangle \in \text{pairs } z \iff \langle x, y \rangle \in \text{elts } z$

by (*simp add: pairs-def*)

lemma *VSigma-iff* [*simp*]: $\langle a, b \rangle \in \text{elts } (\text{VSigma } A B) \iff a \in \text{elts } A \wedge b \in \text{elts } (B a)$

by (*auto simp: VSigma-def UNION-singleton-eq-range*)

lemma *VSigmaI* [*intro!*]: $\llbracket a \in \text{elts } A; b \in \text{elts } (B a) \rrbracket \implies \langle a, b \rangle \in \text{elts } (\text{VSigma } A B)$

by *simp*

lemmas *VSigmaD1* = *VSigma-iff* [*THEN iffD1, THEN conjunct1*]

lemmas *VSigmaD2* = *VSigma-iff* [*THEN iffD1, THEN conjunct2*]

The general elimination rule

lemma *VSigmaE* [*elim!*]:

assumes $c \in \text{elts } (\text{VSigma } A B)$

obtains *x y* **where** $x \in \text{elts } A$ $y \in \text{elts } (B x)$ $c = \langle x, y \rangle$

using *assms* **by** (*auto simp: VSigma-def split: if-split-asm*)

lemma *VSigmaE2* [*elim!*]:

assumes $\langle a, b \rangle \in \text{elts } (\text{VSigma } A B)$ **obtains** $a \in \text{elts } A$ **and** $b \in \text{elts } (B a)$

using *assms* **by** *auto*

lemma *VSigma-empty1* [*simp*]: $VSigma\ 0\ B = 0$
by *auto*

lemma *times-iff* [*simp*]: $\langle a, b \rangle \in elts\ (vtimes\ A\ B) \longleftrightarrow a \in elts\ A \wedge b \in elts\ B$
by *simp*

lemma *timesI* [*intro!*]: $\llbracket a \in elts\ A; b \in elts\ B \rrbracket \implies \langle a, b \rangle \in elts\ (vtimes\ A\ B)$
by *simp*

lemma *times-empty2* [*simp*]: $vtimes\ A\ 0 = 0$
using *elts-0* **by** *blast*

lemma *times-empty-iff*: $VSigma\ A\ B = 0 \longleftrightarrow A=0 \vee (\forall x \in elts\ A. B\ x = 0)$
by (*metis VSigmaE VSigmaI elts-0 empty-iff trad-foundation*)

lemma *elts-VSigma*: $elts\ (VSigma\ A\ B) = (\lambda(x,y). vpair\ x\ y) \text{ ` } Sigma\ (elts\ A)$
 $(\lambda x. elts\ (B\ x))$
by *auto*

lemma *small-Sigma* [*simp*]:
assumes *A*: *small A* **and** *B*: $\bigwedge x. x \in A \implies small\ (B\ x)$
shows *small (Sigma A B)*

proof –

obtain *a* **where** $elts\ a \approx A$
by (*meson assms small-epoll*)
then obtain *f* **where** $f: bij\ betw\ f\ (elts\ a)\ A$
using *epoll-def* **by** *blast*
have $\exists y. elts\ y \approx B\ x$ **if** $x \in A$ **for** *x*
using *B small-epoll that* **by** *blast*
then obtain *g* **where** $g: \bigwedge x. x \in A \implies elts\ (g\ x) \approx B\ x$
by *metis*
with *f* **have** $elts\ (VSigma\ a\ (g \circ f)) \approx Sigma\ A\ B$
by (*simp add: elts-VSigma Sigma-epoll-cong bij-betwE*)
then show *?thesis*
using *small-epoll* **by** *blast*

qed

lemma *small-Times* [*simp*]:
assumes *small A* *small B* **shows** *small (A × B)*
by (*simp add: assms*)

lemma *small-Times-iff*: $small\ (A \times B) \longleftrightarrow small\ A \wedge small\ B \vee A=\{\} \vee B=\{\}$
(is - = ?rhs)

proof

assume ***: *small (A × B)*
{ have *small A* \wedge *small B* **if** $x \in A\ y \in B$ **for** *x y*
proof –
have $A \subseteq fst\ \text{ ` } (A \times B)\ B \subseteq snd\ \text{ ` } (A \times B)$
using *that* **by** *auto*

```

    with that show ?thesis
      by (metis * replacement smaller-than-small)
    qed }
  then show ?rhs
    by (metis equals0I)
next
  assume ?rhs
  then show small (A × B)
    by auto
qed

```

2.3 Disjoint Sum

definition $vsum :: V \Rightarrow V \Rightarrow V$ (**infixl** $\langle \uplus \rangle$ 65) **where**
 $A \uplus B \equiv (VSigma (set \{0\}) (\lambda x. A)) \sqcup (VSigma (set \{1\}) (\lambda x. B))$

definition $Inl :: V \Rightarrow V$ **where**
 $Inl a \equiv \langle 0, a \rangle$

definition $Inr :: V \Rightarrow V$ **where**
 $Inr b \equiv \langle 1, b \rangle$

lemmas $sum-defs = vsum-def Inl-def Inr-def$

lemma $Inl\text{-nonzero}$ [*simp*]: $Inl x \neq 0$
 by (metis $Inl\text{-def}$ $vpair\text{-nonzero}$)

lemma $Inr\text{-nonzero}$ [*simp*]: $Inr x \neq 0$
 by (metis $Inr\text{-def}$ $vpair\text{-nonzero}$)

2.3.1 Equivalences for the injections and an elimination rule

lemma $Inl\text{-in-sum-iff}$ [*iff*]: $Inl a \in elts (A \uplus B) \longleftrightarrow a \in elts A$
 by (auto simp: $sum\text{-defs}$)

lemma $Inr\text{-in-sum-iff}$ [*iff*]: $Inr b \in elts (A \uplus B) \longleftrightarrow b \in elts B$
 by (auto simp: $sum\text{-defs}$)

lemma $sumE$ [*elim!*]:
 assumes $u: u \in elts (A \uplus B)$
 obtains x **where** $x \in elts A$ $u = Inl x$ | y **where** $y \in elts B$ $u = Inr y$ **using** u
 by (auto simp: $sum\text{-defs}$)

2.3.2 Injection and freeness equivalences, for rewriting

lemma $Inl\text{-iff}$ [*iff*]: $Inl a = Inl b \longleftrightarrow a = b$
 by (simp add: $sum\text{-defs}$)

lemma $Inr\text{-iff}$ [*iff*]: $Inr a = Inr b \longleftrightarrow a = b$
 by (simp add: $sum\text{-defs}$)

lemma *inj-on-Inl* [*simp*]: *inj-on Inl A*
by (*simp add: inj-on-def*)

lemma *inj-on-Inr* [*simp*]: *inj-on Inr A*
by (*simp add: inj-on-def*)

lemma *Inl-Inr-iff* [*iff*]: *Inl a = Inr b \longleftrightarrow False*
by (*simp add: sum-defs*)

lemma *Inr-Inl-iff* [*iff*]: *Inr b = Inl a \longleftrightarrow False*
by (*simp add: sum-defs*)

lemma *sum-empty* [*simp*]: *0 \Join 0 = 0*
by *auto*

lemma *elts-vsum*: *elts (a \Join b) = Inl ` (elts a) \cup Inr ` (elts b)*
by *auto*

lemma *sum-iff*: *u \in elts (A \Join B) \longleftrightarrow ($\exists x. x \in$ elts A \wedge u = Inl x) \vee ($\exists y. y \in$ elts B \wedge u = Inr y)*
by *blast*

lemma *sum-subset-iff*: *A \Join B \leq C \Join D \longleftrightarrow A \leq C \wedge B \leq D*
by (*auto simp: less-eq-V-def*)

lemma *sum-equal-iff*:
fixes A :: V **shows** *A \Join B = C \Join D \longleftrightarrow A = C \wedge B = D*
by (*simp add: eq-iff sum-subset-iff*)

definition *is-sum* :: V \Rightarrow bool
where *is-sum z = ($\exists x. z =$ Inl x \vee z = Inr x)*

definition *sum-case* :: (V \Rightarrow 'a) \Rightarrow (V \Rightarrow 'a) \Rightarrow V \Rightarrow 'a
where

sum-case f g a \equiv
THE z. ($\forall x. a =$ Inl x \longrightarrow z = f x) \wedge ($\forall y. a =$ Inr y \longrightarrow z = g y) \wedge (\neg is-sum a \longrightarrow z = undefined)

lemma *sum-case-Inl* [*simp*]: *sum-case f g (Inl x) = f x*
by (*simp add: sum-case-def is-sum-def*)

lemma *sum-case-Inr* [*simp*]: *sum-case f g (Inr y) = g y*
by (*simp add: sum-case-def is-sum-def*)

lemma *sum-case-non* [*simp*]: *\neg is-sum a \implies sum-case f g a = undefined*
by (*simp add: sum-case-def is-sum-def*)

lemma *is-sum-cases*: *($\exists x. z =$ Inl x \vee z = Inr x) \vee \neg is-sum z*

by (auto simp: is-sum-def)

lemma *sum-case-split*:

P (sum-case f g a) \longleftrightarrow $(\forall x. a = \text{Inl } x \longrightarrow P(f\ x)) \wedge (\forall y. a = \text{Inr } y \longrightarrow P(g\ y)) \wedge (\neg \text{is-sum } a \longrightarrow P \text{ undefined})$

by (cases is-sum a) (auto simp: is-sum-def)

lemma *sum-case-split-asm*:

P (sum-case f g a) \longleftrightarrow $\neg ((\exists x. a = \text{Inl } x \wedge \neg P(f\ x)) \vee (\exists y. a = \text{Inr } y \wedge \neg P(g\ y)) \vee (\neg \text{is-sum } a \wedge \neg P \text{ undefined}))$

by (auto simp: sum-case-split)

2.3.3 Applications of disjoint sums and pairs: general union theorems for small sets

lemma *small-Un*:

assumes X : small X and Y : small Y

shows small $(X \cup Y)$

proof –

obtain $x\ y$ where $\text{elts } x \approx X$ $\text{elts } y \approx Y$

by (meson assms small-epoll)

then have $X \cup Y \lesssim \text{Inl } \langle \text{elts } x \rangle \cup \text{Inr } \langle \text{elts } y \rangle$

by (metis (mono-tags, lifting) Inr-Inl-iff Un-lepoll-mono disjnt-iff eqpoll-imp-lepoll eqpoll-sym f-inv-into-f inj-on-Inl inj-on-Inr inj-on-image-lepoll-2)

then show ?thesis

by (metis lepoll-iff replacement small-elts small-sup-iff smaller-than-small)

qed

lemma *small-UN* [simp,intro]:

assumes A : small A and B : $\bigwedge x. x \in A \implies \text{small } (B\ x)$

shows small $(\bigcup_{x \in A} B\ x)$

proof –

obtain a where $\text{elts } a \approx A$

by (meson assms small-epoll)

then obtain f where f : bij-betw f $(\text{elts } a)$ A

using eqpoll-def by blast

have $\exists y. \text{elts } y \approx B\ x$ if $x \in A$ for x

using B small-epoll that by blast

then obtain g where g : $\bigwedge x. x \in A \implies \text{elts } (g\ x) \approx B\ x$

by metis

have $\text{sm: small } (\text{Sigma } (\text{elts } a) (\text{elts } \circ g \circ f))$

by simp

have $(\bigcup_{x \in A} B\ x) \lesssim \text{Sigma } A\ B$

by (metis image-lepoll snd-image-Sigma)

also have ... $\lesssim \text{Sigma } (\text{elts } a) (\text{elts } \circ g \circ f)$

by (smt (verit) Sigma-epoll-cong bij-betw-iff-bijections comp-apply eqpoll-imp-lepoll eqpoll-sym $f\ g$)

finally show ?thesis

using lepoll-small sm by blast

qed

lemma *small-Union* [*simp,intro*]:
 assumes $\mathcal{A} \subseteq \text{Collect small small } \mathcal{A}$
 shows *small* $(\bigcup \mathcal{A})$
 using *small-UN* [*of } \mathcal{A} \lambda x. x*] *assms* **by** (*simp add: subset-iff*)

2.4 Generalised function space and lambda

definition *VLambda* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V$
 where *VLambda* $A b \equiv \text{set } ((\lambda x. \langle x, b \ x \rangle) \text{ ` } \text{elts } A)$

definition *app* :: $[V, V] \Rightarrow V$
 where *app* $f x \equiv \text{THE } y. \langle x, y \rangle \in \text{elts } f$

lemma *beta* [*simp*]:
 assumes $x \in \text{elts } A$
 shows *app* (*VLambda* $A b$) $x = b \ x$
 using *assms* **by** (*auto simp: VLambda-def app-def*)

definition *VPi* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V$
 where *VPi* $A B \equiv \text{set } \{f \in \text{elts } (\text{VPow}(\text{VSigma } A B)). \text{elts } A \leq \text{Domain } (\text{pairs } f) \wedge \text{single-valued } (\text{pairs } f)\}$

lemma *VPi-I*:
 assumes $\bigwedge x. x \in \text{elts } A \implies b \ x \in \text{elts } (B \ x)$
 shows *VLambda* $A b \in \text{elts } (\text{VPi } A B)$
 proof (*clarsimp simp: VPi-def, intro conjI impI*)
 show *VLambda* $A b \leq \text{VSigma } A B$
 by (*auto simp: assms VLambda-def split: if-split-asm*)
 show $\text{elts } A \subseteq \text{Domain } (\text{pairs } (\text{VLambda } A b))$
 by (*force simp: VLambda-def pairs-iff-elts*)
 show *single-valued* $(\text{pairs } (\text{VLambda } A b))$
 by (*auto simp: VLambda-def single-valued-def pairs-iff-elts*)
 show *small* $\{f. f \leq \text{VSigma } A B \wedge \text{elts } A \subseteq \text{Domain } (\text{pairs } f) \wedge \text{single-valued } (\text{pairs } f)\}$
 by (*metis (mono-tags, lifting) down VPow-iff mem-Collect-eq subsetI*)
qed

lemma *apply-pair*:
 assumes $f: f \in \text{elts } (\text{VPi } A B)$ **and** $x: x \in \text{elts } A$
 shows $\langle x, \text{app } f \ x \rangle \in \text{elts } f$
 proof –
 have $x \in \text{Domain } (\text{pairs } f)$
 by (*metis (no-types, lifting) VPi-def assms elts-of-set empty-iff mem-Collect-eq subsetD*)
 then obtain y **where** $y: \langle x, y \rangle \in \text{elts } f$
 using *pairs-iff-elts* **by** *auto*
 show *?thesis*

unfolding *app-def*
proof (*rule theI*)
show $\langle x, y \rangle \in \text{elts } f$
by (*rule y*)
show $z = y$ **if** $\langle x, z \rangle \in \text{elts } f$ **for** z
using *f unfolding VPi-def*
by (*metis (mono-tags, lifting) that elts-of-set empty-iff mem-Collect-eq pairs-iff-elts single-valued-def y*)
qed
qed

lemma *VPi-D*:
assumes $f: f \in \text{elts } (VPi\ A\ B)$ **and** $x: x \in \text{elts } A$
shows $\text{app } f\ x \in \text{elts } (B\ x)$
proof –
have $f \leq VSigma\ A\ B$
by (*metis (no-types, lifting) VPi-def elts-of-set empty-iff f VPow-iff mem-Collect-eq*)
then show *?thesis*
using *apply-pair [OF assms]* **by** *blast*
qed

lemma *VPi-memberD*:
assumes $f: f \in \text{elts } (VPi\ A\ B)$ **and** $p: p \in \text{elts } f$
obtains x **where** $x \in \text{elts } A$ $p = \langle x, \text{app } f\ x \rangle$
proof –
have $f \leq VSigma\ A\ B$
by (*metis (no-types, lifting) VPi-def elts-of-set empty-iff f VPow-iff mem-Collect-eq*)
then obtain $x\ y$ **where** $p = \langle x, y \rangle$ $x \in \text{elts } A$
using p **by** *blast*
then have $y = \text{app } f\ x$
by (*metis (no-types, lifting) VPi-def apply-pair elts-of-set equals0D f mem-Collect-eq p pairs-iff-elts single-valuedD*)
then show *thesis*
using $\langle p = \langle x, y \rangle \rangle$ $\langle x \in \text{elts } A \rangle$ **that** **by** *blast*
qed

lemma *fun-ext*:
assumes $f \in \text{elts } (VPi\ A\ B)$ $g \in \text{elts } (VPi\ A\ B)$ $\bigwedge x. x \in \text{elts } A \implies \text{app } f\ x = \text{app } g\ x$
shows $f = g$
by (*metis VPi-memberD V-equalityI apply-pair assms*)

lemma *eta[simp]*:
assumes $f \in \text{elts } (VPi\ A\ B)$
shows $VLambda\ A\ ((\text{app})f) = f$
proof (*rule fun-ext [OF - assms]*)
show $VLambda\ A\ (\text{app } f) \in \text{elts } (VPi\ A\ B)$
using *VPi-D VPi-I assms* **by** *auto*
qed *auto*

lemma *fst-pairs-VLambda*: $\text{fst} \text{ ' pairs } (VLambda A f) = \text{elts } A$
by (*force simp: VLambda-def pairs-def*)

lemma *snd-pairs-VLambda*: $\text{snd} \text{ ' pairs } (VLambda A f) = f \text{ ' elts } A$
by (*force simp: VLambda-def pairs-def*)

lemma *VLambda-eq-D1*: $VLambda A f = VLambda B g \implies A = B$
by (*metis ZFC-in-HOL.ext fst-pairs-VLambda*)

lemma *VLambda-eq-D2*: $\llbracket VLambda A f = VLambda A g; x \in \text{elts } A \rrbracket \implies f x = g x$
by (*metis beta*)

2.5 Transitive closure of a set

definition *TC* :: $V \Rightarrow V$
where $TC \equiv \text{transrec } (\lambda f x. x \sqcup \llbracket f \text{ ' elts } x \rrbracket)$

lemma *TC*: $TC a = a \sqcup \llbracket TC \text{ ' elts } a \rrbracket$
by (*metis (no-types, lifting) SUP-cong TC-def restrict-apply' transrec*)

lemma *TC-0* [*simp*]: $TC 0 = 0$
by (*metis TC ZFC-in-HOL.Sup-empty elts-0 image-is-empty sup-V-0-left*)

lemma *arg-subset-TC*: $a \leq TC a$
by (*metis (no-types) TC sup-ge1*)

lemma *Transset-TC*: $\text{Transset}(TC a)$

proof (*induction a rule: eps-induct*)

case (*step x*)

have 1: $v \in \text{elts } (TC x)$ **if** $v \in \text{elts } u$ $u \in \text{elts } x$ **for** $u v$

using *that unfolding TC [of x]*

using *arg-subset-TC by fastforce*

have 2: $v \in \text{elts } (TC x)$ **if** $v \in \text{elts } u$ $\exists x \in \text{elts } x. u \in \text{elts } (TC x)$ **for** $u v$

using *that step unfolding TC [of x] Transset-def by auto*

show *?case*

unfolding *Transset-def*

by (*subst TC*) (*force intro: 1 2*)

qed

lemma *TC-least*: $\llbracket \text{Transset } x; a \leq x \rrbracket \implies TC a \leq x$

proof (*induction a rule: eps-induct*)

case (*step y*)

show *?case*

proof (*cases y=0*)

case *True*

then show *?thesis*

```

    by auto
  next
  case False
  have  $\sqcup (TC \text{ ' } elts \ y) \leq x$ 
  proof (rule cSup-least)
    show  $TC \text{ ' } elts \ y \neq \{\}$ 
    using False by auto
    show  $z \leq x$  if  $z \in TC \text{ ' } elts \ y$  for  $z$ 
    using that by (metis Transset-def image-iff step.IH step.prem1 usubsetD)
  qed
  then show ?thesis
  by (simp add: step TC [of y])
qed
qed

```

definition *less-TC* (infix $\langle \sqsubset \rangle$ 50)
 where $x \sqsubset y \equiv x \in elts (TC \ y)$

definition *le-TC* (infix $\langle \sqsubseteq \rangle$ 50)
 where $x \sqsubseteq y \equiv x \sqsubset y \vee x=y$

lemma *less-TC-imp-not-le*: $x \sqsubset a \implies \neg a \leq x$
proof (induction a arbitrary: x rule: eps-induct)
 case (step a)
 then show ?case
 unfolding TC[of a] less-TC-def
 using Transset-TC Transset-def by force
 qed

lemma *non-TC-less-0* [iff]: $\neg (x \sqsubset 0)$
 using less-TC-imp-not-le by blast

lemma *less-TC-iff*: $x \sqsubset y \longleftrightarrow (\exists z \in elts \ y. x \sqsubseteq z)$
 by (auto simp: less-TC-def le-TC-def TC [of y])

lemma *nonzero-less-TC*: $x \neq 0 \implies 0 \sqsubset x$
 by (metis eps-induct le-TC-def less-TC-iff trad-foundation)

lemma *less-irrefl-TC* [simp]: $\neg x \sqsubset x$
 using less-TC-imp-not-le by blast

lemma *less-asym-TC*: $\llbracket x \sqsubset y; y \sqsubset x \rrbracket \implies False$
 by (metis TC-least Transset-TC Transset-def antisym-conv less-TC-def less-TC-imp-not-le order-refl)

lemma *le-antisym-TC*: $\llbracket x \sqsubseteq y; y \sqsubseteq x \rrbracket \implies x = y$
 using le-TC-def less-asym-TC by auto

lemma *less-le-TC*: $x \sqsubset y \longleftrightarrow x \sqsubseteq y \wedge x \neq y$

```

using le-TC-def less-asym-TC by blast

lemma less-imp-le-TC [iff]:  $x \sqsubset y \implies x \sqsubseteq y$ 
by (simp add: le-TC-def)

lemma le-TC-refl [iff]:  $x \sqsubseteq x$ 
by (simp add: le-TC-def)

lemma le-TC-trans [trans]:  $\llbracket x \sqsubseteq y; y \sqsubseteq z \rrbracket \implies x \sqsubseteq z$ 
by (smt (verit, best) TC-least Transset-TC Transset-def le-TC-def less-TC-def vsubsetD)

context order
begin

lemma nless-le-TC:  $(\neg a \sqsubset b) \longleftrightarrow (\neg a \sqsubseteq b) \vee a = b$ 
using le-TC-def less-asym-TC by blast

lemma eq-refl-TC:  $x = y \implies x \sqsubseteq y$ 
by simp

local-setup ‹
  HOL-Order-Tac.declare-order {
    ops = {eq = @{term ‹(=) :: V ⇒ V ⇒ bool›}, le = @{term ‹(⊆)›}, lt =
@{term ‹(⊂)›}},
    thms = {trans = @{thm le-TC-trans}, refl = @{thm le-TC-refl}, eqD1 = @{thm
eq-refl-TC},
    eqD2 = @{thm eq-refl-TC[OF sym]}, antisym = @{thm le-antisym-TC},
    contr = @{thm notE}},
    conv-thms = {less-le = @{thm eq-reflection[OF less-le-TC]},
    nless-le = @{thm eq-reflection[OF nless-le-TC]}}
  }
›

end

lemma less-TC-trans [trans]:  $\llbracket x \sqsubset y; y \sqsubset z \rrbracket \implies x \sqsubset z$ 
and less-le-TC-trans:  $\llbracket x \sqsubset y; y \sqsubseteq z \rrbracket \implies x \sqsubset z$ 
and le-less-TC-trans [trans]:  $\llbracket x \sqsubseteq y; y \sqsubset z \rrbracket \implies x \sqsubset z$ 
by simp-all

lemma TC-sup-distrib:  $TC (x \sqcup y) = TC x \sqcup TC y$ 
by (simp add: Sup-Un-distrib TC [of x ⊔ y] TC [of x] TC [of y] image-Un
sup.assoc sup-left-commute)

lemma TC-Sup-distrib:
assumes small X shows  $TC (\sqcup X) = \sqcup (TC ` X)$ 
proof –

```

have $\sqcup X \leq \sqcup (TC \text{ ' } X)$
using *arg-subset-TC* **by** *fastforce*
moreover have $\sqcup (\bigcup_{x \in X}. TC \text{ ' } elts\ x) \leq \sqcup (TC \text{ ' } X)$
using *assms*
by *clarsimp (meson TC-least Transset-TC Transset-def arg-subset-TC replacement vsubsetD)*
ultimately
have $\sqcup X \sqcup \sqcup (\bigcup_{x \in X}. TC \text{ ' } elts\ x) \leq \sqcup (TC \text{ ' } X)$
by *simp*
moreover have $\sqcup (TC \text{ ' } X) \leq \sqcup X \sqcup \sqcup (\bigcup_{x \in X}. TC \text{ ' } elts\ x)$
proof (*clarsimp simp add: Sup-le-iff assms*)
show $\exists x \in X. y \in elts\ x$
if $x \in X\ y \in elts\ (TC\ x) \forall x \in X. \forall u \in elts\ x. y \notin elts\ (TC\ u)$ **for** $x\ y$
using that by (*auto simp: TC [of x]*)
qed
ultimately show *?thesis*
using *Sup-Un-distrib TC [of $\sqcup X$] image-Union assms*
by (*simp add: image-Union inf-sup-aci(5) sup.absorb-iff2*)
qed

lemma *TC'*: $TC\ x = x \sqcup TC\ (\sqcup (elts\ x))$
by (*simp add: TC [of x] TC-Sup-distrib*)

lemma *TC-eq-0-iff* [*simp*]: $TC\ x = 0 \longleftrightarrow x = 0$
using *arg-subset-TC* **by** *fastforce*

A distinctive induction principle

lemma *TC-induct-down-lemma*:
assumes *ab*: $a \sqsubset b$ **and** *base*: $b \leq d$
and *step*: $\bigwedge y\ z. \llbracket y \sqsubset b; y \in elts\ d; z \in elts\ y \rrbracket \implies z \in elts\ d$
shows $a \in elts\ d$
proof –
have *Transset* ($TC\ b \sqcap d$)
using *Transset-TC*
unfolding *Transset-def*
by (*metis inf.bounded-iff less-TC-def less-eq-V-def local.step subsetI vsubsetD*)
moreover have $b \leq TC\ b \sqcap d$
by (*simp add: arg-subset-TC base*)
ultimately show *?thesis*
using *TC-least [THEN vsubsetD] ab unfolding less-TC-def*
by (*meson TC-least le-inf-iff vsubsetD*)
qed

lemma *TC-induct-down* [*consumes 1, case-names base step small*]:
assumes $a \sqsubset b$
and $\bigwedge y. y \in elts\ b \implies P\ y$
and $\bigwedge y\ z. \llbracket y \sqsubset b; P\ y; z \in elts\ y \rrbracket \implies P\ z$
and *small* (*Collect P*)
shows $P\ a$

using *TC-induct-down-lemma* [of *a b set (Collect P)*] *assms*
by (*metis elts-of-set mem-Collect-eq vsubsetI*)

2.6 Rank of a set

definition *rank* :: $V \Rightarrow V$
where *rank a* \equiv *transrec* ($\lambda f x. \text{set } (\bigcup y \in \text{elts } x. \text{elts } (\text{succ}(f y)))$) *a*

lemma *rank*: *rank a* = *set*($\bigcup y \in \text{elts } a. \text{elts } (\text{succ}(\text{rank } y))$)
by (*subst rank-def [THEN def-transrec], simp*)

lemma *rank-Sup*: *rank a* = $\bigsqcup ((\lambda y. \text{succ}(\text{rank } y)) ` \text{elts } a)$
by (*metis elts-Sup image-image rank replacement set-of-elts small-elts*)

lemma *Ord-rank* [*simp*]: *Ord*(*rank a*)

proof (*induction a rule: eps-induct*)

case (*step x*)

then show ?*case*

unfolding *rank-Sup* [of *x*]

by (*metis (mono-tags, lifting) Ord-Sup Ord-succ imageE*)

qed

lemma *rank-of-Ord*: *Ord i* \implies *rank i* = *i*

by (*induction rule: Ord-induct*) (*metis (no-types, lifting) Ord-equality SUP-cong rank-Sup*)

lemma *Ord-iff-rank*: *Ord x* \longleftrightarrow *rank x* = *x*

using *Ord-rank* [of *x*] *rank-of-Ord* **by** *fastforce*

lemma *rank-lt*: $a \in \text{elts } b \implies \text{rank } a < \text{rank } b$

by (*metis Ord-linear2 Ord-rank ZFC-in-HOL.SUP-le-iff rank-Sup replacement small-elts succ-le-iff order.irrefl*)

lemma *rank-0* [*simp*]: *rank 0* = 0

using *transrec Ord-0 rank-def rank-of-Ord* **by** *presburger*

lemma *rank-succ* [*simp*]: *rank*(*succ x*) = *succ*(*rank x*)

proof (*rule order-antisym*)

show *rank* (*succ x*) \leq *succ* (*rank x*)

by (*metis (no-types, lifting) Sup-insert elts-of-set elts-succ image-insert rank small-UN small-elts subset-insertI sup.orderE vsubsetI*)

show *succ* (*rank x*) \leq *rank* (*succ x*)

by (*metis (mono-tags, lifting) ZFC-in-HOL.Sup-upper elts-succ image-insert insertI1 rank-Sup replacement small-elts*)

qed

lemma *rank-mono*: $a \leq b \implies \text{rank } a \leq \text{rank } b$

using *rank* [of *a*] *rank* [of *b*] *small-UN* **by** *force*

lemma *VsetI*: $\text{rank } b \sqsubset i \implies b \in \text{elts } (Vset\ i)$
proof (*induction i arbitrary; b rule: eps-induct*)
 case (*step x*)
 then consider $\text{rank } b \in \text{elts } x \mid (\exists y \in \text{elts } x. \text{rank } b \in \text{elts } (TC\ y))$
 using *le-TC-def less-TC-def less-TC-iff* **by** *fastforce*
 then have $\exists y \in \text{elts } x. b \leq Vset\ y$
proof *cases*
 case *1*
 then have $b \leq Vset\ (\text{rank } b)$
 unfolding *less-eq-V-def subset-iff*
 by (*meson Ord-mem-iff-lt Ord-rank le-TC-refl less-TC-iff rank-lt step.IH*)
 then show *?thesis*
 using *1* **by** *blast*
next
 case *2*
 then show *?thesis*
 using *step.IH*
 unfolding *less-eq-V-def subset-iff less-TC-def*
 by (*meson Ord-mem-iff-lt Ord-rank Transset-TC Transset-def rank-lt vsubsetD*)
qed
then show *?case*
 by (*simp add: Vset [of x]*)
qed

lemma *Ord-VsetI*: $\llbracket \text{Ord } i; \text{rank } b < i \rrbracket \implies b \in \text{elts } (Vset\ i)$
by (*meson Ord-mem-iff-lt Ord-rank VsetI arg-subset-TC less-TC-def vsubsetD*)

lemma *arg-le-Vset-rank*: $a \leq Vset(\text{rank } a)$
by (*simp add: Ord-VsetI rank-lt vsubsetI*)

lemma *two-in-Vset*:
obtains α **where** $x \in \text{elts } (Vset\ \alpha) \ y \in \text{elts } (Vset\ \alpha)$
by (*metis Ord-rank Ord-VsetI elts-of-set insert-iff rank-lt small-elts small-insert-iff*)

lemma *rank-eq-0-iff [simp]*: $\text{rank } x = 0 \longleftrightarrow x=0$
using *arg-le-Vset-rank* **by** *fastforce*

lemma *small-ranks-imp-small*:
assumes *small (rank ' A)* **shows** *small A*
proof –
 define $i \equiv \text{set } (\bigcup (\text{elts } ' (rank\ ' A)))$
 have *Ord i*
 unfolding *i-def* **using** *Ord-Union Ord-rank assms imageE* **by** *blast*
 have $*$: $Vset\ (\text{rank } x) \leq (Vset\ i)$ **if** $x \in A$ **for** x
 unfolding *i-def* **by** (*metis Ord-rank Sup-V-def ZFC-in-HOL.Sup-upper Vfrom-mono*
assms imageI le-less that)
 have $A \subseteq \text{elts } (VPow\ (Vset\ i))$
 by (*meson * VPow-iff arg-le-Vset-rank order.trans subsetI*)
then show *?thesis*

using *down by blast*
qed

lemma *rank-Union*: $\text{rank}(\bigsqcup A) = \bigsqcup (\text{rank } ' A)$

proof (*rule order-antisym*)

have $\text{elts}(\bigsqcup y \in \text{elts}(\bigsqcup A). \text{succ}(\text{rank } y)) \subseteq \text{elts}(\bigsqcup (\text{rank } ' A))$

by *clarsimp* (*meson Ord-mem-iff-lt Ord-rank less-V-def rank-lt vsubsetD*)

then show $\text{rank}(\bigsqcup A) \leq \bigsqcup (\text{rank } ' A)$

by (*metis less-eq-V-def rank-Sup*)

show $\bigsqcup (\text{rank } ' A) \leq \text{rank}(\bigsqcup A)$

proof (*cases small A*)

case *True*

then show *?thesis*

by (*simp add: ZFC-in-HOL.SUP-le-iff ZFC-in-HOL.Sup-upper rank-mono*)

next

case *False*

then have $\neg \text{small}(\text{rank } ' A)$

using *small-ranks-imp-small* by *blast*

then show *?thesis*

by *blast*

qed

qed

lemma *small-bounded-rank*: $\text{small} \{x. \text{rank } x \in \text{elts } a\}$

proof –

have $\{x. \text{rank } x \in \text{elts } a\} \subseteq \{x. \text{rank } x \sqsubset a\}$

using *less-TC-iff* by *auto*

also have $\dots \subseteq \text{elts}(V\text{set } a)$

using *VsetI* by *blast*

finally show *?thesis*

using *down by simp*

qed

lemma *small-bounded-rank-le*: $\text{small} \{x. \text{rank } x \leq a\}$

using *small-bounded-rank* [*of VPow a*] *VPow-iff* [*of - a*] by *simp*

lemma *TC-rank-lt*: $a \sqsubset b \implies \text{rank } a < \text{rank } b$

proof (*induction rule: TC-induct-down*)

case (*base y*)

then show *?case*

by (*simp add: rank-lt*)

next

case (*step y z*)

then show *?case*

using *less-trans rank-lt* by *blast*

next

case *small*

show *?case*

using *smaller-than-small* [*OF small-bounded-rank-le* [*of rank b*]]

by (*simp add: Collect-mono less-V-def*)
qed

lemma *TC-rank-mem*: $x \sqsubset y \implies \text{rank } x \in \text{elts } (\text{rank } y)$
 by (*simp add: Ord-mem-iff-lt TC-rank-lt*)

lemma *wf-TC-less*: $\text{wf } \{(x,y). x \sqsubset y\}$
proof (*rule wf-subset [OF wf-inv-image [OF foundation, of rank]]*)
 show $\{(x, y). x \sqsubset y\} \subseteq \text{inv-image } \{(x, y). x \in \text{elts } y\}$ *rank*
 by (*auto simp: TC-rank-mem inv-image-def*)
qed

lemma *less-TC-minimal*:
 assumes $P a$
 obtains x where $P x \wedge x \sqsubseteq a \wedge y. y \sqsubset x \implies \neg P y$
 using *wfE-min'* [*OF wf-TC-less, of {x. P x \wedge x \sqsubseteq a}*]
 by *simp (metis le-TC-def less-le-TC-trans assms)*

lemma *Vfrom-rank-eq*: $V\text{from } A (\text{rank}(x)) = V\text{from } A x$
proof (*rule order-antisym*)
 show $V\text{from } A (\text{rank } x) \leq V\text{from } A x$
proof (*induction x rule: eps-induct*)
 case (*step x*)
 have $(\bigsqcup_{j \in \text{elts } (\text{rank } x)}. VPow (V\text{from } A j)) \leq (\bigsqcup_{j \in \text{elts } x}. VPow (V\text{from } A j))$
 apply (*rule Sup-least*)
 apply (*clarsimp simp add: rank [of x]*)
 by (*meson Ord-in-Ord Ord-rank OrdmemD Vfrom-mono order.trans less-imp-le order.refl step*)
 then show ?*case*
 by (*simp add: Vfrom [of - x] Vfrom [of - rank(x)] sup.coboundedI2*)
qed
 show $V\text{from } A x \leq V\text{from } A (\text{rank } x)$
proof (*induction x rule: eps-induct*)
 case (*step x*)
 have $(\bigsqcup_{j \in \text{elts } x}. VPow (V\text{from } A j)) \leq (\bigsqcup_{j \in \text{elts } (\text{rank } x)}. VPow (V\text{from } A j))$
 using *step.IH TC-rank-mem less-TC-iff* by *force*
 then show ?*case*
 by (*simp add: Vfrom [of - x] Vfrom [of - rank(x)] sup.coboundedI2*)
qed
qed

lemma *Vfrom-succ*: $V\text{from } A (\text{succ}(i)) = A \sqcup VPow(V\text{from } A i)$
 by (*metis Ord-rank Vfrom-rank-eq Vfrom-succ-Ord rank-succ*)

lemma *Vset-succ-TC*:
 assumes $x \in \text{elts } (V\text{set } (ZFC\text{-in-HOL.succ } k))$ $u \sqsubset x$
 shows $u \in \text{elts } (V\text{set } k)$

using *assms*
using *TC-least Transset-Vfrom Vfrom-succ less-TC-def* **by** *auto*

2.7 Cardinal Numbers

We extend the membership relation to a wellordering

definition *VWO* :: $(V \times V)$ set
where $VWO \equiv @r. \{(x,y). x \in \text{elts } y\} \subseteq r \wedge \text{Well-order } r \wedge \text{Field } r = \text{UNIV}$

lemma *VWO*: $\{(x,y). x \in \text{elts } y\} \subseteq VWO \wedge \text{Well-order } VWO \wedge \text{Field } VWO = \text{UNIV}$

unfolding *VWO-def*
by (*metis (mono-tags, lifting) VWO-def foundation someI-ex total-well-order-extension*)

lemma *wf-VWO*: $\text{wf}(VWO - \text{Id})$
using *VWO well-order-on-def* **by** *blast*

lemma *wf-Ord-less*: $\text{wf} \{(x, y). \text{Ord } y \wedge x < y\}$
by (*metis (no-types, lifting) Ord-mem-iff-lt eps-induct wfpUNIVI wfp-def*)

lemma *refl-VWO*: $\text{refl } VWO$
using *VWO order-on-defs* **by** *fastforce*

lemma *trans-VWO*: $\text{trans } VWO$
using *VWO* **by** (*simp add: VWO wo-rel.TRANS wo-rel-def*)

lemma *antisym-VWO*: $\text{antisym } VWO$
using *VWO* **by** (*simp add: VWO wo-rel.ANTISYM wo-rel-def*)

lemma *total-VWO*: $\text{total } VWO$
using *VWO* **by** (*metis wo-rel.TOTAL wo-rel.intro*)

lemma *total-VWOId*: $\text{total } (VWO - \text{Id})$
by (*simp add: total-VWO*)

lemma *Linear-order-VWO*: $\text{Linear-order } VWO$
using *VWO well-order-on-def* **by** *blast*

lemma *wo-rel-VWO*: $\text{wo-rel } VWO$
using *VWO wo-rel-def* **by** *blast*

2.7.1 Transitive Closure and VWO

lemma *mem-imp-VWO*: $x \in \text{elts } y \implies (x,y) \in VWO$
using *VWO* **by** *blast*

lemma *less-TC-imp-VWO*: $x \sqsubset y \implies (x,y) \in VWO$
unfolding *less-TC-def*

proof (*induction y arbitrary: x rule: eps-induct*)

case (*step* $y' u$)
then consider $u \in \text{elts } y' \mid v$ **where** $v \in \text{elts } y' u \in \text{elts } (TC v)$
by (*auto simp: TC [of y']*)
then show *?case*
proof cases
case 2
then show *?thesis*
by (*meson mem-imp-VWO step.IH transD trans-VWO*)
qed (*use mem-imp-VWO in blast*)
qed

lemma *le-TC-imp-VWO*: $x \sqsubseteq y \implies (x, y) \in VWO$
by (*metis Diff-iff Linear-order-VWO Linear-order-in-diff-Id UNIV-I VWO le-TC-def less-TC-imp-VWO*)

lemma *le-TC-0-iff [simp]*: $x \sqsubseteq 0 \longleftrightarrow x = 0$
by (*simp add: le-TC-def*)

lemma *less-TC-succ*: $x \sqsubset \text{succ } \beta \longleftrightarrow x \sqsubset \beta \vee x = \beta$
by (*metis elts-succ insert-iff le-TC-def less-TC-iff*)

lemma *le-TC-succ*: $x \sqsubseteq \text{succ } \beta \longleftrightarrow x \sqsubseteq \beta \vee x = \text{succ } \beta$
by (*simp add: le-TC-def less-TC-succ*)

lemma *Transset-TC-eq [simp]*: $\text{Transset } x \implies TC x = x$
by (*simp add: TC-least arg-subset-TC eq-iff*)

lemma *Ord-TC-less-iff*: $\llbracket \text{Ord } \alpha; \text{Ord } \beta \rrbracket \implies \beta \sqsubset \alpha \longleftrightarrow \beta < \alpha$
by (*metis Ord-def Ord-mem-iff-lt Transset-TC-eq less-TC-def*)

lemma *Ord-mem-iff-less-TC*: $\text{Ord } l \implies k \in \text{elts } l \longleftrightarrow k \sqsubset l$
by (*simp add: Ord-def less-TC-def*)

lemma *le-TC-Ord*: $\llbracket \beta \sqsubseteq \alpha; \text{Ord } \alpha \rrbracket \implies \text{Ord } \beta$
by (*metis Ord-def Ord-in-Ord Transset-TC-eq le-TC-def less-TC-def*)

lemma *Ord-less-TC-mem*:
assumes $\text{Ord } \alpha$ $\beta \sqsubset \alpha$ **shows** $\beta \in \text{elts } \alpha$
using *Ord-def assms less-TC-def* **by auto**

lemma *VWO-TC-le*: $\llbracket \text{Ord } \alpha; \text{Ord } \beta; (\beta, \alpha) \in VWO \rrbracket \implies \beta \sqsubseteq \alpha$

proof (*induct* α *arbitrary:* β *rule: Ord-induct*)
case (*step* α)
then show *?case*

by (*metis DiffI IdD Linear-order-VWO Linear-order-in-diff-Id Ord-linear Ord-mem-iff-less-TC VWO iso-tuple-UNIV-I le-TC-def mem-imp-VWO*)

qed

lemma *VWO-iff-Ord-le [simp]*: $\llbracket \text{Ord } \alpha; \text{Ord } \beta \rrbracket \implies (\beta, \alpha) \in VWO \longleftrightarrow \beta \leq \alpha$

by (metis VWO-TC-le Ord-TC-less-iff le-TC-def le-TC-imp-VWO le-less)

lemma zero-TC-le [iff]: $0 \sqsubseteq y$
using le-TC-def nonzero-less-TC by auto

lemma succ-le-TC-iff: $\text{Ord } j \implies \text{succ } i \sqsubseteq j \longleftrightarrow i \sqsubseteq j$
by (metis Ord-in-Ord Ord-linear Ord-mem-iff-less-TC Ord-succ le-TC-def less-TC-succ less-asm-TC)

lemma VWO-0-iff [simp]: $(x, 0) \in \text{VWO} \longleftrightarrow x = 0$
proof
show $x = 0$ if $(x, 0) \in \text{VWO}$
using zero-TC-le [of x] le-TC-imp-VWO that
by (metis DiffI Linear-order-VWO Linear-order-in-diff-Id UNIV-I VWO pair-in-Id-conv)
qed auto

lemma VWO-antisym:
assumes $(x, y) \in \text{VWO}$ $(y, x) \in \text{VWO}$ shows $x = y$
by (metis Diff-iff IdD Linear-order-VWO Linear-order-in-diff-Id UNIV-I VWO assms)

2.7.2 Relation VWF

definition VWF where $\text{VWF} \equiv \text{VWO} - \text{Id}$

lemma wf-VWF [iff]: $\text{wf } \text{VWF}$
by (simp add: VWF-def wf-VWO)

lemma trans-VWF [iff]: $\text{trans } \text{VWF}$
by (simp add: VWF-def antisym-VWO trans-VWO trans-diff-Id)

lemma asym-VWF [iff]: $\text{asym } \text{VWF}$
by (metis wf-VWF wf-imp-asym)

lemma total-VWF [iff]: $\text{total } \text{VWF}$
using VWF-def total-VWOId by auto

lemma total-on-VWF [iff]: $\text{total-on } A \text{ VWF}$
by (meson UNIV-I total-VWF total-on-def)

lemma VWF-asym:
assumes $(x, y) \in \text{VWF}$ $(y, x) \in \text{VWF}$ shows *False*
using VWF-def assms wf-VWO wf-not-sym by fastforce

lemma VWF-non-refl [iff]: $(x, x) \notin \text{VWF}$
by simp

lemma VWF-iff-Ord-less [simp]: $[\text{Ord } \alpha; \text{Ord } \beta] \implies (\alpha, \beta) \in \text{VWF} \longleftrightarrow \alpha < \beta$
by (simp add: VWF-def less-V-def)

lemma *mem-imp-VWF*: $x \in \text{elts } y \implies (x,y) \in \text{VWF}$
using *VWF-def mem-imp-VWO* **by** *fastforce*

2.8 Order types

definition *ordermap* :: $'a \text{ set} \Rightarrow ('a \times 'a) \text{ set} \Rightarrow 'a \Rightarrow V$
where *ordermap* $A \ r \equiv \text{wfrec } r \ (\lambda f x. \text{set } (f \ ` \ \{y \in A. (y,x) \in r\}))$

definition *ordertype* :: $'a \text{ set} \Rightarrow ('a \times 'a) \text{ set} \Rightarrow V$
where *ordertype* $A \ r \equiv \text{set } (\text{ordermap } A \ r \ ` \ A)$

lemma *ordermap-type*:
 $\text{small } A \implies \text{ordermap } A \ r \in A \rightarrow \text{elts } (\text{ordertype } A \ r)$
by (*simp add: ordertype-def*)

lemma *ordermap-in-ordertype* [*intro*]: $\llbracket a \in A; \text{small } A \rrbracket \implies \text{ordermap } A \ r \ a \in \text{elts } (\text{ordertype } A \ r)$
by (*simp add: ordertype-def*)

lemma *ordermap*: $\text{wf } r \implies \text{ordermap } A \ r \ a = \text{set } (\text{ordermap } A \ r \ ` \ \{y \in A. (y,a) \in r\})$
unfolding *ordermap-def*
by (*auto simp: wfrec-fixpoint adm-wf-def*)

lemma *wf-Ord-ordermap* [*iff*]: **assumes** $\text{wf } r \ \text{trans } r$ **shows** $\text{Ord } (\text{ordermap } A \ r \ x)$

using $\langle \text{wf } r \rangle$

proof (*induction x rule: wf-induct-rule*)

case (*less u*)

have $\text{Transset } (\text{set } (\text{ordermap } A \ r \ ` \ \{y \in A. (y, u) \in r\}))$

proof (*clarsimp simp add: Transset-def*)

show $x \in \text{ordermap } A \ r \ ` \ \{y \in A. (y, u) \in r\}$

if $\text{small } (\text{ordermap } A \ r \ ` \ \{y \in A. (y, u) \in r\})$

and $x \in \text{elts } (\text{ordermap } A \ r \ y)$ **and** $y \in A \ (y, u) \in r$ **for** $x \ y$

proof –

have $\text{ordermap } A \ r \ y = \text{ZFC-in-HOL.set } (\text{ordermap } A \ r \ ` \ \{a \in A. (a, y) \in r\})$

using *ordermap assms(1)* **by** *force*

then have $x \in \text{ordermap } A \ r \ ` \ \{z \in A. (z, y) \in r\}$

by (*metis (no-types, lifting) elts-of-set empty-iff x*)

then have $\exists v. v \in A \wedge (v, u) \in r \wedge x = \text{ordermap } A \ r \ v$

using *that transD [OF <trans r>]* **by** *blast*

then show *?thesis*

by *blast*

qed

qed

moreover have $\text{Ord } x$

if $x \in \text{elts } (\text{set } (\text{ordermap } A \ r \ ` \ \{y \in A. (y, u) \in r\}))$ **for** x

using *that less* by (auto simp: split: if-split-asm)
 ultimately show ?case
 by (metis (full-types) Ord-def ordermap assms(1))
 qed

lemma *wf-Ord-ordertype*: assumes *wf r trans r* shows $\text{Ord}(\text{ordertype } A \ r)$

proof –

have $y \leq \text{set } (\text{ordermap } A \ r \ ' \ A)$
 if $y = \text{ordermap } A \ r \ x \ x \in A$ small ($\text{ordermap } A \ r \ ' \ A$) for $x \ y$
 using *that* by (auto simp: less-eq-V-def ordermap [OF $\langle \text{wf } r \rangle$, of $A \ x$])
 moreover have $z \leq y$ if $y \in \text{ordermap } A \ r \ ' \ A \ z \in \text{elts } y$ for $y \ z$
 by (metis *wf-Ord-ordermap OrdmemD assms imageE order.strict-implies-order*
that)
 ultimately show ?thesis
 unfolding *ordertype-def Ord-def Transset-def* by simp
 qed

lemma *Ord-ordertype [simp]*: $\text{Ord}(\text{ordertype } A \ \text{VWF})$
 using *wf-Ord-ordertype* by blast

lemma *Ord-ordermap [simp]*: $\text{Ord } (\text{ordermap } A \ \text{VWF } x)$
 by blast

lemma *ordertype-singleton [simp]*:

assumes *wf r*
 shows *ordertype* $\{x\} \ r = 1$
proof –
 have $\dagger: \{y. y = x \wedge (y, x) \in r\} = \{\}$
 using *assms* by auto
 show ?thesis
 by (auto simp add: *ordertype-def assms* \dagger *ordermap [where a=x]*)
 qed

2.8.1 *ordermap* preserves the orderings in both directions

lemma *ordermap-mono*:

assumes $wx: (w, x) \in r$ and *wf r* $w \in A$ small A
 shows *ordermap* $A \ r \ w \in \text{elts } (\text{ordermap } A \ r \ x)$
proof –
 have small $\{a \in A. (a, x) \in r\} \wedge w \in A \wedge (w, x) \in r$
 by (*simp add: assms*)
 then show ?thesis
 using *assms ordermap [of r A]*
 by (*metis (no-types, lifting) elts-of-set image-eqI mem-Collect-eq replacement*)
 qed

lemma *converse-ordermap-mono*:

assumes *ordermap* $A \ r \ y \in \text{elts } (\text{ordermap } A \ r \ x)$ *wf r total-on* $A \ r \ x \in A \ y \in A$
 small A

```

shows  $(y, x) \in r$ 
proof (cases  $x = y$ )
  case True
    then show ?thesis
      using assms(1) mem-not-refl by blast
  next
    case False
      then consider  $(x,y) \in r \mid (y,x) \in r$ 
        using  $\langle \text{total-on } A \ r \rangle$  assms by (meson UNIV-I total-on-def)
      then show ?thesis
        by (meson ordermap-mono assms mem-not-sym)
qed

```

```

lemma converse-ordermap-mono-iff:
  assumes wf r total-on A r x ∈ A y ∈ A small A
  shows  $\text{ordermap } A \ r \ y \in \text{elts } (\text{ordermap } A \ r \ x) \longleftrightarrow (y, x) \in r$ 
  by (metis assms converse-ordermap-mono ordermap-mono)

```

```

lemma ordermap-surj:  $\text{elts } (\text{ordertype } A \ r) \subseteq \text{ordermap } A \ r \text{ ' } A$ 
  unfolding ordertype-def by simp

```

```

lemma ordermap-bij:
  assumes wf r total-on A r small A
  shows bij-betw (ordermap A r) A (elts (ordertype A r))
  unfolding bij-betw-def
  proof (intro conjI)
    show inj-on (ordermap A r) A
      unfolding inj-on-def by (metis assms mem-not-refl ordermap-mono total-on-def)
    show  $\text{ordermap } A \ r \text{ ' } A = \text{elts } (\text{ordertype } A \ r)$ 
      by (metis ordertype-def <small A> elts-of-set replacement)
qed

```

```

lemma ordermap-eq-iff [simp]:
   $\llbracket x \in A; y \in A; wf \ r; \text{total-on } A \ r; \text{small } A \rrbracket \implies \text{ordermap } A \ r \ x = \text{ordermap } A \ r \ y \longleftrightarrow x = y$ 
  by (metis bij-betw-iff-bijections ordermap-bij)

```

```

lemma inv-into-ordermap:  $\alpha \in \text{elts } (\text{ordertype } A \ r) \implies \text{inv-into } A \ (\text{ordermap } A \ r) \ \alpha \in A$ 
  by (meson in-mono inv-into-into ordermap-surj)

```

```

lemma ordertype-nat-imp-finite:
  assumes  $\text{ordertype } A \ r = \text{ord-of-nat } m \text{ small } A \ wf \ r \ \text{total-on } A \ r$ 
  shows finite A
proof –
  have  $A \approx \text{elts } m$ 
    using eqpoll-def assms ordermap-bij by fastforce
  then show ?thesis
    using eqpoll-finite-iff finite-Ord-omega by blast

```

qed

lemma *wf-ordertype-epoll*:
 assumes *wf r total-on A r small A*
 shows *elts (ordertype A r) \approx A*
 using *assms eqpoll-def eqpoll-sym ordermap-bij* by blast

lemma *ordertype-epoll*:
 assumes *small A*
 shows *elts (ordertype A VWF) \approx A*
 using *assms wf-ordertype-epoll total-VWF wf-VWF*
 by (*simp add: wf-ordertype-epoll total-on-def*)

2.9 More advanced ordertype and ordermap results

lemma *ordermap-VWF-0 [simp]*: *ordermap A VWF 0 = 0*
 by (*simp add: ordermap wf-VWO VWF-def*)

lemma *ordertype-empty [simp]*: *ordertype {} r = 0*
 by (*simp add: ordertype-def*)

lemma *ordertype-eq-0-iff [simp]*: $\llbracket \text{small } X; \text{ wf } r \rrbracket \implies \text{ordertype } X \text{ } r = 0 \iff X = \{\}$
 by (*metis ordertype-def elts-of-set replacement image-is-empty zero-V-def*)

lemma *ordermap-mono-less*:
 assumes $(w, x) \in r$
 and *wf r trans r*
 and $w \in A \ x \in A$
 and *small A*
 shows *ordermap A r w < ordermap A r x*
 by (*simp add: OrdmemD assms ordermap-mono*)

lemma *ordermap-mono-le*:
 assumes $(w, x) \in r \vee w=x$
 and *wf r trans r*
 and $w \in A \ x \in A$
 and *small A*
 shows *ordermap A r w \leq ordermap A r x*
 by (*metis assms dual-order.strict-implies-order eq-refl ordermap-mono-less*)

lemma *converse-ordermap-le-mono*:
 assumes *ordermap A r y \leq ordermap A r x wf r total r x \in A small A*
 shows $(y, x) \in r \vee y=x$
 by (*meson UNIV-I assms mem-not-refl ordermap-mono total-on-def vsubsetD*)

lemma *ordertype-mono*:
 assumes $X \subseteq Y$ and *r: wf r trans r* and *small Y*
 shows *ordertype X r \leq ordertype Y r*

proof –
have *small X*
using *assms smaller-than-small by fastforce*
have *: *ordermap X r x ≤ ordermap Y r x for x*
using *⟨wf r⟩*
proof (*induction x rule: wf-induct-rule*)
case (*less x*)
have *ordermap X r z < ordermap Y r x if z ∈ X and zx: (z,x) ∈ r for z*
using *less [OF zx] assms*
by (*meson Ord-linear2 OrdmemD wf-Ord-ordermap ordermap-mono in-mono leD that(1) vsubsetD zx*)
then show ?*case*
by (*auto simp add: ordermap [of - X x] ⟨small X⟩ Ord-mem-iff-lt set-image-le-iff less-eq-V-def r*)
qed
show ?*thesis*
proof –
have *ordermap Y r ‘ Y = elts (ordertype Y r)*
by (*metis ordertype-def ⟨small Y⟩ elts-of-set replacement*)
then have *ordertype Y r ⊈ ordermap X r ‘ X*
using * *⟨X ⊆ Y⟩ by fastforce*
then show ?*thesis*
by (*metis Ord-linear2 Ord-mem-iff-lt ordertype-def wf-Ord-ordertype ⟨small X⟩ elts-of-set replacement r*)
qed
qed

corollary *ordertype-VWF-mono:*
assumes *X ⊆ Y small Y*
shows *ordertype X VWF ≤ ordertype Y VWF*
using *assms by (simp add: ordertype-mono)*

lemma *ordertype-UNION-ge:*
assumes *A ∈ A wf r trans r A ⊆ Collect small small A*
shows *ordertype A r ≤ ordertype (⋃A) r*
by (*rule ordertype-mono*) (*use assms in auto*)

lemma *inv-ordermap-mono-less:*
assumes (*inv-into M (ordermap M r) α, inv-into M (ordermap M r) β*) ∈ *r*
and *small M and α: α ∈ elts (ordertype M r) and β: β ∈ elts (ordertype M r)*
and *wf r trans r*
shows *α < β*

proof –
have *α = ordermap M r (inv-into M (ordermap M r) α)*
by (*metis α f-inv-into-f ordermap-surj subset-eq*)
also have *... < ordermap M r (inv-into M (ordermap M r) β)*
by (*meson α β assms in-mono inv-into-into ordermap-mono-less ordermap-surj*)
also have *... = β*
by (*meson β f-inv-into-f in-mono ordermap-surj*)

finally show *?thesis* .
qed

lemma *inv-ordermap-mono-eq*:

assumes *inv-into* M (*ordermap* M r) $\alpha = \text{inv-into } M$ (*ordermap* M r) β
and $\alpha \in \text{elts}$ (*ordertype* M r) $\beta \in \text{elts}$ (*ordertype* M r)
shows $\alpha = \beta$
by (*metis* *assms* *f-inv-into-f* *ordermap-surj* *subsetD*)

lemma *inv-ordermap-VWF-mono-le*:

assumes *inv-into* M (*ordermap* M *VWF*) $\alpha \leq \text{inv-into } M$ (*ordermap* M *VWF*) β
and $M \subseteq \text{ON}$ *small* M **and** $\alpha: \alpha \in \text{elts}$ (*ordertype* M *VWF*) **and** $\beta: \beta \in \text{elts}$ (*ordertype* M *VWF*)
shows $\alpha \leq \beta$

proof –

have $\alpha = \text{ordermap } M$ *VWF* (*inv-into* M (*ordermap* M *VWF*) α)
by (*metis* α *f-inv-into-f* *ordermap-surj* *subset-eq*)
also have $\dots \leq \text{ordermap } M$ *VWF* (*inv-into* M (*ordermap* M *VWF*) β)
by (*metis* *ON-imp-Ord* *VWF-iff-Ord-less* *assms* *dual-order.strict-implies-order* *elts-of-set* *eq-refl* *inv-into-into* *order.not-eq-order-implies-strict* *ordermap-mono-less* *ordertype-def* *replacement* *trans-VWF* *wf-VWF*)
also have $\dots = \beta$
by (*meson* β *f-inv-into-f* *in-mono* *ordermap-surj*)
finally show *?thesis* .

qed

lemma *inv-ordermap-VWF-mono-iff*:

assumes $M \subseteq \text{ON}$ *small* M **and** $\alpha \in \text{elts}$ (*ordertype* M *VWF*) **and** $\beta \in \text{elts}$ (*ordertype* M *VWF*)
shows *inv-into* M (*ordermap* M *VWF*) $\alpha \leq \text{inv-into } M$ (*ordermap* M *VWF*) β
 $\longleftrightarrow \alpha \leq \beta$
by (*metis* *ON-imp-Ord* *Ord-linear-le* *assms* *dual-order.eq-iff* *inv-into-ordermap* *inv-ordermap-VWF-mono-le*)

lemma *inv-ordermap-VWF-strict-mono-iff*:

assumes $M \subseteq \text{ON}$ *small* M **and** $\alpha \in \text{elts}$ (*ordertype* M *VWF*) **and** $\beta \in \text{elts}$ (*ordertype* M *VWF*)
shows *inv-into* M (*ordermap* M *VWF*) $\alpha < \text{inv-into } M$ (*ordermap* M *VWF*) β
 $\longleftrightarrow \alpha < \beta$
by (*simp* *add*: *assms* *inv-ordermap-VWF-mono-iff* *less-le-not-le*)

lemma *strict-mono-on-ordertype*:

assumes $M \subseteq \text{ON}$ *small* M
obtains f **where** $f \in \text{elts}$ (*ordertype* M *VWF*) $\rightarrow M$ *strict-mono-on* (*elts* (*ordertype* M *VWF*)) f

proof

show *inv-into* M (*ordermap* M *VWF*) $\in \text{elts}$ (*ordertype* M *VWF*) $\rightarrow M$
by (*meson* *Pi-I'* *in-mono* *inv-into-into* *ordermap-surj*)

show *strict-mono-on* (*elts* (*ordertype* *M* *VWF*)) (*inv-into* *M* (*ordermap* *M* *VWF*))
proof (*clarsimp simp: strict-mono-on-def*)
fix *x y*
assume $x \in \text{elts } (\text{ordertype } M \text{ VWF})$ $y \in \text{elts } (\text{ordertype } M \text{ VWF})$ $x < y$
then show *inv-into* *M* (*ordermap* *M* *VWF*) $x < \text{inv-into } M$ (*ordermap* *M* *VWF*) *y*
using *assms* **by** (*meson ON-imp-Ord Ord-linear2 inv-into-into inv-ordermap-VWF-mono-le leD ordermap-surj subsetD*)
qed
qed

lemma *ordermap-inc-eq*:
assumes $x \in A$ *small A*
and $\pi: \bigwedge x y. \llbracket x \in A; y \in A; (x, y) \in r \rrbracket \implies (\pi x, \pi y) \in s$
and $r: \text{wf } r$ *total-on* *A* *r* **and** $\text{wf } s$
shows $\text{ordermap } (\pi \text{ ' } A) s (\pi x) = \text{ordermap } A r x$
using $\langle \text{wf } r \rangle \langle x \in A \rangle$
proof (*induction x rule: wf-induct-rule*)
case (*less x*)
then have $1: \{y \in A. (y, x) \in r\} = A \cap \{y. (y, x) \in r\}$
using *r* **by** *auto*
have $2: \{y \in \pi \text{ ' } A. (y, \pi x) \in s\} = \pi \text{ ' } A \cap \{y. (y, \pi x) \in s\}$
by *auto*
have $\text{inv}\pi: \bigwedge x y. \llbracket x \in A; y \in A; (\pi x, \pi y) \in s \rrbracket \implies (x, y) \in r$
by (*metis* $\pi \langle \text{wf } s \rangle \langle \text{total-on } A r \rangle \text{total-on-def wf-not-sym}$)
have $\text{eq}: f \text{ ' } (\pi \text{ ' } A \cap \{y. (y, \pi x) \in s\}) = (f \circ \pi) \text{ ' } (A \cap \{y. (y, x) \in r\})$ **for** *f*
 $:: 'b \Rightarrow V$
using *less* **by** (*auto simp: image-subset-iff inv\pi \pi*)
show *?case*
using *less*
by (*simp add: ordermap [OF $\langle \text{wf } r \rangle$, of - x] ordermap [OF $\langle \text{wf } s \rangle$, of - πx] 1 2*)
eq)
qed

lemma *ordertype-inc-eq*:
assumes *small A*
and $\pi: \bigwedge x y. \llbracket x \in A; y \in A; (x, y) \in r \rrbracket \implies (\pi x, \pi y) \in s$
and $r: \text{wf } r$ *total-on* *A* *r* **and** $\text{wf } s$
shows $\text{ordertype } (\pi \text{ ' } A) s = \text{ordertype } A r$
proof –
have $\text{ordermap } (\pi \text{ ' } A) s (\pi x) = \text{ordermap } A r x$ **if** $x \in A$ **for** *x*
using *assms* **that** **by** (*auto simp: ordermap-inc-eq*)
then show *?thesis*
unfolding *ordertype-def*
by (*metis (no-types, lifting) image-cong image-image*)
qed

lemma *ordertype-inc-le*:
assumes *small A* *small B*

and $\pi: \bigwedge x y. \llbracket x \in A; y \in A; (x, y) \in r \rrbracket \implies (\pi x, \pi y) \in s$
and $r: \text{wf } r \text{ total-on } A \text{ r and wf } s \text{ trans } s$
and $\pi ' A \subseteq B$
shows $\text{ordertype } A \text{ r} \leq \text{ordertype } B \text{ s}$
by (*metis assms ordertype-inc-eq ordertype-mono*)

corollary *ordertype-VWF-inc-eq:*

assumes $A \subseteq ON \pi ' A \subseteq ON \text{ small } A$ **and** $\bigwedge x y. \llbracket x \in A; y \in A; x < y \rrbracket \implies \pi x < \pi y$
shows $\text{ordertype } (\pi ' A) \text{ VWF} = \text{ordertype } A \text{ VWF}$
proof (*rule ordertype-inc-eq*)
show $(\pi x, \pi y) \in \text{VWF}$
if $x \in A \ y \in A \ (x, y) \in \text{VWF}$ **for** $x \ y$
using *that ON-imp-Ord assms by auto*
show $\text{total-on } A \text{ VWF}$
by (*meson UNIV-I total-VWF total-on-def*)
qed (*use assms in auto*)

lemma *ordertype-image-ordermap:*

assumes $\text{small } A \ X \subseteq A \ \text{wf } r \ \text{trans } r \ \text{total-on } X \ r$
shows $\text{ordertype } (\text{ordermap } A \ r ' X) \text{ VWF} = \text{ordertype } X \ r$
proof (*rule ordertype-inc-eq*)
show $\text{small } X$
by (*meson assms smaller-than-small*)
show $(\text{ordermap } A \ r \ x, \text{ordermap } A \ r \ y) \in \text{VWF}$
if $x \in X \ y \in X \ (x, y) \in r$ **for** $x \ y$
by (*meson that wf-Ord-ordermap VWF-iff-Ord-less assms ordermap-mono-less subsetD*)
qed (*use assms in auto*)

lemma *ordertype-map-image:*

assumes $B \subseteq A \ \text{small } A$
shows $\text{ordertype } (\text{ordermap } A \ \text{VWF} ' A - \text{ordermap } A \ \text{VWF} ' B) \text{ VWF} = \text{ordertype } (A - B) \ \text{VWF}$
proof –
have $\text{ordermap } A \ \text{VWF} ' A - \text{ordermap } A \ \text{VWF} ' B = \text{ordermap } A \ \text{VWF} ' (A - B)$
using *assms by auto*
then have $\text{ordertype } (\text{ordermap } A \ \text{VWF} ' A - \text{ordermap } A \ \text{VWF} ' B) \text{ VWF} = \text{ordertype } (\text{ordermap } A \ \text{VWF} ' (A - B)) \text{ VWF}$
by *simp*
also have $\dots = \text{ordertype } (A - B) \ \text{VWF}$
using $\langle \text{small } A \rangle$ *ordertype-image-ordermap by fastforce*
finally show *?thesis .*
qed

proposition *ordertype-le-ordertype:*

assumes $r: \text{wf } r \ \text{total-on } A \ r$ **and** $\text{small } A$
assumes $s: \text{wf } s \ \text{total-on } B \ s \ \text{trans } s$ **and** $\text{small } B$

shows $\text{ordertype } A \ r \leq \text{ordertype } B \ s \iff$
 $(\exists f \in A \rightarrow B. \text{inj-on } f \ A \wedge (\forall x \in A. \forall y \in A. ((x,y) \in r \implies (f \ x, f \ y) \in s)))$
(is $?lhs = ?rhs$)

proof
assume $L: ?lhs$
define f **where** $f \equiv \text{inv-into } B \ (\text{ordermap } B \ s) \circ \text{ordermap } A \ r$
show $?rhs$
proof (*intro* bexI *conjI* *ballI* *impI*)
have $AB: \text{elts } (\text{ordertype } A \ r) \subseteq \text{ordermap } B \ s \ ' B$
by (*metis* L *assms*(γ) *ordertype-def* *replacement* *set-of-elts* *small-elts* *subset-iff-less-eq-V*)
have $\text{bij}A: \text{bij-betw } (\text{ordermap } A \ r) \ A \ (\text{elts } (\text{ordertype } A \ r))$
using *ordermap-bij* $\langle \text{small } A \rangle \ r$ **by** *blast*
have $\text{inv-into } B \ (\text{ordermap } B \ s) \ (\text{ordermap } A \ r \ i) \in B$ **if** $i \in A$ **for** i
by (*meson* L $\langle \text{small } A \rangle$ *inv-into-into* *ordermap-in-ordertype* *ordermap-surj* *subsetD* *that* *vsubsetD*)
then show $f \in A \rightarrow B$
by (*auto* *simp*: *Pi-iff* *f-def*)
show $\text{inj-on } f \ A$
proof (*clarsimp* *simp* *add*: *f-def* *inj-on-def*)
fix $x \ y$
assume $x \in A \ y \in A$
and $\text{inv-into } B \ (\text{ordermap } B \ s) \ (\text{ordermap } A \ r \ x) = \text{inv-into } B \ (\text{ordermap } B \ s) \ (\text{ordermap } A \ r \ y)$
then have $\text{ordermap } A \ r \ x = \text{ordermap } A \ r \ y$
by (*meson* AB $\langle \text{small } A \rangle$ *inv-into-injective* *ordermap-in-ordertype* *subsetD*)
then show $x = y$
by (*metis* $\langle x \in A \rangle \langle y \in A \rangle$ *bijA* *bij-betw-inv-into-left*)
qed
next
fix $x \ y$
assume $x \in A \ y \in A$ **and** $(x, y) \in r$
have $\ddagger: \text{ordermap } A \ r \ y \in \text{ordermap } B \ s \ ' B$
by (*meson* L $\langle y \in A \rangle \langle \text{small } A \rangle$ *in-mono* *ordermap-in-ordertype* *ordermap-surj* *vsubsetD*)
moreover have $\dagger: \bigwedge x. \text{inv-into } B \ (\text{ordermap } B \ s) \ (\text{ordermap } A \ r \ x) = f \ x$
by (*simp* *add*: *f-def*)
then have $*$: $\text{ordermap } B \ s \ (f \ y) = \text{ordermap } A \ r \ y$
using \ddagger **by** (*metis* *f-inv-into-f*)
moreover have $\text{ordermap } A \ r \ x \in \text{ordermap } B \ s \ ' B$
by (*meson* L $\langle x \in A \rangle \langle \text{small } A \rangle$ *in-mono* *ordermap-in-ordertype* *ordermap-surj* *vsubsetD*)
moreover have $\text{ordermap } A \ r \ x < \text{ordermap } A \ r \ y$
using $*$ $r \ s$ **by** (*metis* (*no-types*) *wf-Ord-ordermap* *OrdmemD* $\langle (x, y) \in r \rangle \langle x \in A \rangle \langle \text{small } A \rangle$ *ordermap-mono*)
ultimately show $(f \ x, f \ y) \in s$
using $\dagger \ s$ **by** (*metis* *assms*(γ) *f-inv-into-f* *inv-into-into* *less-asm* *ordermap-mono-less* *total-on-def*)

qed
next
assume $R: ?rhs$
then obtain f **where** $f: f \in A \rightarrow B$ *inj-on* $f A \forall x \in A. \forall y \in A. (x, y) \in r \longrightarrow (f x, f y) \in s$
by *blast*
show $?lhs$
by (*rule ordertype-inc-le [where $\pi=f$]*) (*use f assms in auto*)
qed

lemma *iso-imp-ordertype-eq-ordertype*:
assumes *iso: iso r r' f*
and *wf r*
and *Total r*
and *sm: small (Field r)*
shows *ordertype (Field r) r = ordertype (Field r') r'*
by (*metis (no-types, lifting) iso-forward iso-wf assms iso-Field ordertype-inc-eq sm*)

lemma *ordertype-infinite-ge- ω* :
assumes *infinite A small A*
shows *ordertype A VWF $\geq \omega$*
proof –
have *inj-on (ordermap A VWF) A*
by (*meson ordermap-bij <small A> bij-betw-def total-on-VWF wf-VWF*)
then have *infinite (ordermap A VWF ' A)*
using *<infinite A> finite-image-iff* **by** *blast*
then show *?thesis*
using *Ord-ordertype <small A> infinite-Ord-omega* **by** (*auto simp: ordertype-def*)
qed

lemma *ordertype-eqI*:
assumes *wf r total-on A r small A wf s*
bij-betw f A B ($\forall x \in A. \forall y \in A. (f x, f y) \in s \longleftrightarrow (x, y) \in r$)
shows *ordertype A r = ordertype B s*
by (*metis assms bij-betw-imp-surj-on ordertype-inc-eq*)

lemma *ordermap-eq-self*:
assumes *Ord α and $x: x \in \text{elts } \alpha$*
shows *ordermap (elts α) VWF $x = x$*
using *Ord-in-Ord [OF assms] x*
proof (*induction x rule: Ord-induct*)
case (*step x*)
have $1: \{y \in \text{elts } \alpha. (y, x) \in \text{VWF}\} = \text{elts } x$ (**is** $?A = -$)
proof
show $?A \subseteq \text{elts } x$
using *<Ord α >* **by** *clarify (meson Ord-in-Ord Ord-mem-iff-lt VWF-iff-Ord-less step.hyps)*
show $\text{elts } x \subseteq ?A$

using $\langle \text{Ord } \alpha \rangle$ **by** *clarify (meson Ord-in-Ord Ord-trans OrdmemD VWF-iff-Ord-less step.prem)*
qed
show *?case*
using *step*
by (*simp add: ordermap [OF wf-VWF, of - x] 1 Ord-trans [of - - α] step.prem*
 $\langle \text{Ord } \alpha \rangle$ *cong: image-cong*)
qed

lemma *ordertype-eq-Ord [simp]:*
assumes *Ord α*
shows *ordertype (elts α) VWF = α*
using *assms ordermap-eq-self [OF assms]* **by** (*simp add: ordertype-def*)

proposition *ordertype-eq-iff:*
assumes α : *Ord α and r : wf r and small A total-on A r trans r*
shows *ordertype A r = $\alpha \iff$*
 $(\exists f. \text{bij-betw } f \ A \ (\text{elts } \alpha) \wedge (\forall x \in A. \forall y \in A. f \ x < f \ y \iff (x, y) \in r))$
(is ?lhs = ?rhs)
proof *safe*
assume *eq: α = ordertype A r*
show $\exists f. \text{bij-betw } f \ A \ (\text{elts } (\text{ordertype } A \ r)) \wedge (\forall x \in A. \forall y \in A. f \ x < f \ y \iff ((x, y) \in r))$
proof (*intro exI conjI ballI*)
show *bij-betw (ordermap A r) A (elts (ordertype A r))*
by (*simp add: assms ordermap-bij*)
then show *ordermap A r x < ordermap A r $y \iff (x, y) \in r$*
if $x \in A$ $y \in A$ **for** x y
using *that assms*
by (*metis order.asym ordermap-mono-less total-on-def*)
qed
next
fix f
assume f : *bij-betw f A (elts α) $\forall x \in A. \forall y \in A. f \ x < f \ y \iff (x, y) \in r$*
have *ordertype A r = ordertype (elts α) VWF*
proof (*rule ordertype-eqI*)
show $\forall x \in A. \forall y \in A. ((f \ x, f \ y) \in \text{VWF}) = ((x, y) \in r)$
by (*meson Ord-in-Ord VWF-iff-Ord-less α bij-betwE f*)
qed (*use assms f in auto*)
then show *?lhs*
by (*simp add: α*)
qed

corollary *ordertype-VWF-eq-iff:*
assumes *Ord α small A*
shows *ordertype A VWF = $\alpha \iff$*
 $(\exists f. \text{bij-betw } f \ A \ (\text{elts } \alpha) \wedge (\forall x \in A. \forall y \in A. f \ x < f \ y \iff (x, y) \in \text{VWF}))$
by (*metis UNIV-I assms ordertype-eq-iff total-VWF total-on-def trans-VWF wf-VWF*)

lemma *ordertype-le-Ord*:

assumes $Ord\ \alpha\ X \subseteq elts\ \alpha$

shows $ordertype\ X\ VWF \leq \alpha$

by (*metis* *assms* *ordertype-VWF-mono* *ordertype-eq-Ord* *small-elts*)

lemma *ordertype-inc-le-Ord*:

assumes *small* $A\ Ord\ \alpha$

and $\pi: \bigwedge x\ y. \llbracket x \in A; y \in A; (x, y) \in r \rrbracket \implies \pi\ x < \pi\ y$

and *wf* r *total-on* $A\ r$

and *sub*: $\pi\ ` A \subseteq elts\ \alpha$

shows $ordertype\ A\ r \leq \alpha$

proof –

have $\bigwedge x\ y. \llbracket x \in A; y \in A; (x, y) \in r \rrbracket \implies (\pi\ x, \pi\ y) \in VWF$

by (*meson* *Ord-in-Ord* *VWF-iff-Ord-less* $\pi\ ` Ord\ \alpha$ *sub* *image-subset-iff*)

with *assms* **show** *?thesis*

by (*metis* *ordertype-inc-eq* *ordertype-le-Ord* *wf-VWF*)

qed

lemma *le-ordertype-obtains-subset*:

assumes $\alpha: \beta \leq \alpha$ *ordertype* $H\ VWF = \alpha$ **and** *small* $H\ Ord\ \beta$

obtains G **where** $G \subseteq H$ *ordertype* $G\ VWF = \beta$

proof (*intro* *exI* *conjI* *that*)

let $?f = ordermap\ H\ VWF$

show $\ddagger: inv\text{-}into\ H\ ?f\ ` elts\ \beta \subseteq H$

unfolding *image-subset-iff*

by (*metis* α *inv-into-into* *ordermap-surj* *subsetD* *vsubsetD*)

have $\exists f. bij\text{-}betw\ f\ (inv\text{-}into\ H\ ?f\ ` elts\ \beta)\ (elts\ \beta) \wedge (\forall x \in inv\text{-}into\ H\ ?f\ ` elts\ \beta. \forall y \in inv\text{-}into\ H\ ?f\ ` elts\ \beta. (f\ x < f\ y) = ((x, y) \in VWF))$

proof (*intro* *exI* *conjI* *ballI* *iffI*)

show *bij-betw* $?f\ (inv\text{-}into\ H\ ?f\ ` elts\ \beta)\ (elts\ \beta)$

using *ordermap-bij* [*OF* *wf-VWF* *total-on-VWF* $\langle small\ H \rangle\ \alpha$]

by (*metis* *bij-betw-inv-into-RIGHT* *bij-betw-subset* *less-eq-V-def* \ddagger)

next

fix $x\ y$

assume $x: x \in inv\text{-}into\ H\ ?f\ ` elts\ \beta$

and $y: y \in inv\text{-}into\ H\ ?f\ ` elts\ \beta$

show $?f\ x < ?f\ y$ **if** $(x, y) \in VWF$

using *that* $\ddagger\ \langle small\ H \rangle$ *in-mono* *ordermap-mono-less* $x\ y$ **by** *fastforce*

show $(x, y) \in VWF$ **if** $?f\ x < ?f\ y$

using *that* $\ddagger\ \langle small\ H \rangle$ *in-mono* *ordermap-mono-less* [*OF* - *wf-VWF* *trans-VWF*]

$x\ y$

by (*metis* *UNIV-I* *less-imp-not-less* *total-VWF* *total-on-def*)

qed

then **show** *ordertype* $(inv\text{-}into\ H\ ?f\ ` elts\ \beta)\ VWF = \beta$

by (*subst* *ordertype-eq-iff*) (*use* *assms* **in** *auto*)

qed

lemma *ordertype-infinite- ω* :
assumes $A \subseteq \text{elts } \omega$ *infinite* A
shows *ordertype* A $VWF = \omega$
proof (*rule antisym*)
show *ordertype* A $VWF \leq \omega$
by (*simp add: assms ordertype-le-Ord*)
show $\omega \leq \text{ordertype } A$ VWF
using *assms down ordertype-infinite-ge- ω* **by** *auto*
qed

For infinite sets of natural numbers

lemma *ordertype-nat- ω* :
assumes *infinite* N **shows** *ordertype* N *less-than* $= \omega$
proof –
have *small* N
by (*meson inj-on-def ord-of-nat-inject small-def small-iff-range small-image-nat-V*)
have *ordertype* (*ord-of-nat* ‘ N) $VWF = \omega$
by (*force simp: assms finite-image-iff inj-on-def intro: ordertype-infinite- ω*)
moreover **have** *ordertype* (*ord-of-nat* ‘ N) $VWF = \text{ordertype } N$ *less-than*
by (*auto intro: ordertype-inc-eq ‹small N›*)
ultimately show *?thesis*
by *simp*
qed

proposition *ordertype-eq-ordertype*:
assumes r : *wf* r *total-on* A r *trans* r **and** *small* A
assumes s : *wf* s *total-on* B s *trans* s **and** *small* B
shows *ordertype* A $r = \text{ordertype } B$ $s \iff$
 $(\exists f. \text{bij-betw } f$ A $B \wedge (\forall x \in A. \forall y \in A. (f$ x, f $y) \in s \iff (x, y) \in r))$
(is *?lhs* $=$ *?rhs**)**
proof
assume L : *?lhs*
define γ **where** $\gamma = \text{ordertype } A$ r
have A : *bij-betw* (*ordermap* A r) A (*ordermap* A r ‘ A)
by (*meson ordermap-bij assms(4) bij-betw-def r*)
have B : *bij-betw* (*ordermap* B s) B (*ordermap* B s ‘ B)
by (*meson ordermap-bij assms(8) bij-betw-def s*)
define f **where** $f \equiv \text{inv-into } B$ (*ordermap* B s) \circ *ordermap* A r
show *?rhs*
proof (*intro exI conjI*)
have *bijA*: *bij-betw* (*ordermap* A r) A (*elts* γ)
unfolding γ -*def* **using** *ordermap-bij ‹small A› r* **by** *blast*
moreover **have** *bijB*: *bij-betw* (*ordermap* B s) B (*elts* γ)
by (*simp add: L* γ -*def* *ordermap-bij ‹small B› s*)
ultimately show *bij*: *bij-betw* f A B
unfolding f -*def* **using** *bij-betw-comp-iff bij-betw-inv-into* **by** *blast*
have *invB*: $\bigwedge \alpha. \alpha \in \text{elts } \gamma \implies \text{ordermap } B$ s (*inv-into* B (*ordermap* B s) α)
 $= \alpha$
by (*meson bijB bij-betw-inv-into-right*)*


```

have ordermap-A- $\gamma$ :  $\bigwedge a. a \in A \implies \text{ordermap } A \ r \ a \in \text{elts } \gamma$ 
  using bijA bij-betwE by auto
have f-in-B:  $\bigwedge a. a \in A \implies f \ a \in B$ 
  using bij bij-betwE by fastforce
show  $\forall x \in A. \forall y \in A. (f \ x, f \ y) \in s \longleftrightarrow (x, y) \in r$ 
  proof (intro iffI ballI)
    fix x y
    assume  $x \in A \ y \in A$  and ins:  $(f \ x, f \ y) \in s$ 
    then have  $\text{ordermap } A \ r \ x < \text{ordermap } A \ r \ y$ 
      unfolding o-def
      by (metis (mono-tags, lifting) f-def <small B> comp-apply f-in-B invB
ordermap-A- $\gamma$  ordermap-mono-less s(1) s(3))
    then show  $(x, y) \in r$ 
      by (metis <x \in A> <y \in A> <small A> order.asym ordermap-mono-less r
total-on-def)
    next
    fix x y
    assume  $x \in A \ y \in A$  and  $(x, y) \in r$ 
    then have  $\text{ordermap } A \ r \ x < \text{ordermap } A \ r \ y$ 
      by (simp add: <small A> ordermap-mono-less r)
    then have  $(f \ y, f \ x) \notin s$ 
      by (metis (mono-tags, lifting) <x \in A> <y \in A> <small B> comp-apply f-def
f-in-B invB order.asym ordermap-A- $\gamma$  ordermap-mono-less s(1) s(3))
    moreover have  $f \ y \neq f \ x$ 
      by (metis <(x, y) \in r> <x \in A> <y \in A> bij bij-betw-inv-into-left r(1)
wf-not-sym)
    ultimately show  $(f \ x, f \ y) \in s$ 
      by (meson <x \in A> <y \in A> f-in-B s(2) total-on-def)
    qed
  qed
next
assume ?rhs
then show ?lhs
  using assms ordertype-eqI by blast
qed

corollary ordertype-eq-ordertype-iso:
  assumes r: wf r total-on A r trans r and small A and FA: Field r = A
  assumes s: wf s total-on B s trans s and small B and FB: Field s = B
  shows  $\text{ordertype } A \ r = \text{ordertype } B \ s \longleftrightarrow (\exists f. \text{iso } r \ s \ f)$ 
  (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  then obtain f where bij-betw f A B  $\forall x \in A. \forall y \in A. (f \ x, f \ y) \in s \longleftrightarrow (x, y) \in r$ 
  by (metis <small A> <small B> ordermap-mono-less r s)
  using assms ordertype-eq-ordertype by blast
  then show ?rhs
    using FA FB iso-iff2 by blast
next

```

assume $?rhs$
then show $?lhs$
using $FA\ FB\ \langle small\ A \rangle\ iso-imp-ordertype-eq-ordertype\ r$ **by** $blast$
qed

lemma *Limit-ordertype-imp-Field-Restr:*

assumes $Lim: Limit\ (ordertype\ A\ r)$ **and** $r: wf\ r\ total-on\ A\ r$ **and** $small\ A$
shows $Field\ (Restr\ r\ A) = A$

proof –

have $\exists y \in A. (x, y) \in r$ **if** $x \in A$ **for** x

proof –

let $?oy = succ\ (ordermap\ A\ r\ x)$

have $\S: ?oy \in elts\ (ordertype\ A\ r)$

by $(simp\ add: Lim\ \langle small\ A \rangle\ ordermap-in-ordertype\ succ-in-Limit-iff\ that)$

then have $A: inv-into\ A\ (ordermap\ A\ r)\ ?oy \in A$

by $(simp\ add: inv-into-ordermap)$

moreover have $(x, inv-into\ A\ (ordermap\ A\ r)\ ?oy) \in r$

proof –

have $ordermap\ A\ r\ x \in elts\ (ordermap\ A\ r\ (inv-into\ A\ (ordermap\ A\ r)\ ?oy))$

by $(metis\ \S\ elts-succ\ f-inv-into-f\ insert-iff\ ordermap-surj\ subsetD)$

then show $?thesis$

by $(metis\ \langle small\ A \rangle\ A\ converse-ordermap-mono\ r\ that)$

qed

ultimately show $?thesis\ ..$

qed

then have $A \subseteq Field\ (Restr\ r\ A)$

by $(auto\ simp: Field-def)$

then show $?thesis$

by $(simp\ add: Field-Restr-subset\ subset-antisym)$

qed

lemma *ordertype-Field-Restr:*

assumes $wf\ r\ total-on\ A\ r\ trans\ r\ small\ A$ $Field\ (Restr\ r\ A) = A$

shows $ordertype\ (Field\ (Restr\ r\ A))\ (Restr\ r\ A) = ordertype\ A\ r$

using $assms$ **by** $(force\ simp: ordertype-eq-ordertype\ wf-Int1\ total-on-def\ trans-Restr)$

proposition *ordertype-eq-ordertype-iso-Restr:*

assumes $r: wf\ r\ total-on\ A\ r\ trans\ r$ **and** $small\ A$ **and** $FA: Field\ (Restr\ r\ A) = A$

assumes $s: wf\ s\ total-on\ B\ s\ trans\ s$ **and** $small\ B$ **and** $FB: Field\ (Restr\ s\ B) = B$

shows $ordertype\ A\ r = ordertype\ B\ s \iff (\exists f. iso\ (Restr\ r\ A)\ (Restr\ s\ B)\ f)$

(is $?lhs = ?rhs)$

proof

assume $L: ?lhs$

then obtain f **where** $bij-betw\ f\ A\ B\ \forall x \in A. \forall y \in A. (f\ x, f\ y) \in s \iff (x, y) \in r$

using $assms\ ordertype-eq-ordertype$ **by** $blast$

then show $?rhs$

using *FA FB bij-betwE unfolding iso-iff2* by *fastforce*
 next
 assume *?rhs*
 moreover
 have *ordertype (Field (Restr r A)) (Restr r A) = ordertype A r*
 using *FA ‹small A› ordertype-Field-Restr r* by *blast*
 moreover
 have *ordertype (Field (Restr s B)) (Restr s B) = ordertype B s*
 using *FB ‹small B› ordertype-Field-Restr s* by *blast*
 ultimately show *?lhs*
 using *iso-imp-ordertype-eq-ordertype FA FB ‹small A› r*
 by (*fastforce intro: total-on-imp-Total-Restr trans-Restr wf-Int1*)
 qed

lemma *ordermap-insert:*

assumes *Ord α and y: Ord y y ≤ α and U: U ⊆ elts α*
 shows *ordermap (insert α U) VWF y = ordermap U VWF y*
 using *y*
 proof (*induction rule: Ord-induct*)
 case (*step y*)
 then have *1: {u ∈ U. (u, y) ∈ VWF} = elts y ∩ U*
 apply (*simp add: set-eq-iff*)
 by (*meson Ord-in-Ord Ord-mem-iff-lt VWF-iff-Ord-less assms subsetD*)
 have *2: {u ∈ insert α U. (u, y) ∈ VWF} = elts y ∩ U*
 apply (*simp add: set-eq-iff*)
 by (*meson Ord-in-Ord Ord-mem-iff-lt VWF-iff-Ord-less assms leD step.hyps*
step.prem subsetD)
 show *?case*
 using *step*
 apply (*simp only: ordermap [OF wf-VWF, of - y] 1 2*)
 by (*meson Int-lower1 Ord-is-Transset Sup.SUP-cong Transset-def assms(1)*
in-mono vsubsetD)
 qed

lemma *ordertype-insert:*

assumes *Ord α and U: U ⊆ elts α*
 shows *ordertype (insert α U) VWF = succ (ordertype U VWF)*
 proof –
 have $\dagger: \{y \in \text{insert } \alpha \ U. (y, \alpha) \in \text{VWF}\} = U \{y \in U. (y, \alpha) \in \text{VWF}\} = U$
 using *Ord-in-Ord OrdmemD assms* by *auto*
 have *eq: $\bigwedge x. x \in U \implies \text{ordermap (insert } \alpha \ U) \ \text{VWF } x = \text{ordermap } U \ \text{VWF } x$*
 by (*meson Ord-in-Ord Ord-is-Transset Transset-def U assms(1) in-mono ordermap-insert*)
 have *ordertype (insert α U) VWF =*
 ZFC-in-HOL.set (insert (ordermap U VWF α) (ordermap U VWF ‘ U))
 by (*simp add: ordertype-def ordermap-insert assms eq*)
 also have $\dots = \text{succ (ZFC-in-HOL.set (ordermap U VWF ‘ U))}$
 using \dagger *U* by (*simp add: ordermap [OF wf-VWF, of - α] down succ-def vinsert-def*)

also have $\dots = \text{succ} (\text{ordertype } U \text{ VWF})$
by (*simp add: ordertype-def*)
finally show *?thesis* .
qed

lemma *finite-ordertype-le-card*:
assumes *finite A wf r trans r*
shows $\text{ordertype } A \ r \leq \text{ord-of-nat} (\text{card } A)$
proof –
have *Ord (ordertype A r)*
by (*simp add: wf-Ord-ordertype assms*)
moreover have $\text{ordermap } A \ r \text{ ' } A = \text{elts} (\text{ordertype } A \ r)$
by (*simp add: ordertype-def finite-imp-small <finite A>*)
moreover have $\text{card} (\text{ordermap } A \ r \text{ ' } A) \leq \text{card } A$
using *<finite A> card-image-le by blast*
ultimately show *?thesis*
by (*metis Ord-linear-le Ord-ord-of-nat <finite A> card-ord-of-nat card-seteq finite-imageI less-eq-V-def*)
qed

lemma *ordertype-VWF- ω* :
assumes *finite A*
shows $\text{ordertype } A \ \text{VWF} \in \text{elts } \omega$
proof –
have *finite (ordermap A VWF ' A)*
using *assms by blast*
then have $\text{ordertype } A \ \text{VWF} < \omega$
by (*meson Ord- ω OrdmemD trans-VWF wf-VWF assms finite-ordertype-le-card le-less-trans ord-of-nat- ω*)
then show *?thesis*
by (*simp add: Ord-mem-iff-lt*)
qed

lemma *ordertype-VWF-finite-nat*:
assumes *finite A*
shows $\text{ordertype } A \ \text{VWF} = \text{ord-of-nat} (\text{card } A)$
by (*metis finite-imp-small ordermap-bij total-on-VWF wf-VWF ω -def assms bij-betw-same-card card-ord-of-nat elts-of-set f-inv-into-f inf ordertype-VWF- ω*)

lemma *finite-ordertype-eq-card*:
assumes *small A wf r trans r total-on A r*
shows $\text{ordertype } A \ r = \text{ord-of-nat } m \iff \text{finite } A \wedge \text{card } A = m$
using *ordermap-bij [OF <wf r>]*
proof –
have **: bij-betw (ordermap A r) A (elts (ordertype A r))*
by (*simp add: assms ordermap-bij*)
moreover have $\text{card} (\text{ordermap } A \ r \text{ ' } A) = \text{card } A$
by (*meson bij-betw-def * card-image*)
ultimately show *?thesis*

using *assms bij-betw-finite bij-betw-imp-surj-on finite-Ord-omega ordertype-VWF-finite-nat wf-Ord-ordertype* **by** *fastforce*

qed

lemma *ex-bij-betw-strict-mono-card:*

assumes *finite M M ⊆ ON*

obtains *h* **where** *bij-betw h {.. \lt card M} M* **and** *strict-mono-on {.. \lt card M} h*

proof –

have *bij: bij-betw (ordermap M VWF) M (elts (card M))*

using *Finite-V ⟨finite M⟩ ordermap-bij ordertype-VWF-finite-nat* **by** *fastforce*

let *?h = (inv-into M (ordermap M VWF)) ∘ ord-of-nat*

show *thesis*

proof

show *bijh: bij-betw ?h {.. \lt card M} M*

proof (*rule bij-betw-trans*)

show *bij-betw ord-of-nat {.. \lt card M} (elts (card M))*

by (*simp add: bij-betw-def elts-ord-of-nat inj-on-def*)

show *bij-betw (inv-into M (ordermap M VWF)) (elts (card M)) M*

using *Finite-V assms bij-betw-inv-into ordermap-bij ordertype-VWF-finite-nat*

by *fastforce*

qed

show *strict-mono-on {.. \lt card M} ?h*

proof –

have *?h m < ?h n*

if *m < n n < card M* **for** *m n*

proof (*rule ccontr*)

obtain *mn: m ∈ elts (ordertype M VWF) n ∈ elts (ordertype M VWF)*

using *⟨m < n⟩ ⟨n < card M⟩ ⟨finite M⟩ ordertype-VWF-finite-nat* **by** *auto*

have *ord: Ord (?h m) Ord (?h n)*

using *bijh assms(2) bij-betwE* **that** **by** *fastforce+*

moreover

assume \neg *?h m < ?h n*

ultimately consider *?h m = ?h n | ?h m > ?h n*

using *Ord-linear-lt* **by** *blast*

then show *False*

proof *cases*

case *1*

then have *m = n*

by (*metis inv-ordermap-mono-eq mn comp-apply ord-of-nat-inject*)

with *⟨m < n⟩* **show** *False* **by** *blast*

next

case *2*

then have *ord-of-nat n ≤ ord-of-nat m*

by (*metis Finite-V mn assms comp-def inv-ordermap-VWF-mono-le*

less-imp-le)

then show *?thesis*

using *leD ⟨m < n⟩* **by** *blast*

qed

```

qed
with assms show ?thesis
  by (auto simp: strict-mono-on-def)
qed
qed
qed
qed

lemma ordertype-finite-less-than [simp]:
  assumes finite A shows ordertype A less-than = card A
proof -
  let ?M = ord-of-nat ' A
  obtain M: finite ?M ?M ⊆ ON
  using Ord-ord-of-nat assms by blast
  have ordertype A less-than = ordertype ?M VWF
  by (rule ordertype-inc-eq [symmetric]) (use assms finite-imp-small total-on-def)
in ⟨force+⟩
  also have ... = card A
  proof (subst ordertype-eq-iff)
    let ?M = ord-of-nat ' A
    obtain h where bijh: bij-betw h {..card A} ?M and smh: strict-mono-on
    {..card A} h
    by (metis M card-image ex-bij-betw-strict-mono-card inj-on-def ord-of-nat-inject)
    define f where f ≡ ord-of-nat ∘ inv-into {..card A} h
    show  $\exists f. \text{bij-betw } f \text{ } ?M \text{ (elts (card } A)) \wedge (\forall x \in ?M. \forall y \in ?M. f x < f y \longleftrightarrow ((x, y) \in VWF))$ 
    proof (intro exI conjI ballI)
      have bij-betw (ord-of-nat ∘ inv-into {..card A} h) (ord-of-nat ' A) (ord-of-nat
      '{..card A})
      by (meson UNIV-I bijh bij-betw-def bij-betw-inv-into bij-betw-subset bij-betw-trans
      inj-ord-of-nat subsetI)
      then show bij-betw f ?M (elts (card A))
      by (metis elts-ord-of-nat f-def)
    next
      fix x y
      assume xy: x ∈ ?M y ∈ ?M
      then obtain m n where x = ord-of-nat m y = ord-of-nat n
      by auto
      have  $(f x < f y) \longleftrightarrow ((h \circ \text{inv-into } \{..card A\} h) x < (h \circ \text{inv-into } \{..card$ 
       $A\} h) y)$ 
      unfolding f-def using smh bij-betw-imp-surj-on [OF bijh]
      apply simp
      by (metis (mono-tags, lifting) inv-into-into not-less-iff-gr-or-eq order.asym
      strict-mono-onD xy)
      also have ... = (x < y)
      using bijh
      by (simp add: bij-betw-inv-into-right xy)
      also have ... ⟷ ((x, y) ∈ VWF)
      using M(2) ON-imp-Ord xy by auto
      finally show  $(f x < f y) \longleftrightarrow ((x, y) \in VWF)$  .
  qed

```

qed
qed *auto*
finally show *?thesis* .
qed

2.10 Cardinality of an arbitrary HOL set

definition *gcard* :: 'a set \Rightarrow V
where *gcard* X \equiv if small X then (LEAST i. Ord i \wedge elts i \approx X) else 0

2.11 Cardinality of a set

definition *vcard* :: V \Rightarrow V
where *vcard* a \equiv (LEAST i. Ord i \wedge elts i \approx elts a)

lemma *gcard-eq-vcard* [*simp*]: *gcard* (elts x) = *vcard* x
by (*simp* add: *vcard-def* *gcard-def*)

definition *Card*:: V \Rightarrow bool
where *Card* i \equiv i = *vcard* i

abbreviation *CARD* **where** *CARD* \equiv Collect *Card*

lemma *cardinal-cong*: elts x \approx elts y \Longrightarrow *vcard* x = *vcard* y
unfolding *vcard-def* **by** (*meson* *eqpoll-sym* *eqpoll-trans*)

lemma *gcardinal-cong*:
assumes X \approx Y **shows** *gcard* X = *gcard* Y
proof –
have (LEAST i. Ord i \wedge elts i \approx X) = (LEAST i. Ord i \wedge elts i \approx Y)
by (*meson* *eqpoll-sym* *eqpoll-trans* *assms*)
then show *?thesis*
unfolding *gcard-def*
by (*meson* *eqpoll-sym* *small-econg* *assms*)
qed

lemma *vcard-set-image*: inj-on f (elts x) \Longrightarrow *vcard* (set (f ' elts x)) = *vcard* x
by (*simp* add: *cardinal-cong*)

lemma *gcard-image*: inj-on f X \Longrightarrow *gcard* (f ' X) = *gcard* X
by (*simp* add: *gcardinal-cong*)

lemma *Card-cardinal-eq*: Card κ \Longrightarrow *vcard* κ = κ
by (*simp* add: *Card-def*)

lemma *Card-is-Ord*:
assumes Card κ **shows** Ord κ
proof –
obtain α **where** Ord α elts α \approx elts κ
using Ord-ordertype ordertype-ecpoll **by** blast

then have Ord ($LEAST$ i . Ord $i \wedge elts$ $i \approx elts$ κ)
by ($metis$ Ord - $Least$)
then show $?thesis$
using $Card$ - def $vcard$ - def $assms$ **by** $auto$
qed

lemma $cardinal$ - $eqpoll$: $elts$ ($vcard$ a) \approx $elts$ a
unfolding $vcard$ - def
using $ordertype$ - $eqpoll$ [of $elts$ a] Ord - $LeastI$ **by** ($meson$ Ord - $ordertype$ $small$ - $elts$)

lemma inj - $into$ - $vcard$:
obtains f **where** $f \in elts$ $A \rightarrow elts$ ($vcard$ A) inj - on f ($elts$ A)
using $cardinal$ - $eqpoll$ [of A] inj - on - the - inv - $into$ the - inv - $into$ - $onto$
by ($fastforce$ $simp$: Pi - iff bij - $betw$ - def $eqpoll$ - def)

lemma $cardinal$ - $idem$ [$simp$]: $vcard$ ($vcard$ a) = $vcard$ a
using $cardinal$ - $cong$ $cardinal$ - $eqpoll$ **by** $blast$

lemma $subset$ - $smaller$ - $vcard$:
assumes $\kappa \leq vcard$ x $Card$ κ
obtains y **where** $y \leq x$ $vcard$ $y = \kappa$
proof –
obtain φ **where** φ : bij - $betw$ φ ($elts$ ($vcard$ x)) ($elts$ x)
using $cardinal$ - $eqpoll$ $eqpoll$ - def **by** $blast$
show $thesis$
proof
let $?y = ZFC$ - in - HOL . set (φ ‘ $elts$ κ)
show $?y \leq x$
by ($meson$ φ $assms$ bij - $betwE$ set - $image$ - le - iff $small$ - $elts$ $vsubsetD$)
show $vcard$ $?y = \kappa$
by ($metis$ $vcard$ - set - $image$ $Card$ - def $assms$ bij - $betw$ - def bij - $betw$ - $subset$ φ $less$ - eq - V - def)
qed
qed

every natural number is a (finite) cardinal

lemma nat - $into$ - $Card$:
assumes $\alpha \in elts$ ω **shows** $Card$ (α)
proof ($unfold$ $Card$ - def $vcard$ - def , $rule$ sym)
obtain n **where** n : $\alpha = ord$ - of - nat n
by ($metis$ ω - def $assms$ $elts$ - of - set $imageE$ inf)
have Ord (α) **using** $assms$ **by** $auto$
moreover
{ fix β
assume $\beta < \alpha$ Ord β $elts$ $\beta \approx elts$ α
with n **have** $elts$ $\beta \approx \{.. n \}$
by ($simp$ add : ord - of - nat - eq - $initial$ [of n] $eqpoll$ - $trans$ inj - on - def inj - on - $image$ - $eqpoll$ - $self$)
hence $False$ **using** $assms$ n $\langle Ord$ $\beta \rangle$ $\langle \beta < \alpha \rangle$ $\langle Ord$ (α) \rangle
by ($metis$ $\langle elts$ $\beta \approx elts$ $\alpha \rangle$ $card$ - $seteq$ $eqpoll$ - $finite$ - iff $eqpoll$ - iff - $card$ $finite$ - $lessThan$


```

less-eq-V-def less-le-not-le order-refl)
}
ultimately
  show (LEAST i. Ord i ∧ elts i ≈ elts α) = α
  by (metis (no-types, lifting) Least-equality Ord-linear-le eqpoll-refl less-le-not-le)
qed

lemma Card-ord-of-nat [simp]: Card (ord-of-nat n)
  by (simp add: ω-def nat-into-Card)

lemma Card-0 [iff]: Card 0
  by (simp add: nat-into-Card)

lemma CardI: [Ord i; ∧j. [j < i; Ord j]] ⇒ ¬ elts j ≈ elts i ⇒ Card i
  unfolding Card-def vcard-def
  by (metis Ord-Least Ord-linear-lt cardinal-epoll eqpoll-refl not-less-Ord-Least
vcard-def)

lemma vcard-0 [simp]: vcard 0 = 0
  using Card-0 Card-def by auto

lemma Ord-cardinal [simp,intro!]: Ord(vcard a)
  unfolding vcard-def by (metis Card-def Card-is-Ord cardinal-cong cardinal-epoll
vcard-def)

lemma gcard-big-0: ¬ small X ⇒ gcard X = 0
  by (simp add: gcard-def)

lemma gcard-eq-card:
  assumes finite X shows gcard X = ord-of-nat (card X)
proof -
  have ∧y. Ord y ∧ elts y ≈ X ⇒ ord-of-nat (card X) ≤ y
  by (metis assms eqpoll-finite-iff eqpoll-iff-card order-le-less ordertype-VWF-finite-nat
ordertype-eq-Ord)
  then have (LEAST i. Ord i ∧ elts i ≈ X) = ord-of-nat (card X)
  by (simp add: assms eqpoll-iff-card finite-Ord-omega Least-equality)
  with assms show ?thesis
  by (simp add: finite-imp-small gcard-def)
qed

lemma gcard-empty-0 [simp]: gcard {} = 0
  by (simp add: gcard-eq-card)

lemma gcard-single-1 [simp]: gcard {x} = 1
  by (simp add: gcard-eq-card one-V-def)

lemma Card-gcard [iff]: Card (gcard X)
  by (metis Card-0 Card-def cardinal-idem gcard-big-0 gcardinal-cong small-epoll
gcard-eq-vcard)

```

lemma *gcard-eqpoll*: $\text{small } X \implies \text{elts } (\text{gcard } X) \approx X$
by (*metis cardinal-eqpoll eqpoll-trans gcard-eq-vcard gcardinal-cong small-eqpoll*)

lemma *lepoll-imp-gcard-le*:
assumes $A \lesssim B$ *small B*
shows $\text{gcard } A \leq \text{gcard } B$
proof –
have $\text{elts } (\text{gcard } A) \approx A$ $\text{elts } (\text{gcard } B) \approx B$
by (*meson assms gcard-eqpoll lepoll-small*)
with $\langle A \lesssim B \rangle$ **show** *?thesis*
by (*metis Ord-cardinal Ord-linear2 eqpoll-sym gcard-eq-vcard gcardinal-cong lepoll-antisym lepoll-trans2 less-V-def less-eq-V-def subset-imp-lepoll*)
qed

lemma *gcard-image-le*:
assumes *small A* **shows** $\text{gcard } (f \text{ ` } A) \leq \text{gcard } A$
using *assms image-lepoll lepoll-imp-gcard-le* **by** *blast*

lemma *subset-imp-gcard-le*:
assumes $A \subseteq B$ *small B*
shows $\text{gcard } A \leq \text{gcard } B$
by (*simp add: assms lepoll-imp-gcard-le subset-imp-lepoll*)

lemma *gcard-le-lepoll*: $\llbracket \text{gcard } A \leq \alpha; \text{small } A \rrbracket \implies A \lesssim \text{elts } \alpha$
by (*meson eqpoll-sym gcard-eqpoll lepoll-trans1 less-eq-V-def subset-imp-lepoll*)

2.12 Cardinality of a set

The cardinals are the initial ordinals.

lemma *Card-iff-initial*: $\text{Card } \kappa \longleftrightarrow \text{Ord } \kappa \wedge (\forall \alpha. \text{Ord } \alpha \wedge \alpha < \kappa \longrightarrow \sim \text{elts } \alpha \approx \text{elts } \kappa)$
by (*metis CardI Card-def Card-is-Ord not-less-Ord-Least vcard-def*)

lemma *Card- ω [iff]*: $\text{Card } \omega$
by (*meson CardI Ord- ω eqpoll-finite-iff infinite-Ord-omega infinite- ω leD*)

lemma *lt-Card-imp-lesspoll*: $\llbracket i < a; \text{Card } a; \text{Ord } i \rrbracket \implies \text{elts } i \prec \text{elts } a$
by (*meson Card-iff-initial less-eq-V-def less-imp-le lesspoll-def subset-imp-lepoll*)

lemma *lepoll-imp-Card-le*:
assumes $\text{elts } a \lesssim \text{elts } b$ **shows** $\text{vcard } a \leq \text{vcard } b$
using *assms lepoll-imp-gcard-le* **by** *fastforce*

lemma *lepoll-cardinal-le*: $\llbracket \text{elts } A \lesssim \text{elts } i; \text{Ord } i \rrbracket \implies \text{vcard } A \leq i$
by (*metis Ord-Least Ord-linear2 dual-order.trans eqpoll-refl lepoll-imp-Card-le not-less-Ord-Least vcard-def*)

lemma *cardinal-le-lepoll*: $vcard\ A \leq \alpha \implies elts\ A \lesssim elts\ \alpha$
by (*meson cardinal-epoll eqpoll-sym lepoll-trans1 less-eq-V-def subset-imp-lepoll*)

lemma *lesspoll-imp-Card-less*:
assumes $elts\ a \prec elts\ b$ **shows** $vcard\ a < vcard\ b$
by (*metis assms cardinal-epoll eqpoll-sym eqpoll-trans lepoll-imp-Card-le less-V-def lesspoll-def*)

lemma *Card-Union* [*simp,intro*]:
assumes $A: \bigwedge x. x \in A \implies Card(x)$ **shows** $Card(\bigsqcup A)$
proof (*rule CardI*)
show $Ord(\bigsqcup A)$ **using** A
by (*simp add: Card-is-Ord Ord-Sup*)
next
fix j
assume $j: j < \bigsqcup A\ Ord\ j$
hence $\exists c \in A. j < c \wedge Card(c)$ **using** A
by (*meson Card-is-Ord Ord-linear2 ZFC-in-HOL.Sup-least leD*)
then obtain c **where** $c: c \in A\ j < c\ Card(c)$
by *blast*
hence $jls: elts\ j \prec elts\ c$
using $j(2)\ lt-Card-imp-lesspoll$ **by** *blast*
{ **assume** $eqp: elts\ j \approx elts\ (\bigsqcup A)$
have $elts\ c \lesssim elts\ (\bigsqcup A)$ **using** c
by (*metis Card-def Sup-V-def ZFC-in-HOL.Sup-upper cardinal-le-lepoll j(1) not-less-0*)
also have $\dots \approx elts\ j$ **by** (*rule eqpoll-sym [OF eqp]*)
also have $\dots \prec elts\ c$ **by** (*rule jls*)
finally have $elts\ c \prec elts\ c$.
hence *False*
by *auto*
} **thus** $\neg elts\ j \approx elts\ (\bigsqcup A)$ **by** *blast*
qed

lemma *Card-UN*: $(\bigwedge x. x \in A \implies Card(K\ x)) \implies Card(Sup\ (K\ ` A))$
by *blast*

2.13 Transfinite recursion for definitions based on the three cases of ordinals

definition
transrec3 $:: [V, [V, V] \Rightarrow V, [V, V \Rightarrow V] \Rightarrow V, V] \Rightarrow V$ **where**
transrec3 $a\ b\ c \equiv$
transrec $(\lambda r\ x.$
if $x=0$ *then* a
else if *Limit* x *then* $c\ x\ (\lambda y \in elts\ x. r\ y)$
else $b(pred\ x)\ (r\ (pred\ x))$)

lemma *transrec3-0* [*simp*]: $\text{transrec3 } a \ b \ c \ 0 = a$
by (*simp add: transrec transrec3-def*)

lemma *transrec3-succ* [*simp*]:
 $\text{transrec3 } a \ b \ c \ (\text{succ } i) = b \ i \ (\text{transrec3 } a \ b \ c \ i)$
by (*simp add: transrec transrec3-def*)

lemma *transrec3-Limit* [*simp*]:
 $\text{Limit } i \implies \text{transrec3 } a \ b \ c \ i = c \ i \ (\lambda j \in \text{elts } i. \text{transrec3 } a \ b \ c \ j)$
unfolding *transrec3-def*
by (*subst transrec*) *auto*

2.14 Cardinal Addition

definition *cadd* :: $[V, V] \Rightarrow V$ (**infixl** $\langle \oplus \rangle$ 65)
where $\kappa \oplus \mu \equiv \text{vcard } (\kappa \uplus \mu)$

2.14.1 Cardinal addition is commutative

lemma *vsum-commute-epoll*: $\text{elts } (a \uplus b) \approx \text{elts } (b \uplus a)$

proof –

have *bij-betw* $(\lambda z \in \text{elts } (a \uplus b). \text{sum-case } \text{Inr } \text{Inl } z) (\text{elts } (a \uplus b)) (\text{elts } (b \uplus a))$
unfolding *bij-betw-def*

proof (*intro conjI inj-onI*)

show *restrict* $(\text{sum-case } \text{Inr } \text{Inl}) (\text{elts } (a \uplus b)) \text{ ‘ } \text{elts } (a \uplus b) = \text{elts } (b \uplus a)$

apply *auto*

apply (*metis (no-types) imageI sum-case-Inr sum-iff*)

by (*metis Inl-in-sum-iff imageI sum-case-Inl*)

qed *auto*

then show *?thesis*

using *epoll-def* **by** *blast*

qed

lemma *cadd-commute*: $i \oplus j = j \oplus i$
by (*simp add: cadd-def cardinal-cong vsum-commute-epoll*)

2.14.2 Cardinal addition is associative

lemma *sum-assoc-bij*:

bij-betw $(\lambda z \in \text{elts } ((a \uplus b) \uplus c). \text{sum-case}(\text{sum-case } \text{Inl } (\lambda y. \text{Inr}(\text{Inl } y))) (\lambda y. \text{Inr}(\text{Inr } y)) z)$

$(\text{elts } ((a \uplus b) \uplus c)) (\text{elts } (a \uplus (b \uplus c)))$

by (*rule-tac f' = sum-case* $(\lambda x. \text{Inl } (\text{Inl } x)) (\text{sum-case } (\lambda x. \text{Inl } (\text{Inr } x)) \text{Inr})$)

in *bij-betw-byWitness*) *auto*

lemma *sum-assoc-epoll*: $\text{elts } ((a \uplus b) \uplus c) \approx \text{elts } (a \uplus (b \uplus c))$
unfolding *epoll-def* **by** (*metis sum-assoc-bij*)

lemma *elts-vcard-vsum-epoll*: $\text{elts } (\text{vcard } (i \uplus j)) \approx \text{Inl ‘ } \text{elts } i \cup \text{Inr ‘ } \text{elts } j$

proof –

have $\text{elts } (i \uplus j) \approx \text{Inl } \text{'elts } i \cup \text{Inr } \text{'elts } j$
by (*simp add: elts-vsum*)
then show *?thesis*
using *cardinal-epoll epoll-trans* **by** *blast*
qed

lemma *cadd-assoc*: $(i \oplus j) \oplus k = i \oplus (j \oplus k)$
proof (*unfold cadd-def, rule cardinal-cong*)
have $\text{elts } (\text{vcard}(i \uplus j) \uplus k) \approx \text{elts } ((i \uplus j) \uplus k)$
by (*auto simp: disjnt-def elts-vsum elts-vcard-vsum-epoll intro: Un-epoll-cong*)
also have $\dots \approx \text{elts } (i \uplus (j \uplus k))$
by (*rule sum-assoc-epoll*)
also have $\dots \approx \text{elts } (i \uplus \text{vcard}(j \uplus k))$
by (*auto simp: disjnt-def elts-vsum elts-vcard-vsum-epoll [THEN epoll-sym]*)
intro: Un-epoll-cong
finally show $\text{elts } (\text{vcard } (i \uplus j) \uplus k) \approx \text{elts } (i \uplus \text{vcard } (j \uplus k))$.
qed

lemma *cadd-left-commute*: $j \oplus (i \oplus k) = i \oplus (j \oplus k)$
using *cadd-assoc cadd-commute* **by** *force*

lemmas *cadd-ac = cadd-assoc cadd-commute cadd-left-commute*

0 is the identity for addition

lemma *vsum-0-epoll*: $\text{elts } (0 \uplus a) \approx \text{elts } a$
by (*simp add: elts-vsum*)

lemma *cadd-0 [simp]*: $\text{Card } \kappa \implies 0 \oplus \kappa = \kappa$
by (*metis Card-def cadd-def cardinal-cong vsum-0-epoll*)

lemma *cadd-0-right [simp]*: $\text{Card } \kappa \implies \kappa \oplus 0 = \kappa$
by (*simp add: cadd-commute*)

lemma *vsum-lepoll-self*: $\text{elts } a \lesssim \text{elts } (a \uplus b)$
unfolding *elts-vsum* **by** (*meson Inl-iff Un-upper1 inj-onI lepoll-def*)

lemma *cadd-le-self*:
assumes $\kappa: \text{Card } \kappa$ **shows** $\kappa \leq \kappa \oplus a$
proof (*unfold cadd-def*)
have $\kappa \leq \text{vcard } \kappa$
using *Card-def* κ **by** *auto*
also have $\dots \leq \text{vcard } (\kappa \uplus a)$
by (*simp add: lepoll-imp-Card-le vsum-lepoll-self*)
finally show $\kappa \leq \text{vcard } (\kappa \uplus a)$.
qed

Monotonicity of addition

lemma *cadd-le-mono*: $[\kappa' \leq \kappa; \mu' \leq \mu] \implies \kappa' \oplus \mu' \leq \kappa \oplus \mu$
unfolding *cadd-def*

by (*metis (no-types) lepoll-imp-Card-le less-eq-V-def subset-imp-lepoll sum-subset-iff*)

2.15 Cardinal multiplication

definition *cmult* :: $[V, V] \Rightarrow V$ (**infixl** $\langle \otimes \rangle$ 70)
where $\kappa \otimes \mu \equiv \text{vcard } (V\text{Sigma } \kappa (\lambda z. \mu))$

2.15.1 Cardinal multiplication is commutative

lemma *prod-bij*: $[[\text{bij-betw } f \ A \ C; \text{bij-betw } g \ B \ D]]$
 $\implies \text{bij-betw } (\lambda(x, y). (f \ x, \ g \ y)) \ (A \times B) \ (C \times D)$
apply (*rule* *bij-betw-byWitness* [**where** $f' = \lambda(x, y). (inv\text{-into } A \ f \ x, \ inv\text{-into } B \ g \ y)$])
apply (*auto simp: bij-betw-inv-into-left bij-betw-inv-into-right bij-betwE*)
using *bij-betwE bij-betw-inv-into* **apply** *blast+*
done

lemma *cmult-commute*: $i \otimes j = j \otimes i$

proof –

have $(\lambda(x, y). \langle x, y \rangle) \text{ ' } (elts \ i \times elts \ j) \approx (\lambda(x, y). \langle x, y \rangle) \text{ ' } (elts \ j \times elts \ i)$
by (*simp add: times-commute-epoll*)

then show *?thesis*

unfolding *cmult-def*

using *cardinal-cong elts-VSigma* **by** *auto*

qed

2.15.2 Cardinal multiplication is associative

lemma *elts-vcard-VSigma-epoll*: $elts \ (\text{vcard } (\text{vtimes } i \ j)) \approx elts \ i \times elts \ j$

proof –

have $elts \ (\text{vtimes } i \ j) \approx elts \ i \times elts \ j$

by (*simp add: elts-VSigma*)

then show *?thesis*

using *cardinal-epoll eqpoll-trans* **by** *blast*

qed

lemma *elts-cmult*: $elts \ (\kappa' \otimes \kappa) \approx elts \ \kappa' \times elts \ \kappa$

by (*simp add: cmult-def elts-vcard-VSigma-epoll*)

lemma *cmult-assoc*: $(i \otimes j) \otimes k = i \otimes (j \otimes k)$

unfolding *cmult-def*

proof (*rule cardinal-cong*)

have $elts \ (\text{vcard } (\text{vtimes } i \ j)) \times elts \ k \approx (elts \ i \times elts \ j) \times elts \ k$

by (*blast intro: times-epoll-cong elts-vcard-VSigma-epoll cardinal-epoll*)

also have $\dots \approx elts \ i \times (elts \ j \times elts \ k)$

by (*rule times-assoc-epoll*)

also have $\dots \approx elts \ i \times elts \ (\text{vcard } (\text{vtimes } j \ k))$

by (*blast intro: times-epoll-cong elts-vcard-VSigma-epoll cardinal-epoll eqpoll-sym*)

finally show $elts \ (V\text{Sigma } (\text{vcard } (\text{vtimes } i \ j)) \ (\lambda z. k)) \approx elts \ (V\text{Sigma } i \ (\lambda z.$

$\text{vcard } (\text{vtimes } j \ k)))$

by (*simp add: elts-VSigma*)
 qed

2.15.3 Cardinal multiplication distributes over addition

lemma *cadd-cmult-distrib*: $(i \oplus j) \otimes k = (i \otimes k) \oplus (j \otimes k)$
unfolding *cadd-def cmult-def*
proof (*rule cardinal-cong*)
have *elts (vtimes (vcard (i ⊔ j)) k) ≈ elts (vcard (vsum i j)) × elts k*
using *cardinal-epoll elts-vcard-VSigma-epoll eqpoll-sym eqpoll-trans* **by** *blast*
also have $\dots \approx (\text{Inl } \text{'elts } i \cup \text{Inr } \text{'elts } j) \times \text{elts } k$
using *elts-vcard-vsum-epoll times-epoll-cong* **by** *blast*
also have $\dots \approx (\text{Inl } \text{'elts } i) \times \text{elts } k \cup (\text{Inr } \text{'elts } j) \times \text{elts } k$
by (*simp add: Sigma-Un-distrib1*)
also have $\dots \approx \text{elts } (vtimes\ i\ k) \uplus vtimes\ j\ k$
unfolding *Plus-def*
by (*auto simp: elts-vsum elts-VSigma disjnt-iff intro!: Un-epoll-cong times-epoll-cong*)
also have $\dots \approx \text{elts } (vcard (vtimes\ i\ k) \uplus vtimes\ j\ k)$
by (*simp add: cardinal-epoll eqpoll-sym*)
also have $\dots \approx \text{elts } (vcard (vtimes\ i\ k) \uplus vcard (vtimes\ j\ k))$
by (*metis cadd-assoc cadd-def cardinal-cong cardinal-epoll vsum-0-epoll vsum-commute-epoll*)
finally show $\text{elts } (VSigma (vcard (i \uplus j)) (\lambda z. k))$
 $\approx \text{elts } (vcard (vtimes\ i\ k) \uplus vcard (vtimes\ j\ k))$.
 qed

Multiplication by 0 yields 0

lemma *cmult-0* [*simp*]: $0 \otimes i = 0$
using *Card-0 Card-def cmult-def* **by** *auto*

1 is the identity for multiplication

lemma *cmult-1* [*simp*]: **assumes** *Card* κ **shows** $1 \otimes \kappa = \kappa$
proof –
have *elts (vtimes (set {0}) κ) ≈ elts κ*
by (*auto simp: elts-VSigma intro!: times-singleton-epoll*)
then show *?thesis*
by (*metis Card-def assms cardinal-cong cmult-def elts-1 set-of-elts*)
 qed

2.16 Some inequalities for multiplication

lemma *cmult-square-le*: **assumes** *Card* κ **shows** $\kappa \leq \kappa \otimes \kappa$
proof –
have $\text{elts } \kappa \lesssim \text{elts } (\kappa \otimes \kappa)$
using *times-square-lepoll [of elts κ] cmult-def elts-vcard-VSigma-epoll eqpoll-sym*
lepoll-trans2
by *fastforce*
then show *?thesis*
using *Card-def assms cmult-def lepoll-cardinal-le* **by** *fastforce*
 qed

Multiplication by a non-empty set

lemma *cmult-le-self*: **assumes** $\text{Card } \kappa \ \alpha \neq 0$ **shows** $\kappa \leq \kappa \otimes \alpha$

proof –

have $\kappa = \text{vcard } \kappa$
using *Card-def* $\langle \text{Card } \kappa \rangle$ **by** *blast*
also have $\dots \leq \text{vcard } (\text{vtimes } \kappa \ \alpha)$
apply (*rule lepoll-imp-Card-le*)
apply (*simp add: elts-VSigma*)
by (*metis ZFC-in-HOL.ext* $\langle \alpha \neq 0 \rangle$ *elts-0 lepoll-times1*)
also have $\dots = \kappa \otimes \alpha$
by (*simp add: cmult-def*)
finally show *?thesis* .

qed

Monotonicity of multiplication

lemma *cmult-le-mono*: $\llbracket \kappa' \leq \kappa; \mu' \leq \mu \rrbracket \implies \kappa' \otimes \mu' \leq \kappa \otimes \mu$

unfolding *cmult-def*

by (*auto simp: elts-VSigma intro!: lepoll-imp-Card-le times-lepoll-mono subset-imp-lepoll*)

lemma *vcard-Sup-le-cmult*:

assumes *small U* **and** $\kappa: \bigwedge x. x \in U \implies \text{vcard } x \leq \kappa$

shows $\text{vcard } (\bigsqcup U) \leq \text{vcard } (\text{set } U) \otimes \kappa$

proof –

have $\exists f. f \in \text{elts } x \rightarrow \text{elts } \kappa \wedge \text{inj-on } f \ (\text{elts } x)$ **if** $x \in U$ **for** x

using κ [*OF that*] **by** (*metis cardinal-le-lepoll image-subset-iff-funcset lepoll-def*)

then obtain φ **where** $\varphi: \bigwedge x. x \in U \implies (\varphi x) \in \text{elts } x \rightarrow \text{elts } \kappa \wedge \text{inj-on } (\varphi x) \ (\text{elts } x)$

by *metis*

define u **where** $u \equiv \lambda y. @x. x \in U \wedge y \in \text{elts } x$

have $u: u y \in U \wedge y \in \text{elts } (u y)$ **if** $y \in \bigcup (\text{elts } \text{' } U)$ **for** y

unfolding *u-def* **by** (*metis (mono-tags, lifting)that someI2-ex UN-iff*)

define ψ **where** $\psi \equiv \lambda y. (u y, \varphi (u y) y)$

have $U: \text{elts } (\text{vcard } (\text{set } U)) \approx U$

by (*metis* $\langle \text{small } U \rangle$ *cardinal-epoll elts-of-set*)

have $\text{elts } (\bigsqcup U) = \bigcup (\text{elts } \text{' } U)$

using $\langle \text{small } U \rangle$ **by** *blast*

also have $\dots \lesssim U \times \text{elts } \kappa$

unfolding *lepoll-def*

proof (*intro exI conjI*)

show *inj-on* $\psi \ (\bigcup (\text{elts } \text{' } U))$

using φu **by** (*smt (verit) ψ -def inj-on-def prod.inject*)

show $\psi \text{' } \bigcup (\text{elts } \text{' } U) \subseteq U \times \text{elts } \kappa$

using φu **by** (*auto simp: ψ -def*)

qed

also have $\dots \approx \text{elts } (\text{vcard } (\text{set } U) \otimes \kappa)$

using *U elts-cmult eqpoll-sym eqpoll-trans times-epoll-cong* **by** *blast*

finally have $\text{elts } (\bigsqcup U) \lesssim \text{elts } (\text{vcard } (\text{set } U) \otimes \kappa)$.

then show *?thesis*

by (*simp add: cmult-def lepoll-cardinal-le*)

qed

2.17 The finite cardinals

lemma *succ-lepoll-succD*: $\text{elts} (\text{succ}(m)) \lesssim \text{elts} (\text{succ}(n)) \implies \text{elts } m \lesssim \text{elts } n$
by (*simp add: insert-lepoll-insertD*)

Congruence law for *succ* under equipollence

lemma *succ-epoll-cong*: $\text{elts } a \approx \text{elts } b \implies \text{elts} (\text{succ}(a)) \approx \text{elts} (\text{succ}(b))$
by (*simp add: succ-def insert-epoll-cong*)

lemma *sum-succ-epoll*: $\text{elts} (\text{succ } a \uplus b) \approx \text{elts} (\text{succ}(a \uplus b))$
unfolding *epoll-def*

proof (*rule exI*)

let $?f = \lambda z. \text{if } z = \text{Inl } a \text{ then } a \uplus b \text{ else } z$

let $?g = \lambda z. \text{if } z = a \uplus b \text{ then } \text{Inl } a \text{ else } z$

show *bij-betw* $?f$ ($\text{elts} (\text{succ } a \uplus b)$) ($\text{elts} (\text{succ} (a \uplus b))$)

apply (*rule bij-betw-byWitness* [**where** $f' = ?g$], *auto*)

apply (*metis Inl-in-sum-iff mem-not-refl*)

by (*metis Inr-in-sum-iff mem-not-refl*)

qed

lemma *cadd-succ*: $\text{succ } m \oplus n = \text{vcard} (\text{succ}(m \oplus n))$

proof (*unfold cadd-def*)

have [*intro*]: $\text{elts} (m \uplus n) \approx \text{elts} (\text{vcard} (m \uplus n))$

using *cardinal-epoll epoll-sym* **by** *blast*

have $\text{vcard} (\text{succ } m \uplus n) = \text{vcard} (\text{succ}(m \uplus n))$

by (*rule sum-succ-epoll* [*THEN cardinal-cong*])

also have $\dots = \text{vcard} (\text{succ}(\text{vcard} (m \uplus n)))$

by (*blast intro: succ-epoll-cong cardinal-cong*)

finally show $\text{vcard} (\text{succ } m \uplus n) = \text{vcard} (\text{succ}(\text{vcard} (m \uplus n)))$.

qed

lemma *nat-cadd-eq-add*: $\text{ord-of-nat } m \oplus \text{ord-of-nat } n = \text{ord-of-nat} (m + n)$

proof (*induct m*)

case (*Suc m*) **thus** $?case$

by (*metis Card-def Card-ord-of-nat add-Suc cadd-succ ord-of-nat.simps(2)*)

qed *auto*

lemma *vcard-disjoint-sup*:

assumes $x \sqcap y = 0$ **shows** $\text{vcard} (x \sqcup y) = \text{vcard } x \oplus \text{vcard } y$

proof –

have $\text{elts} (x \sqcup y) \approx \text{elts} (x \uplus y)$

unfolding *epoll-def*

proof (*rule exI*)

let $?f = \lambda z. \text{if } z \in \text{elts } x \text{ then } \text{Inl } z \text{ else } \text{Inr } z$

let $?g = \text{sum-case id id}$

show *bij-betw* $?f$ ($\text{elts} (x \sqcup y)$) ($\text{elts} (x \uplus y)$)

by (*rule bij-betw-byWitness* [**where** $f' = ?g$]) (*use assms V-disjoint-iff in*

```

auto)
qed
then show ?thesis
  by (metis cadd-commute cadd-def cardinal-cong cardinal-idem vsum-0-epoll
cadd-assoc)
qed

```

lemma *vcard-sup*: $vcard (x \sqcup y) \leq vcard x \oplus vcard y$

```

proof -
  have elts (x \sqcup y) \lesssim elts (x \uplus y)
    unfolding lepoll-def
  proof (intro exI conjI)
    let ?f = \lambda z. if z \in elts x then Inl z else Inr z
    show inj-on ?f (elts (x \sqcup y))
      by (simp add: inj-on-def)
    show ?f ` elts (x \sqcup y) \subseteq elts (x \uplus y)
      by force
  qed
then show ?thesis
  using cadd-ac
  by (metis cadd-def cardinal-cong cardinal-idem lepoll-imp-Card-le vsum-0-epoll)
qed

```

2.18 Infinite cardinals

definition *InfCard* :: $V \Rightarrow bool$
 where $InfCard \kappa \equiv Card \kappa \wedge \omega \leq \kappa$

lemma *InfCard-iff*: $InfCard \kappa \longleftrightarrow Card \kappa \wedge infinite (elts \kappa)$

```

proof (cases \omega \leq \kappa)
  case True
    then show ?thesis
      using inj-ord-of-nat lepoll-def less-eq-V-def
      by (auto simp: InfCard-def \omega-def infinite-le-lepoll)
  next
    case False
    then show ?thesis
      using Card-iff-initial InfCard-def infinite-Ord-omega by blast
qed

```

lemma *InfCard-ge-ord-of-nat*:

assumes $InfCard \kappa$ **shows** $ord-of-nat n \leq \kappa$
using *InfCard-def* *assms ord-of-nat-le-omega* **by** *blast*

lemma *InfCard-not-0[iff]*: $\neg InfCard 0$

by (*simp add: InfCard-iff*)

definition *csucc* :: $V \Rightarrow V$

where $csucc \kappa \equiv LEAST \kappa'. Ord \kappa' \wedge (Card \kappa' \wedge \kappa < \kappa')$

lemma *less-vcard-VPow*: $\text{vcard } A < \text{vcard } (\text{VPow } A)$
proof (*rule lesspoll-imp-Card-less*)
 show $\text{elts } A \prec \text{elts } (\text{VPow } A)$
 by (*simp add: elts-VPow down inj-on-def lesspoll-Pow-self*)
qed

lemma *greater-Card*:
 assumes *Card* κ **shows** $\kappa < \text{vcard } (\text{VPow } \kappa)$
proof –
 have $\kappa = \text{vcard } \kappa$
 using *Card-def assms by blast*
 also have $\dots < \text{vcard } (\text{VPow } \kappa)$
 proof (*rule lesspoll-imp-Card-less*)
 show $\text{elts } \kappa \prec \text{elts } (\text{VPow } \kappa)$
 by (*simp add: elts-VPow down inj-on-def lesspoll-Pow-self*)
qed
 finally show *?thesis* .
qed

lemma
 assumes *Card* κ
 shows *Card-csucc* [*simp*]: *Card* (*csucc* κ) **and** *less-csucc* [*simp*]: $\kappa < \text{csucc } \kappa$
proof –
 have *Card* (*csucc* κ) $\wedge \kappa < \text{csucc } \kappa$
 unfolding *csucc-def*
 proof (*rule Ord-LeastI2*)
 show *Card* ($\text{vcard } (\text{VPow } \kappa)$) $\wedge \kappa < (\text{vcard } (\text{VPow } \kappa))$
 using *Card-def assms greater-Card by auto*
qed *auto*
 then show *Card* (*csucc* κ) $\kappa < \text{csucc } \kappa$
 by *auto*
qed

lemma *le-csucc*:
 assumes *Card* κ **shows** $\kappa \leq \text{csucc } \kappa$
 by (*simp add: assms less-csucc less-imp-le*)

lemma *csucc-le*: $\llbracket \text{Card } \mu; \kappa \in \text{elts } \mu \rrbracket \implies \text{csucc } \kappa \leq \mu$
 unfolding *csucc-def*
 by (*simp add: Card-is-Ord Ord-Least-le OrdmemD*)

lemma *finite-csucc*: $a \in \text{elts } \omega \implies \text{csucc } a = \text{succ } a$
 unfolding *csucc-def*
 proof (*rule Least-equality*)
 show *Ord* (*ZFC-in-HOL.succ* a) \wedge *Card* (*ZFC-in-HOL.succ* a) $\wedge a < \text{ZFC-in-HOL.succ } a$
qed

```

    if  $a \in \text{elts } \omega$ 
    using that by (auto simp: less-V-def less-eq-V-def nat-into-Card)
  show ZFC-in-HOL.succ  $a \leq y$ 
    if  $a \in \text{elts } \omega$ 
    and  $\text{Ord } y \wedge \text{Card } y \wedge a < y$ 
    for  $y :: V$ 
    using that
    using Ord-mem-iff-lt dual-order.strict-implies-order by fastforce
qed

```

```

lemma Finite-imp-cardinal-cons [simp]:
  assumes FA: finite A and a:  $a \notin A$ 
  shows vcard (set (insert a A)) = csucc(vcard (set A))
proof -
  show ?thesis
    unfolding csucc-def
  proof (rule Least-equality [THEN sym])
    have small A
      by (simp add: FA Finite-V)
    then have  $\neg \text{elts (set A)} \approx \text{elts (set (insert a A))}$ 
      using FA a eqpoll-imp-lepoll eqpoll-sym finite-insert-lepoll by fastforce
    then show  $\text{Ord (vcard (set (insert a A)))} \wedge \text{Card (vcard (set (insert a A)))} \wedge$ 
      vcard (set A) < vcard (set (insert a A))
      by (simp add: Card-def lesspoll-imp-Card-less lesspoll-def subset-imp-lepoll
        subset-insertI)
    show vcard (set (insert a A))  $\leq i$ 
      if  $\text{Ord } i \wedge \text{Card } i \wedge \text{vcard (set A)} < i$  for  $i$ 
    proof -
      have elts (vcard (set A))  $\approx A$ 
        by (metis FA finite-imp-small cardinal-epoll elts-of-set)
      then have less:  $A \prec \text{elts } i$ 
        using eq-lesspoll-trans eqpoll-sym lt-Card-imp-lesspoll that by blast
      show ?thesis
        using that less by (auto simp: less-imp-insert-lepoll lepoll-cardinal-le)
    qed
  qed
qed

```

```

lemma vcard-finite-set: finite A  $\implies$  vcard (set A) = ord-of-nat (card A)
  by (induction A rule: finite-induct) (auto simp: set-empty  $\omega$ -def finite-csucc)

```

```

lemma lt-csucc-iff:
  assumes Ord  $\alpha$  Card  $\kappa$ 
  shows  $\alpha < \text{csucc } \kappa \iff \text{vcard } \alpha \leq \kappa$ 
proof
  show vcard  $\alpha \leq \kappa$  if  $\alpha < \text{csucc } \kappa$ 
  proof -
    have vcard  $\alpha \leq \text{csucc } \kappa$ 
      by (meson  $\langle \text{Ord } \alpha \rangle$  dual-order.trans lepoll-cardinal-le lepoll-refl less-le-not-le)
  qed

```

that)
then show *?thesis*
by (*metis (no-types) Card-def Card-iff-initial Ord-linear2 Ord-mem-iff-lt assms cardinal-epoll cardinal-idem csucc-le eq-iff eqpoll-sym that*)
qed
show $\alpha < csucc\ \kappa$ **if** $vcard\ \alpha \leq \kappa$
proof –
have $\neg csucc\ \kappa \leq \alpha$
using *that*
by (*metis Card-csucc Card-def assms(2) le-less-trans lepoll-imp-Card-le less-csucc less-eq-V-def less-le-not-le subset-imp-lepoll*)
then show *?thesis*
by (*meson Card-csucc Card-is-Ord Ord-linear2 assms*)
qed
qed

lemma *Card-lt-csucc-iff*: $\llbracket Card\ \kappa'; Card\ \kappa \rrbracket \implies (\kappa' < csucc\ \kappa) = (\kappa' \leq \kappa)$
by (*simp add: lt-csucc-iff Card-cardinal-eq Card-is-Ord*)

lemma *csucc-lt-csucc-iff*: $\llbracket Card\ \kappa'; Card\ \kappa \rrbracket \implies (csucc\ \kappa' < csucc\ \kappa) = (\kappa' < \kappa)$
by (*metis Card-csucc Card-is-Ord Card-lt-csucc-iff Ord-not-less*)

lemma *csucc-le-csucc-iff*: $\llbracket Card\ \kappa'; Card\ \kappa \rrbracket \implies (csucc\ \kappa' \leq csucc\ \kappa) = (\kappa' \leq \kappa)$
by (*metis Card-csucc Card-is-Ord Card-lt-csucc-iff Ord-not-less*)

lemma *csucc-0 [simp]*: $csucc\ 0 = 1$
by (*simp add: finite-csucc one-V-def*)

lemma *Card-Un [simp,intro]*:
assumes $Card\ x\ Card\ y$ **shows** $Card(x \sqcup y)$
by (*metis Card-is-Ord Ord-linear-le assms sup.absorb2 sup.orderE*)

lemma *InfCard-csucc*: $InfCard\ \kappa \implies InfCard\ (csucc\ \kappa)$
using *InfCard-def le-csucc* **by** *auto*

Kunen's Lemma 10.11

lemma *InfCard-is-Limit*:
assumes $InfCard\ \kappa$ **shows** $Limit\ \kappa$
proof (*rule non-succ-LimitI*)
show $\kappa \neq 0$
using *InfCard-def assms mem-not-refl* **by** *blast*
show $Ord\ \kappa$
using *Card-is-Ord InfCard-def assms* **by** *blast*
show $ZFC\text{-in-HOL.succ}\ y \neq \kappa$ **for** y
proof
assume $succ\ y = \kappa$
then have $Card\ (succ\ y)$
using *InfCard-def assms* **by** *auto*
moreover have $\omega \leq y$

by (*metis* *InfCard-iff* *Ord-in-Ord* $\langle \text{Ord } \kappa \rangle$ $\langle \text{ZFC-in-HOL.succ } y = \kappa \rangle$ *assms*
elts-succ *finite-insert* *infinite-Ord-omega* *insertI1*)
moreover have $\text{elts } y \approx \text{elts } (\text{succ } y)$
using *InfCard-iff* $\langle \text{ZFC-in-HOL.succ } y = \kappa \rangle$ *assms* *eqpoll-sym* *infinite-insert-epoll*
by *fastforce*
ultimately show *False*
by (*metis* *Card-iff-initial* *Ord-in-Ord* *OrdmemD* *elts-succ* *insertI1*)
qed
qed

2.19 Toward's Kunen's Corollary 10.13 (1)

Kunen's Theorem 10.12

lemma *InfCard-csquare-eq*:
assumes *InfCard* κ **shows** $\kappa \otimes \kappa = \kappa$
using *infinite-times-epoll-self* [*of elts* κ] *assms*
unfolding *InfCard-iff* *Card-def*
by (*metis* *cardinal-cong* *cardinal-epoll* *cmult-def* *elts-vcard-VSigma-epoll* *epoll-trans*)

lemma *InfCard-le-cmult-eq*:
assumes *InfCard* κ $\mu \leq \kappa$ $\mu \neq 0$
shows $\kappa \otimes \mu = \kappa$
proof (*rule* *order-antisym*)
have $\kappa \otimes \mu \leq \kappa \otimes \kappa$
by (*simp* *add*: *assms*(2) *cmult-le-mono*)
also have $\dots \leq \kappa$
by (*simp* *add*: *InfCard-csquare-eq* *assms*(1))
finally show $\kappa \otimes \mu \leq \kappa$.
show $\kappa \leq \kappa \otimes \mu$
using *InfCard-def* *assms*(1) *assms*(3) *cmult-le-self* **by** *auto*
qed

Kunen's Corollary 10.13 (1), for cardinal multiplication

lemma *InfCard-cmult-eq*: $\llbracket \text{InfCard } \kappa; \text{InfCard } \mu \rrbracket \implies \kappa \otimes \mu = \kappa \sqcup \mu$
by (*metis* *Card-is-Ord* *InfCard-def* *InfCard-le-cmult-eq* *Ord-linear-le* *cmult-commute*
inf-sup-aci(5) *mem-not-refl* *sup.orderE* *sup-V-0-right* *zero-in-omega*)

lemma *cmult-succ*:
 $\text{succ}(m) \otimes n = n \oplus (m \otimes n)$
unfolding *cmult-def* *cadd-def*
proof (*rule* *cardinal-cong*)
have $\text{elts } (v\text{times } (\text{ZFC-in-HOL.succ } m) n) \approx \text{elts } n \langle + \rangle \text{elts } m \times \text{elts } n$
by (*simp* *add*: *elts-VSigma* *prod-insert-epoll*)
also have $\dots \approx \text{elts } (n \uplus \text{vcard } (v\text{times } m n))$
unfolding *elts-VSigma* *elts-vsum* *Plus-def*
proof (*rule* *Un-epoll-cong*)
show $(\text{Sum-Type.Inr } \langle \text{elts } m \times \text{elts } n \rangle :: (V + V \times V) \text{ set}) \approx \text{Inr } \langle \text{elts } (\text{vcard } (v\text{times } m n)) \rangle$
by (*simp* *add*: *elts-vcard-VSigma-epoll* *epoll-sym*)

qed (*auto simp: disjnt-def*)
finally show $\text{elts } (\text{vtimes } (\text{ZFC-in-HOL.succ } m) n) \approx \text{elts } (n \sqcup \text{vcard } (\text{vtimes } m n))$.
qed

lemma *cmult-2*:
assumes *Card n* **shows** $\text{ord-of-nat } 2 \otimes n = n \oplus n$
proof –
have $\text{ord-of-nat } 2 = \text{succ } (\text{succ } 0)$
by *force*
then show *?thesis*
by (*simp add: cmult-succ assms*)
qed

lemma *InfCard-cdouble-eq*:
assumes *InfCard* κ **shows** $\kappa \oplus \kappa = \kappa$
proof –
have $\kappa \oplus \kappa = \kappa \otimes \text{ord-of-nat } 2$
using *InfCard-def assms cmult-2 cmult-commute* **by** *auto*
also have $\dots = \kappa$
by (*simp add: InfCard-le-cmult-eq InfCard-ge-ord-of-nat assms*)
finally show *?thesis* .
qed

Corollary 10.13 (1), for cardinal addition

lemma *InfCard-le-cadd-eq*: $\llbracket \text{InfCard } \kappa; \mu \leq \kappa \rrbracket \implies \kappa \oplus \mu = \kappa$
by (*metis InfCard-cdouble-eq InfCard-def antisym cadd-le-mono cadd-le-self*)

lemma *InfCard-cadd-eq*: $\llbracket \text{InfCard } \kappa; \text{InfCard } \mu \rrbracket \implies \kappa \oplus \mu = \kappa \sqcup \mu$
by (*metis Card-iff-initial InfCard-def InfCard-le-cadd-eq Ord-linear-le cadd-commute sup.absorb2 sup.orderE*)

lemma *csucc-le-Card-iff*: $\llbracket \text{Card } \kappa'; \text{Card } \kappa \rrbracket \implies \text{csucc } \kappa' \leq \kappa \iff \kappa' < \kappa$
by (*metis Card-csucc Card-is-Ord Card-lt-csucc-iff Ord-not-le*)

lemma *cadd-InfCard-le*:
assumes $\alpha \leq \kappa \beta \leq \kappa$ *InfCard* κ
shows $\alpha \oplus \beta \leq \kappa$
using *assms* **by** (*metis InfCard-cdouble-eq cadd-le-mono*)

lemma *cmult-InfCard-le*:
assumes $\alpha \leq \kappa \beta \leq \kappa$ *InfCard* κ
shows $\alpha \otimes \beta \leq \kappa$
using *assms*
by (*metis InfCard-csquare-eq cmult-le-mono*)

2.20 The Aleph-sequence

This is the well-known transfinite enumeration of the cardinal numbers.

definition *Aleph* :: $V \Rightarrow V \quad (\aleph \rightarrow [90] 90)$
where *Aleph* $\equiv \text{transrec } (\lambda f x. \omega \sqcup \bigsqcup ((\lambda y. \text{csucc}(f y)) \text{ `elts } x))$

lemma *Aleph*: *Aleph* $\alpha = \omega \sqcup (\bigsqcup_{y \in \text{elts } \alpha}. \text{csucc } (\text{Aleph } y))$
by (*simp add: Aleph-def transrec[of - α]*)

lemma *InfCard-Aleph* [*simp, intro*]: *InfCard*(*Aleph* x)
proof (*induction x rule: eps-induct*)
case (*step x*)
then show ?*case*
by (*simp add: Aleph [of x] InfCard-def Card-UN step*)
qed

lemma *Card-Aleph* [*simp, intro*]: *Card*(*Aleph* x)
using *InfCard-def* **by** *auto*

lemma *Aleph-0* [*simp*]: $\aleph 0 = \omega$
by (*simp add: Aleph [of 0]*)

lemma *mem-Aleph-succ*: $\aleph \alpha \in \text{elts } (\text{Aleph } (\text{succ } \alpha))$
apply (*simp add: Aleph [of succ α]*)
by (*meson InfCard-Aleph Card-csucc Card-is-Ord InfCard-def Ord-mem-iff-lt less-csucc*)

lemma *Aleph-lt-succD* [*simp*]: $\aleph \alpha < \text{Aleph } (\text{succ } \alpha)$
by (*simp add: InfCard-is-Limit Limit-is-Ord OrdmemD mem-Aleph-succ*)

lemma *Aleph-succ* [*simp*]: *Aleph* (*succ* x) = *csucc* (*Aleph* x)
proof (*rule antisym*)
show *Aleph* (*ZFC-in-HOL.succ* x) \leq *csucc* (*Aleph* x)
apply (*simp add: Aleph [of succ x]*)
by (*metis Aleph InfCard-Aleph InfCard-def Sup-V-insert le-csucc le-sup-iff order-refl*
replacement small-elts)
show *csucc* (*Aleph* x) \leq *Aleph* (*ZFC-in-HOL.succ* x)
by (*force simp add: Aleph [of succ x]*)
qed

lemma *csucc-Aleph-le-Aleph*: $\alpha \in \text{elts } \beta \implies \text{csucc } (\aleph \alpha) \leq \aleph \beta$
by (*metis Aleph ZFC-in-HOL.SUP-le-iff replacement small-elts sup-ge2*)

lemma *Aleph-in-Aleph*: $\alpha \in \text{elts } \beta \implies \aleph \alpha \in \text{elts } (\aleph \beta)$
using *csucc-Aleph-le-Aleph mem-Aleph-succ* **by** *auto*

lemma *Aleph-Limit*:
assumes *Limit* γ
shows *Aleph* $\gamma = \bigsqcup (\text{Aleph } \text{ `elts } \gamma)$
proof –
have [*simp*]: $\gamma \neq 0$

using *assms* **by** *auto*
show *?thesis*
proof (*rule antisym*)
show $\text{Aleph } \gamma \leq \bigsqcup (\text{Aleph } \ulcorner \text{elts } \gamma)$
apply (*simp add: Aleph [of]*)
by (*metis (mono-tags, lifting) Aleph-0 Aleph-succ Limit-def ZFC-in-HOL.SUP-le-iff*)

ZFC-in-HOL.Sup-upper assms imageI replacement small-elts
show $\bigsqcup (\text{Aleph } \ulcorner \text{elts } \gamma) \leq \text{Aleph } \gamma$
apply (*simp add: cSup-le-iff*)
by (*meson InfCard-Aleph InfCard-def csucc-Aleph-le-Aleph le-csucc order-trans*)
qed
qed

lemma *Aleph-increasing*:
assumes *ab*: $\alpha < \beta$ *Ord* α *Ord* β **shows** $\aleph_\alpha < \aleph_\beta$
by (*meson Aleph-in-Aleph InfCard-Aleph Card-iff-initial InfCard-def Ord-mem-iff-lt assms*)

lemma *countable-iff-le-Aleph0*: $\text{countable } (\text{elts } A) \longleftrightarrow \text{vcard } A \leq \aleph_0$

proof
show $\text{vcard } A \leq \aleph_0$
if *countable* (*elts* *A*)
proof (*cases finite* (*elts* *A*))
case *True*
then show *?thesis*
using *vcard-finite-set* **by** *fastforce*
next
case *False*
then have *elts* $\omega \approx \text{elts } A$
using *countableE-infinite* [*OF that*]
by (*simp add: eqpoll-def* ω -*def*)
(meson bij-betw-def bij-betw-inv bij-betw-trans inj-ord-of-nat)
then show *?thesis*
using *Card- ω Card-def cardinal-cong vcard-def* **by** *auto*
qed
show *countable* (*elts* *A*)
if $\text{vcard } A \leq \text{Aleph } 0$
proof –
have *elts* $A \lesssim \text{elts } \omega$
using *cardinal-le-lepoll* [*OF that*] **by** *simp*
then show *?thesis*
by (*simp add: countable-iff-lepoll* ω -*def inj-ord-of-nat*)
qed
qed

lemma *Aleph-csquare-eq* [*simp*]: $\aleph_\alpha \otimes \aleph_\alpha = \aleph_\alpha$
using *InfCard-csquare-eq* **by** *auto*

lemma *vcard-Aleph* [*simp*]: $vcard (\aleph\alpha) = \aleph\alpha$
using *Card-def InfCard-Aleph InfCard-def* **by** *auto*

lemma *omega-le-Aleph* [*simp*]: $\omega \leq \aleph\alpha$
using *InfCard-def* **by** *auto*

lemma *finite-iff-less-Aleph0*: $finite (elts x) \longleftrightarrow vcard x < \omega$
proof

show $finite (elts x) \implies vcard x < \omega$
by (*metis Card- ω Card-def finite-lesspoll-infinite infinite- ω lesspoll-imp-Card-less*)
show $vcard x < \omega \implies finite (elts x)$
by (*meson Ord-cardinal cardinal-epoll eqpoll-finite-iff infinite-Ord-omega less-le-not-le*)
qed

lemma *countable-iff-vcard-less1*: $countable (elts x) \longleftrightarrow vcard x < \aleph 1$
by (*simp add: countable-iff-le-Aleph0 lt-csucc-iff one-V-def*)

lemma *countable-infinite-vcard*: $countable (elts x) \wedge infinite (elts x) \longleftrightarrow vcard x = \aleph 0$
by (*metis Aleph-0 countable-iff-le-Aleph0 dual-order.refl finite-iff-less-Aleph0 less-V-def*)

2.21 The ordinal $\omega 1$

abbreviation $\omega 1 \equiv Aleph 1$

lemma *Ord- $\omega 1$* [*simp*]: $Ord \omega 1$
by (*metis Ord-cardinal vcard-Aleph*)

lemma *omega- $\omega 1$* [*iff*]: $\omega \in elts \omega 1$
by (*metis Aleph-0 mem-Aleph-succ one-V-def*)

lemma *ord-of-nat- $\omega 1$* [*iff*]: $ord-of-nat n \in elts \omega 1$
using *Ord- $\omega 1$ Ord-trans* **by** *blast*

lemma *countable-iff-less- $\omega 1$* :
assumes $Ord \alpha$
shows $countable (elts \alpha) \longleftrightarrow \alpha < \omega 1$
by (*simp add: assms countable-iff-le-Aleph0 lt-csucc-iff one-V-def*)

lemma *less- $\omega 1$ -imp-countable*:
assumes $\alpha \in elts \omega 1$
shows $countable (elts \alpha)$
using *Ord- $\omega 1$ Ord-in-Ord OrdmemD assms countable-iff-less- $\omega 1$* **by** *blast*

lemma *$\omega 1$ -gt0* [*simp*]: $\omega 1 > 0$
using *Ord- $\omega 1$ Ord-trans OrdmemD* **by** *blast*

lemma *$\omega 1$ -gt1* [*simp*]: $\omega 1 > 1$
using *Ord- $\omega 1$ OrdmemD ω -gt1 less-trans* **by** *blast*

lemma *Limit- $\omega 1$* [*simp*]: *Limit $\omega 1$*
by (*simp add: InfCard-def InfCard-is-Limit le-csucc one-V-def*)
end

3 Addition and Multiplication of Sets

theory *Kirby*
imports *ZFC-Cardinals*

begin

3.1 Generalised Addition

Source: Laurence Kirby, Addition and multiplication of sets Math. Log. Quart. 53, No. 1, 52-65 (2007) / DOI 10.1002/malq.200610026 <http://faculty.baruch.cuny.edu/lkirby/mlqarticlejan2007.pdf>

3.1.1 Addition is a monoid

instantiation *V* :: *plus*
begin

This definition is credited to Tarski

definition *plus-V* :: $V \Rightarrow V \Rightarrow V$
where *plus-V* $x \equiv \text{transrec } (\lambda f z. x \sqcup \text{set } (f \text{ ` elts } z))$

instance ..
end

definition *lift* :: $V \Rightarrow V \Rightarrow V$
where *lift* $x y \equiv \text{set } (\text{plus } x \text{ ` elts } y)$

lemma *plus*: $x + y = x \sqcup \text{set } ((+)x \text{ ` elts } y)$
unfolding *plus-V-def* **by** (*subst transrec*) *auto*

lemma *plus-eq-lift*: $x + y = x \sqcup \text{lift } x y$
unfolding *lift-def* **using** *plus* **by** *blast*

Lemma 3.2

lemma *lift-sup-distrib*: $\text{lift } x (a \sqcup b) = \text{lift } x a \sqcup \text{lift } x b$
by (*simp add: image-Un lift-def sup-V-def*)

lemma *lift-Sup-distrib*: $\text{small } Y \Longrightarrow \text{lift } x (\bigsqcup Y) = \bigsqcup (\text{lift } x \text{ ` } Y)$
by (*auto simp: lift-def Sup-V-def image-Union*)

lemma *add-Sup-distrib*:

fixes $x::V$ **shows** $y \neq 0 \implies x + (\bigsqcup z \in \text{elts } y. f z) = (\bigsqcup z \in \text{elts } y. x + f z)$
by (*auto simp: plus-eq-lift SUP-sup-distrib lift-Sup-distrib image-image*)

lemma *Limit-add-Sup-distrib*:

fixes $x::V$ **shows** $\text{Limit } \alpha \implies x + (\bigsqcup z \in \text{elts } \alpha. f z) = (\bigsqcup z \in \text{elts } \alpha. x + f z)$
using *add-Sup-distrib* **by** *force*

Proposition 3.3(ii)

instantiation $V :: \text{monoid-add}$

begin

instance

proof

show $a + b + c = a + (b + c)$ **for** $a b c :: V$

proof (*induction c rule: eps-induct*)

case (*step c*)

have $(a+b) + c = a + b \sqcup \text{set } ((+) (a + b) \text{ 'elts } c)$

by (*metis plus*)

also have $\dots = a \sqcup \text{lift } a b \sqcup \text{set } ((\lambda u. a + (b+u)) \text{ 'elts } c)$

using *plus-eq-lift step.IH* **by** *auto*

also have $\dots = a \sqcup \text{lift } a (b + c)$

proof –

have $\text{lift } a b \sqcup \text{set } ((\lambda u. a + (b + u)) \text{ 'elts } c) = \text{lift } a (b + c)$

unfolding *lift-def*

by (*metis elts-of-set image-image lift-def lift-sup-distrib plus-eq-lift replacement small-elts*)

then show *?thesis*

by (*simp add: sup-assoc*)

qed

also have $\dots = a + (b + c)$

using *plus-eq-lift* **by** *auto*

finally show *?case* .

qed

show $0 + x = x$ **for** $x :: V$

proof (*induction rule: eps-induct*)

case (*step x*)

then show *?case*

by (*subst plus*) *auto*

qed

show $x + 0 = x$ **for** $x :: V$

by (*subst plus*) *auto*

qed

end

lemma *lift-0 [simp]: lift 0 x = x*

by (*simp add: lift-def*)

lemma *lift-by0 [simp]: lift x 0 = 0*

by (*simp add: lift-def*)

lemma *lift-by1* [*simp*]: $\text{lift } x \ 1 = \text{set}\{x\}$
by (*simp add: lift-def*)

lemma *add-eq-0-iff* [*simp*]:
fixes $x \ y :: V$
shows $x + y = 0 \longleftrightarrow x = 0 \wedge y = 0$
proof *safe*
show $x = 0$ **if** $x + y = 0$
by (*metis that le-imp-less-or-eq not-less-0 plus sup-ge1*)
then show $y = 0$ **if** $x + y = 0$
using *that by auto*
qed *auto*

lemma *plus-vinsert*: $x + \text{vinsert } z \ y = \text{vinsert } (x+z) \ (x + y)$
proof –
have $f1: \text{elts } (x + y) = \text{elts } x \cup (+) \ x \ \text{' } \text{elts } \ y$
by (*metis elts-of-set lift-def plus-eq-lift replacement small-Un small-elts sup-V-def*)
moreover have $\text{lift } x \ (\text{vinsert } z \ y) = \text{set } ((+) \ x \ \text{' } \text{elts } (\text{set } (\text{insert } z \ (\text{elts } y))))$
using *vinsert-def lift-def by presburger*
ultimately show *?thesis*
by (*simp add: vinsert-def plus-eq-lift sup-V-def*)
qed

lemma *plus-V-succ-right*: $x + \text{succ } y = \text{succ } (x + y)$
by (*metis plus-vinsert succ-def*)

lemma *succ-eq-add1*: $\text{succ } x = x + 1$
by (*simp add: plus-V-succ-right one-V-def*)

lemma *ord-of-nat-add*: $\text{ord-of-nat } (m+n) = \text{ord-of-nat } m + \text{ord-of-nat } n$
by (*induction n (auto simp: plus-V-succ-right)*)

lemma *succ-0-plus-eq* [*simp*]:
assumes $\alpha \in \text{elts } \omega$
shows $\text{succ } 0 + \alpha = \text{succ } \alpha$
proof –
obtain n **where** $\alpha = \text{ord-of-nat } n$
using *assms elts- ω by blast*
then show *?thesis*
by (*metis One-nat-def ord-of-nat.simps ord-of-nat-add plus-1-eq-Suc*)
qed

lemma *omega-closed-add* [*intro*]:
assumes $\alpha \in \text{elts } \omega \ \beta \in \text{elts } \omega$ **shows** $\alpha + \beta \in \text{elts } \omega$
proof –
obtain $m \ n$ **where** $\alpha = \text{ord-of-nat } m \ \beta = \text{ord-of-nat } n$
using *assms elts- ω by auto*
then have $\alpha + \beta = \text{ord-of-nat } (m+n)$
using *ord-of-nat-add by auto*

then show *?thesis*
by (*simp add: ω -def*)
qed

lemma *mem-plus-V-E*:
assumes $l \in \text{elts } (x + y)$
obtains $l \in \text{elts } x \mid z$ **where** $z \in \text{elts } y \mid l = x + z$
using l **by** (*auto simp: plus [of $x \ y$] split: if-split-asm*)

lemma *not-add-less-right*: **assumes** *Ord y* **shows** $\neg (x + y < x)$
using *assms*
proof (*induction rule: Ord-induct*)
case (*step i*)
then show *?case*
by (*metis less-le-not-le plus sup-ge1*)
qed

lemma *not-add-mem-right*: $\neg (x + y \in \text{elts } x)$
by (*metis sup-ge1 mem-not-refl plus vsubsetD*)

Proposition 3.3(iii)

lemma *add-not-less-TC-self*: $\neg x + y \sqsubset x$
proof (*induction y arbitrary: x rule: eps-induct*)
case (*step y*)
then show *?case*
using *less-TC-imp-not-le plus-eq-lift* **by** *fastforce*
qed

lemma *TC-sup-lift*: $TC \ x \sqcap \text{lift } x \ y = 0$
proof –
have $\text{elts } (TC \ x) \cap \text{elts } (\text{set } ((+) \ x \ \text{elts } y)) = \{\}$
using *add-not-less-TC-self* **by** (*auto simp: less-TC-def*)
then have $TC \ x \sqcap \text{set } ((+) \ x \ \text{elts } y) = \text{set } \{\}$
by (*metis inf-V-def*)
then show *?thesis*
using *lift-def* **by** *auto*
qed

lemma *lift-lift*: $\text{lift } x \ (\text{lift } y \ z) = \text{lift } (x+y) \ z$
using *add.assoc* **by** (*auto simp: lift-def*)

lemma *lift-self-disjoint*: $x \sqcap \text{lift } x \ u = 0$
by (*metis TC-sup-lift arg-subset-TC inf.absorb-iff2 inf-assoc inf-sup-aci(3) lift-0*)

lemma *sup-lift-eq-lift*:
assumes $x \sqcup \text{lift } x \ u = x \sqcup \text{lift } x \ v$
shows $\text{lift } x \ u = \text{lift } x \ v$
by (*metis (no-types) assms inf-sup-absorb inf-sup-distrib2 lift-self-disjoint sup-commute sup-inf-absorb*)

3.1.2 Deeper properties of addition

Proposition 3.4(i)

proposition *lift-eq-lift*: $\text{lift } x \ y = \text{lift } x \ z \implies y = z$

proof (*induction y arbitrary: z rule: eps-induct*)

case (*step y*)

show *?case*

proof (*intro vsubsetI order-antisym*)

show $u \in \text{elts } z$ **if** $u \in \text{elts } y$ **for** u

proof –

have $x+u \in \text{elts } (\text{lift } x \ z)$

using *lift-def step.prem*s that **by** *fastforce*

then obtain v **where** $v \in \text{elts } z$ $x+u = x+v$

using *lift-def* **by** *auto*

then have $\text{lift } x \ u = \text{lift } x \ v$

using *sup-lift-eq-lift* **by** (*simp add: plus-eq-lift*)

then have $u=v$

using *step.IH* that **by** *blast*

then show *?thesis*

using $\langle v \in \text{elts } z \rangle$ **by** *blast*

qed

show $u \in \text{elts } y$ **if** $u \in \text{elts } z$ **for** u

proof –

have $x+u \in \text{elts } (\text{lift } x \ y)$

using *lift-def step.prem*s that **by** *fastforce*

then obtain v **where** $v \in \text{elts } y$ $x+u = x+v$

using *lift-def* **by** *auto*

then have $\text{lift } x \ u = \text{lift } x \ v$

using *sup-lift-eq-lift* **by** (*simp add: plus-eq-lift*)

then have $u=v$

using *step.IH* **by** (*metis* $\langle v \in \text{elts } y \rangle$)

then show *?thesis*

using $\langle v \in \text{elts } y \rangle$ **by** *auto*

qed

qed

qed

corollary *inj-lift*: *inj-on* (*lift x*) A

by (*auto simp: inj-on-def dest: lift-eq-lift*)

corollary *add-right-cancel* [*iff*]:

fixes $x \ y \ z :: V$ **shows** $x+y = x+z \longleftrightarrow y=z$

by (*metis lift-eq-lift plus-eq-lift sup-lift-eq-lift*)

corollary *add-mem-right-cancel* [*iff*]:

fixes $x \ y \ z :: V$ **shows** $x+y \in \text{elts } (x+z) \longleftrightarrow y \in \text{elts } z$

apply *safe*

apply (*metis mem-plus-V-E not-add-mem-right add-right-cancel*)

by (*metis ZFC-in-HOL.ext dual-order.antisym elts-vinsert insert-subset order-refl*)

plus-vinsert)

corollary *add-le-cancel-left* [*iff*]:

fixes $x y z :: V$ **shows** $x + y \leq x + z \longleftrightarrow y \leq z$

by *auto* (*metis add-mem-right-cancel mem-plus-V-E plus-sup-ge1 vsubsetD*)

corollary *add-less-cancel-left* [*iff*]:

fixes $x y z :: V$ **shows** $x + y < x + z \longleftrightarrow y < z$

by (*simp add: less-le-not-le*)

corollary *lift-le-self* [*simp*]: $\text{lift } x \ y \leq x \longleftrightarrow y = 0$

by (*auto simp: inf.absorb-iff2 lift-eq-lift lift-self-disjoint*)

lemma *succ-less- ω -imp*: $\text{succ } x < \omega \implies x < \omega$

by (*metis add-le-cancel-left add.right-neutral le-0 le-less-trans succ-eq-add1*)

Proposition 3.5

lemma *card-lift*: $\text{vcard } (\text{lift } x \ y) = \text{vcard } y$

proof (*rule cardinal-cong*)

have *bij-betw* $((+)x)$ $(\text{elts } y)$ $(\text{elts } (\text{lift } x \ y))$

unfolding *bij-betw-def*

by (*simp add: inj-on-def lift-def*)

then show $\text{elts } (\text{lift } x \ y) \approx \text{elts } y$

using *eqpoll-def eqpoll-sym* **by** *blast*

qed

lemma *eqpoll-lift*: $\text{elts } (\text{lift } x \ y) \approx \text{elts } y$

by (*metis card-lift cardinal-*eqpoll* eqpoll-sym eqpoll-trans*)

lemma *vcard-add*: $\text{vcard } (x + y) = \text{vcard } x \oplus \text{vcard } y$

using *card-lift* [*of x y*] *lift-self-disjoint* [*of x*]

by (*simp add: plus-eq-lift vcard-disjoint-sup*)

lemma *countable-add*:

assumes *countable* $(\text{elts } A)$ *countable* $(\text{elts } B)$

shows *countable* $(\text{elts } (A+B))$

proof –

have $\text{vcard } A \leq \aleph_0$ $\text{vcard } B \leq \aleph_0$

using *assms countable-iff-le-Aleph0* **by** *blast+*

then have $\text{vcard } (A+B) \leq \aleph_0$

unfolding *vcard-add*

by (*metis Aleph-0 Card- ω InfCard-cdouble-eq InfCard-def cadd-le-mono order-refl*)

then show *?thesis*

by (*simp add: countable-iff-le-Aleph0*)

qed

Proposition 3.6

proposition *TC-add*: $TC (x + y) = TC \ x \sqcup \text{lift } x \ (TC \ y)$

proof (*induction y rule: eps-induct*)
case (*step y*)
have *: $\sqcup (TC \text{ ' } (+) x \text{ ' } elts y) = TC x \sqcup (\sqcup u \in elts y. TC (set ((+) x \text{ ' } elts u)))$
if $elts y \neq \{\}$
proof –
obtain w **where** $w \in elts y$
using $\langle elts y \neq \{\} \rangle$ **by** *blast*
then have $TC x \leq TC (x + w)$
by (*simp add: step.IH*)
then have †: $TC x \leq (\sqcup w \in elts y. TC (x + w))$
using $\langle w \in elts y \rangle$ **by** *blast*
show *?thesis*
using *that*
apply (*intro conjI ballI impI order-antisym; clarsimp simp add: image-comp*)
†)
apply(*metis TC-sup-distrib Un-iff elts-sup-iff plus*)
by (*metis TC-least Transset-TC arg-subset-TC le-sup-iff plus vsubsetD*)
qed
have $TC (x + y) = (x + y) \sqcup \sqcup (TC \text{ ' } elts (x + y))$
using *TC* **by** *blast*
also have $\dots = x \sqcup lift x y \sqcup \sqcup (TC \text{ ' } elts x) \sqcup \sqcup ((\lambda u. TC (x+u)) \text{ ' } elts y)$
apply (*simp add: plus-eq-lift image-Un Sup-Un-distrib sup.left-commute sup-assoc*
TC-sup-distrib SUP-sup-distrib)
apply (*simp add: lift-def sup.commute sup-aci **)
done
also have $\dots = x \sqcup \sqcup (TC \text{ ' } elts x) \sqcup lift x y \sqcup \sqcup ((\lambda u. TC x \sqcup lift x (TC u))$
 $\text{ ' } elts y)$
by (*simp add: sup-aci step.IH*)
also have $\dots = TC x \sqcup lift x y \sqcup \sqcup ((\lambda u. lift x (TC u)) \text{ ' } elts y)$
by (*simp add: sup-aci SUP-sup-distrib flip: TC [of x]*)
also have $\dots = TC x \sqcup lift x (y \sqcup \sqcup (TC \text{ ' } elts y))$
by (*metis (no-types) elts-of-set lift-Sup-distrib image-image lift-sup-distrib re-*
placement small-elts sup-assoc)
also have $\dots = TC x \sqcup lift x (TC y)$
by (*simp add: TC [of y]*)
finally show *?case* .
qed

corollary *TC-add'*: $z \sqsubseteq x + y \iff z \sqsubseteq x \vee (\exists v. v \sqsubseteq y \wedge z = x + v)$
using *TC-add* **by** (*force simp: less-TC-def lift-def*)

Corollary 3.7

corollary *vcard-TC-add*: $vcard (TC (x+y)) = vcard (TC x) \oplus vcard (TC y)$
by (*simp add: TC-add TC-sup-lift card-lift vcard-disjoint-sup*)

Corollary 3.8

corollary *TC-lift*:
assumes $y \neq 0$
shows $TC (lift x y) = TC x \sqcup lift x (TC y)$

proof –
have $TC (lift\ x\ y) = lift\ x\ y \sqcup \bigsqcup ((\lambda u. TC(x+u)) \text{ ` } elts\ y)$
unfolding $TC [of\ lift\ x\ y]$ **by** (*simp add: lift-def image-image*)
also have $\dots = lift\ x\ y \sqcup (\bigsqcup u \in elts\ y. TC\ x \sqcup lift\ x\ (TC\ u))$
by (*simp add: TC-add*)
also have $\dots = lift\ x\ y \sqcup TC\ x \sqcup (\bigsqcup u \in elts\ y. lift\ x\ (TC\ u))$
using *assms* **by** (*auto simp: SUP-sup-distrib*)
also have $\dots = TC\ x \sqcup lift\ x\ (TC\ y)$
by (*simp add: TC [of y] sup-aci image-image lift-sup-distrib lift-Sup-distrib*)
finally show *?thesis* .
qed

proposition *rank-add-distrib*: $rank\ (x+y) = rank\ x + rank\ y$

proof (*induction y rule: eps-induct*)

case (*step y*)

show *?case*

proof (*cases y=0*)

case *False*

then obtain e **where** $e: e \in elts\ y$

by *fastforce*

have $rank\ (x+y) = (\bigsqcup u \in elts\ (x \sqcup ZFC\text{-in-HOL.set}\ ((+)\ x \text{ ` } elts\ y)). succ\ (rank\ u))$

by (*metis plus rank-Sup*)

also have $\dots = (\bigsqcup x \in elts\ x. succ\ (rank\ x)) \sqcup (\bigsqcup z \in elts\ y. succ\ (rank\ x + rank\ z))$

apply (*simp add: Sup-Un-distrib image-Un image-image*)

apply (*simp add: step cong: SUP-cong-simp*)

done

also have $\dots = (\bigsqcup z \in elts\ y. rank\ x + succ\ (rank\ z))$

proof –

have $rank\ x \leq (\bigsqcup z \in elts\ y. ZFC\text{-in-HOL.succ}\ (rank\ x + rank\ z))$

using $\langle y \neq 0 \rangle$

by (*auto simp: plus-eq-lift intro: order-trans [OF - cSUP-upper [OF e]]*)

then show *?thesis*

by (*force simp: plus-V-succ-right simp flip: rank-Sup [of x] intro!: order-antisym*)

qed

also have $\dots = rank\ x + (\bigsqcup z \in elts\ y. succ\ (rank\ z))$

by (*simp add: add-Sup-distrib False*)

also have $\dots = rank\ x + rank\ y$

by (*simp add: rank-Sup [of y]*)

finally show *?thesis* .

qed *auto*

qed

lemma *Ord-add [simp]*: $[[Ord\ x; Ord\ y]] \implies Ord\ (x+y)$

proof (*induction y rule: eps-induct*)

case (*step y*)

then show *?case*

by (metis Ord-rank rank-add-distrib rank-of-Ord)
qed

lemma *add-Sup-distrib-id*: $A \neq 0 \implies x + \bigsqcup(\text{elts } A) = (\bigsqcup_{z \in \text{elts } A} x + z)$
by (metis add-Sup-distrib image-ident image-image)

lemma *add-Limit*: $\text{Limit } \alpha \implies x + \alpha = (\bigsqcup_{z \in \text{elts } \alpha} x + z)$
by (metis Limit-add-Sup-distrib Limit-eq-Sup-self image-ident image-image)

lemma *add-le-left*:
assumes *Ord* α *Ord* β **shows** $\beta \leq \alpha + \beta$
using $\langle \text{Ord } \beta \rangle$
proof (*induction rule: Ord-induct3*)
case 0
then show ?case
by auto
next
case (*succ* α)
then show ?case
by (auto simp: plus-V-succ-right Ord-mem-iff-lt assms(1))
next
case (*Limit* μ)
then have $k: \mu = (\bigsqcup \beta \in \text{elts } \mu. \beta)$
by (simp add: Limit-eq-Sup-self)
also have $\dots \leq (\bigsqcup \beta \in \text{elts } \mu. \alpha + \beta)$
using *Limit.IH* by auto
also have $\dots = \alpha + (\bigsqcup \beta \in \text{elts } \mu. \beta)$
using *Limit.hyps* *Limit-add-Sup-distrib* by presburger
finally show ?case
using k by simp
qed

lemma *plus- ω -equals- ω* :
assumes $\alpha \in \text{elts } \omega$ **shows** $\alpha + \omega = \omega$
proof (*rule antisym*)
show $\alpha + \omega \leq \omega$
using *Ord-trans assms* by (auto simp: elim!: mem-plus-V-E)
show $\omega \leq \alpha + \omega$
by (simp add: add-le-left assms)
qed

lemma *one-plus- ω -equals- ω* [*simp*]: $1 + \omega = \omega$
by (simp add: one-V-def plus- ω -equals- ω)

3.1.3 Cancellation / set subtraction

definition *vle* :: $V \Rightarrow V \Rightarrow \text{bool}$ (*infix* $\langle \triangleleft \rangle$ 50)
where $x \triangleleft y \equiv \exists z::V. x+z = y$

lemma *vle-refl* [*iff*]: $x \leq x$
by (*metis* (*no-types*) *add.right-neutral vle-def*)

lemma *vle-antisym*: $\llbracket x \leq y; y \leq x \rrbracket \implies x = y$
by (*metis* *V-equalityI plus-eq-lift sup-ge1 vle-def vsubsetD*)

lemma *vle-trans* [*trans*]: $\llbracket x \leq y; y \leq z \rrbracket \implies x \leq z$
by (*metis* *add.assoc vle-def*)

definition *vle-comparable* :: $V \Rightarrow V \Rightarrow \text{bool}$
where *vle-comparable* $x\ y \equiv x \leq y \vee y \leq x$

Lemma 3.13

lemma *comparable*:
assumes $a+b = c+d$
shows *vle-comparable* $a\ c$
unfolding *vle-comparable-def*
proof (*rule ccontr*)
assume *non*: $\neg (a \leq c \vee c \leq a)$
let $?\varphi = \lambda x. \forall z. a+x \neq c+z$
have $?\varphi\ x$ **for** x
proof (*induction x rule: eps-induct*)
case (*step x*)
show $?\text{case}$
proof (*cases x=0*)
case *True*
with *non nonzero-less-TC* **show** $?\text{thesis}$
using *vle-def* **by** *auto*
next
case *False*
then obtain v **where** $v \in \text{elts } x$
using *trad-foundation* **by** *blast*
show $?\text{thesis}$
proof *clarsimp*
fix z
assume *eq*: $a + x = c + z$
then have $z \neq 0$
using *vle-def non* **by** *auto*
have $av: a+v \in \text{elts } (a+x)$
by (*simp add: v ∈ elts x*)
moreover have $a+x = c \sqcup \text{lift } c\ z$
using *eq plus-eq-lift* **by** *fastforce*
ultimately have $a+v \in \text{elts } (c \sqcup \text{lift } c\ z)$
by *simp*
moreover
define u **where** $u \equiv \text{set } (\text{elts } x - \{v\})$
have $u: v \notin \text{elts } u$ **and** $x \text{eq}: x = \text{vinsert } v\ u$
using $\langle v \in \text{elts } x \rangle$ **by** (*auto simp: u-def intro: order-antisym*)
have *case1*: $a+v \notin \text{elts } c$

```

proof
  assume avc:  $a + v \in \text{elts } c$ 
  then have  $a \leq c$ 
    by clarify (metis Un-iff elts-sup-iff eq mem-not-sym mem-plus-V-E
plus-eq-lift)
  moreover have  $a \sqcup \text{lift } a \ x = c \sqcup \text{lift } c \ z$ 
    using eq by (simp add: plus-eq-lift)
  ultimately have  $\text{lift } c \ z \leq \text{lift } a \ x$ 
    by (metis inf.absorb-iff2 inf-commute inf-sup-absorb inf-sup-distrib2
lift-self-disjoint sup.commute)
  also have  $\dots = \text{vinsert } (a+v) (\text{lift } a \ u)$ 
    by (simp add: lift-def vinsert-def xeq)
  finally have  $*$ :  $\text{lift } c \ z \leq \text{vinsert } (a + v) (\text{lift } a \ u)$  .
  have  $\text{lift } c \ z \leq \text{lift } a \ u$ 
  proof -
    have  $a + v \notin \text{elts } (\text{lift } c \ z)$ 
      using lift-self-disjoint [of c z] avc V-disjoint-iff by auto
    then show ?thesis
      using  $*$  less-eq-V-def by auto
  qed
  { fix e
    assume  $e \in \text{elts } z$ 
    then have  $c+e \in \text{elts } (\text{lift } c \ z)$ 
      by (simp add: lift-def)
    then have  $c+e \in \text{elts } (\text{lift } a \ u)$ 
      using  $\langle \text{lift } c \ z \leq \text{lift } a \ u \rangle$  by blast
    then obtain y where  $y \in \text{elts } u \ c+e = a+y$ 
      using lift-def by auto
    then have False
      by (metis elts-vinsert insert-iff step.IH xeq)
  }
  then show False
    using  $\langle z \neq 0 \rangle$  by fastforce
  qed
  ultimately show False
    by (metis (no-types) \langle v \in \text{elts } x \rangle av case1 eq mem-plus-V-E step.IH)
  qed
  qed
  then show False
    using assms by blast
  qed

```

```

lemma vle1:  $x \triangleleft y \implies x \leq y$ 
  using vle-def plus-eq-lift by auto

```

```

lemma vle2:  $x \triangleleft y \implies x \sqsubseteq y$ 
  by (metis (full-types) TC-add' add.right-neutral le-TC-def vle-def nonzero-less-TC)

```

lemma *vle-iff-le-Ord*:
assumes *Ord* α *Ord* β
shows $\alpha \sqsubseteq \beta \iff \alpha \leq \beta$
proof
show $\alpha \leq \beta$ **if** $\alpha \sqsubseteq \beta$
using *that* **by** (*simp add: vle1*)
show $\alpha \sqsubseteq \beta$ **if** $\alpha \leq \beta$
using $\langle \text{Ord } \alpha \rangle \langle \text{Ord } \beta \rangle$ *that*
proof (*induction* α *arbitrary:* β *rule: Ord-induct*)
case (*step* γ)
then show *?case*
unfolding *vle-def*
by (*metis Ord-add Ord-linear add-le-left mem-not-refl mem-plus-V-E vsubsetD*)
qed
qed

lemma *add-le-cancel-left0 [iff]*:
fixes $x::V$ **shows** $x \leq x+z$
by (*simp add: vle1 vle-def*)

lemma *add-less-cancel-left0 [iff]*:
fixes $x::V$ **shows** $x < x+z \iff 0 < z$
by (*metis add-less-cancel-left add.right-neutral*)

lemma *le-Ord-diff*:
assumes $\alpha \leq \beta$ *Ord* α *Ord* β
obtains γ **where** $\alpha+\gamma = \beta$ $\gamma \leq \beta$ *Ord* γ
proof –
obtain γ **where** $\gamma: \alpha+\gamma = \beta$ $\gamma \leq \beta$
by (*metis add-le-cancel-left add-le-left assms vle-def vle-iff-le-Ord*)
then have *Ord* γ
using *Ord-def Transset-def* $\langle \text{Ord } \beta \rangle$ **by force**
with γ **that show** *thesis* **by blast**
qed

lemma *plus-Ord-le*:
assumes $\alpha \in \text{elts } \omega$ *Ord* β **shows** $\alpha+\beta \leq \beta+\alpha$
proof (*cases* $\beta \in \text{elts } \omega$)
case *True*
with *assms* **have** $\alpha+\beta = \beta+\alpha$
by (*auto simp: elts- ω add commute ord-of-nat-add [symmetric]*)
then show *?thesis* **by simp**
next
case *False*
then have $\omega \leq \beta$
using *Ord-linear2 Ord-mem-iff-lt* $\langle \text{Ord } \beta \rangle$ **by auto**
then obtain γ **where** $\omega+\gamma = \beta$ $\gamma \leq \beta$ *Ord* γ
using $\langle \text{Ord } \beta \rangle$ *le-Ord-diff* **by auto**
then have $\alpha+\beta = \beta$

```

    by (metis add.assoc assms(1) plus- $\omega$ -equals- $\omega$ )
  then show ?thesis
    by simp
qed

lemma add-right-mono:  $[\alpha \leq \beta; \text{Ord } \alpha; \text{Ord } \beta; \text{Ord } \gamma] \implies \alpha + \gamma \leq \beta + \gamma$ 
  by (metis add-le-cancel-left add.assoc add-le-left le-Ord-diff)

lemma add-strict-mono:  $[\alpha < \beta; \gamma < \delta; \text{Ord } \alpha; \text{Ord } \beta; \text{Ord } \gamma; \text{Ord } \delta] \implies \alpha + \gamma < \beta + \delta$ 
  by (metis order.strict-implies-order add-less-cancel-left add-right-mono le-less-trans)

lemma add-right-strict-mono:  $[\alpha \leq \beta; \gamma < \delta; \text{Ord } \alpha; \text{Ord } \beta; \text{Ord } \gamma; \text{Ord } \delta] \implies \alpha + \gamma < \beta + \delta$ 
  using add-strict-mono le-imp-less-or-eq by blast

lemma Limit-add-Limit [simp]:
  assumes Limit  $\mu$  Ord  $\beta$  shows Limit  $(\beta + \mu)$ 
  unfolding Limit-def
  proof (intro conjI allI impI)
    show Ord  $(\beta + \mu)$ 
      using Limit-def assms by auto
    show  $0 \in \text{elts } (\beta + \mu)$ 
      using Limit-def add-le-left assms by auto
  next
  fix  $\gamma$ 
  assume  $\gamma \in \text{elts } (\beta + \mu)$ 
  then consider  $\gamma \in \text{elts } \beta \mid \xi$  where  $\xi \in \text{elts } \mu$   $\gamma = \beta + \xi$ 
    using mem-plus-V-E by blast
  then show  $\text{succ } \gamma \in \text{elts } (\beta + \mu)$ 
  proof cases
    case 1
    then show ?thesis
      by (metis Kirby.add-strict-mono Limit-def Ord-add Ord-in-Ord Ord-mem-iff-lt
        assms one-V-def succ-eq-add1)
    next
    case 2
    then show ?thesis
      by (metis Limit-def add-mem-right-cancel assms(1) plus-V-succ-right)
  qed
qed

```

3.2 Generalised Difference

definition *odiff* where $\text{odiff } y \ x \equiv \text{THE } z::V. (x+z = y) \vee (z=0 \wedge \neg x \leq y)$

lemma *vle-imp-odiff-eq*: $x \leq y \implies x + (\text{odiff } y \ x) = y$
 by (auto simp: vle-def odiff-def)

lemma *not-vle-imp-odiff-0*: $\neg x \leq y \implies (\text{odiff } y \ x) = 0$
by (*auto simp: vle-def odiff-def*)

lemma *Ord-odiff-eq*:
assumes $\alpha \leq \beta$ *Ord* α *Ord* β
shows $\alpha + \text{odiff } \beta \ \alpha = \beta$
by (*simp add: assms vle-iff-le-Ord vle-imp-odiff-eq*)

lemma *Ord-odiff*:
assumes *Ord* α *Ord* β **shows** *Ord* ($\text{odiff } \beta \ \alpha$)
proof (*cases* $\alpha \leq \beta$)
case *True*
then show *?thesis*
by (*metis add-right-cancel assms le-Ord-diff vle1 vle-imp-odiff-eq*)
next
case *False*
then show *?thesis*
by (*simp add: odiff-def vle-def*)
qed

lemma *Ord-odiff-le*:
assumes *Ord* α *Ord* β **shows** $\text{odiff } \beta \ \alpha \leq \beta$
proof (*cases* $\alpha \leq \beta$)
case *True*
then show *?thesis*
by (*metis add-right-cancel assms le-Ord-diff vle1 vle-imp-odiff-eq*)
next
case *False*
then show *?thesis*
by (*simp add: odiff-def vle-def*)
qed

lemma *odiff-0-right* [*simp*]: $\text{odiff } x \ 0 = x$
by (*metis add.left-neutral vle-def vle-imp-odiff-eq*)

lemma *odiff-succ*: $y \leq x \implies \text{odiff } (\text{succ } x) \ y = \text{succ } (\text{odiff } x \ y)$
unfolding *odiff-def*
by (*metis add-right-cancel odiff-def plus-V-succ-right vle-def vle-imp-odiff-eq*)

lemma *odiff-eq-iff*: $z \leq x \implies \text{odiff } x \ z = y \longleftrightarrow x = z + y$
by (*auto simp: odiff-def vle-def*)

lemma *odiff-le-iff*: $z \leq x \implies \text{odiff } x \ z \leq y \longleftrightarrow x \leq z + y$
by (*auto simp: odiff-def vle-def*)

lemma *odiff-less-iff*: $z \leq x \implies \text{odiff } x \ z < y \longleftrightarrow x < z + y$
by (*auto simp: odiff-def vle-def*)

lemma *odiff-ge-iff*: $z \leq x \implies \text{odiff } x z \geq y \iff x \geq z + y$
by (*auto simp: odiff-def vle-def*)

lemma *Ord-odiff-le-iff*: $[\alpha \leq x; \text{Ord } x; \text{Ord } \alpha] \implies \text{odiff } x \alpha \leq y \iff x \leq \alpha + y$
by (*simp add: odiff-le-iff vle-iff-le-Ord*)

lemma *odiff-le-odiff*:
assumes $x \leq y$ **shows** $\text{odiff } x z \leq \text{odiff } y z$
proof (*cases z \leq x*)
case *True*
then show *?thesis*
using *assms odiff-le-iff vle1 vle-imp-odiff-eq vle-trans* **by** *presburger*
next
case *False*
then show *?thesis*
by (*simp add: not-vle-imp-odiff-0*)
qed

lemma *Ord-odiff-le-odiff*: $[\alpha \leq y; \text{Ord } x; \text{Ord } y] \implies \text{odiff } x \alpha \leq \text{odiff } y \alpha$
by (*simp add: odiff-le-odiff vle-iff-le-Ord*)

lemma *Ord-odiff-less-odiff*: $[\alpha \leq x; x < y; \text{Ord } x; \text{Ord } y; \text{Ord } \alpha] \implies \text{odiff } x \alpha < \text{odiff } y \alpha$
by (*metis Ord-odiff-eq Ord-odiff-le-odiff dual-order.strict-trans less-V-def*)

lemma *Ord-odiff-less-imp-less*: $[\text{odiff } x \alpha < \text{odiff } y \alpha; \text{Ord } x; \text{Ord } y] \implies x < y$
by (*meson Ord-linear2 leD odiff-le-odiff vle-iff-le-Ord*)

lemma *odiff-add-cancel* [*simp*]: $\text{odiff } (x + y) x = y$
by (*simp add: odiff-eq-iff vle-def*)

lemma *odiff-add-cancel-0* [*simp*]: $\text{odiff } x x = 0$
by (*simp add: odiff-eq-iff*)

lemma *odiff-add-cancel-both* [*simp*]: $\text{odiff } (x + y) (x + z) = \text{odiff } y z$
by (*simp add: add.assoc odiff-def vle-def*)

3.3 Generalised Multiplication

Credited to Dana Scott

instantiation *V :: times*
begin

This definition is credited to Tarski

definition *times-V* :: $V \Rightarrow V \Rightarrow V$
where *times-V* $x \equiv \text{transrec } (\lambda f y. \sqcup ((\lambda u. \text{lift } (f u) x) \text{ `elts } y))$

instance ..
end

lemma *mult*: $x * y = (\sqcup_{u \in \text{elts } y} \text{lift } (x * u) x)$
unfolding *times-V-def* **by** (*subst transrec*) (*force simp*)

lemma *elts-multE*:
assumes $z \in \text{elts } (x * y)$
obtains $u v$ **where** $u \in \text{elts } x \ v \in \text{elts } y \ z = x*v + u$
using *mult [of x y] lift-def assms* **by** *auto*

Lemma 4.2

lemma *mult-zero-right* [*simp*]:
fixes $x::V$ **shows** $x * 0 = 0$
by (*metis ZFC-in-HOL.Sup-empty elts-0 image-empty mult*)

lemma *mult-insert*: $x * (\text{vinsert } y z) = x*z \sqcup \text{lift } (x*y) x$
by (*metis (no-types, lifting) elts-vinsert image-insert replacement small-elts sup-commute mult Sup-V-insert*)

lemma *mult-succ*: $x * \text{succ } y = x*y + x$
by (*simp add: mult-insert plus-eq-lift succ-def*)

lemma *ord-of-nat-mult*: $\text{ord-of-nat } (m*n) = \text{ord-of-nat } m * \text{ord-of-nat } n$
proof (*induction n*)
case (*Suc n*)
then show *?case*
by (*simp add: add.commute [of m]*) (*simp add: ord-of-nat-add mult-succ*)
qed *auto*

lemma *omega-closed-mult* [*intro*]:
assumes $\alpha \in \text{elts } \omega \ \beta \in \text{elts } \omega$ **shows** $\alpha*\beta \in \text{elts } \omega$
proof –
obtain $m n$ **where** $\alpha = \text{ord-of-nat } m \ \beta = \text{ord-of-nat } n$
using *assms elts- ω* **by** *auto*
then have $\alpha*\beta = \text{ord-of-nat } (m*n)$
by (*simp add: ord-of-nat-mult*)
then show *?thesis*
by (*simp add: ω -def*)
qed

lemma *zero-imp-le-mult*: $0 \in \text{elts } y \implies x \leq x*y$
by (*auto simp: mult [of x y]*)

3.3.1 Proposition 4.3

lemma *mult-zero-left* [*simp*]:
fixes $x::V$ **shows** $0 * x = 0$
proof (*induction x rule: eps-induct*)
case (*step x*)
then show *?case*

by (subst mult) auto
qed

lemma *mult-sup-distrib*:

fixes $x::V$ shows $x * (y \sqcup z) = x*y \sqcup x*z$
unfolding *mult [of x y \sqcup z] mult [of x y] mult [of x z]*
by (*simp add: Sup-Un-distrib image-Un*)

lemma *mult-Sup-distrib*: $\text{small } Y \implies x * (\bigsqcup Y) = \bigsqcup ((*) x \text{ ' } Y)$ **for** $Y::V \text{ set}$

unfolding *mult [of x \sqcup Y]*
by (*simp add: cSUP-UNION (metis mult)*)

lemma *mult-lift-imp-distrib*: $x * (\text{lift } y \ z) = \text{lift } (x*y) \ (x*z) \implies x * (y+z) = x*y$
 $+ x*z$

by (*simp add: mult-sup-distrib plus-eq-lift*)

lemma *mult-lift*: $x * (\text{lift } y \ z) = \text{lift } (x*y) \ (x*z)$

proof (*induction z rule: eps-induct*)

case (*step z*)

have $x * \text{lift } y \ z = (\bigsqcup u \in \text{elts } (\text{lift } y \ z). \text{lift } (x * u) \ x)$
using *mult by blast*

also have $\dots = (\bigsqcup v \in \text{elts } z. \text{lift } (x * (y + v)) \ x)$
using *lift-def by auto*

also have $\dots = (\bigsqcup v \in \text{elts } z. \text{lift } (x * y + x * v) \ x)$
using *mult-lift-imp-distrib step.IH by auto*

also have $\dots = (\bigsqcup v \in \text{elts } z. \text{lift } (x * y) \ (\text{lift } (x * v) \ x))$
by (*simp add: lift-lift*)

also have $\dots = \text{lift } (x * y) \ (\bigsqcup v \in \text{elts } z. \text{lift } (x * v) \ x)$
by (*simp add: image-image lift-Sup-distrib*)

also have $\dots = \text{lift } (x*y) \ (x*z)$
by (*metis mult*)

finally show *?case .*

qed

lemma *mult-Limit*: $\text{Limit } \gamma \implies x * \gamma = \bigsqcup ((*) x \text{ ' } \text{elts } \gamma)$

by (*metis Limit-eq-Sup-self mult-Sup-distrib small-elts*)

lemma *add-mult-distrib*: $x * (y+z) = x*y + x*z$ **for** $x::V$

by (*simp add: mult-lift mult-lift-imp-distrib*)

instantiation $V :: \text{monoid-mult}$

begin

instance

proof

show $1 * x = x$ **for** $x::V$

proof (*induction x rule: eps-induct*)

case (*step x*)

then show *?case*

by (*subst mult*) *auto*

```

qed
show  $x * 1 = x$  for  $x :: V$ 
  by (subst mult) auto
show  $(x * y) * z = x * (y * z)$  for  $x y z :: V$ 
proof (induction z rule: eps-induct)
  case (step z)
  have  $(x * y) * z = (\bigsqcup u \in \text{elts } z. \text{lift } (x * y * u) (x * y))$ 
    using mult by blast
  also have  $\dots = (\bigsqcup u \in \text{elts } z. \text{lift } (x * (y * u)) (x * y))$ 
    using step.IH by auto
  also have  $\dots = (\bigsqcup u \in \text{elts } z. x * \text{lift } (y * u) y)$ 
    using mult-lift by auto
  also have  $\dots = x * (\bigsqcup u \in \text{elts } z. \text{lift } (y * u) y)$ 
    by (simp add: image-image mult-Sup-distrib)
  also have  $\dots = x * (y * z)$ 
    by (metis mult)
  finally show ?case .
qed
qed

end

lemma le-mult:
  assumes Ord  $\beta$   $\beta \neq 0$  shows  $\alpha \leq \alpha * \beta$ 
  using assms
proof (induction rule: Ord-induct3)
  case (succ  $\alpha$ )
  then show ?case
    using mult-insert succ-def by fastforce
next
  case (Limit  $\mu$ )
  have  $\alpha \in (*) \alpha \text{ 'elts } \mu$ 
    using Limit.hyps Limit-def one-V-def by (metis imageI mult.right-neutral)
  then have  $\alpha \leq \bigsqcup ((*) \alpha \text{ 'elts } \mu)$ 
    by auto
  then show ?case
    by (simp add: Limit.hyps mult-Limit)
qed auto

lemma mult-sing-1 [simp]:
  fixes  $x :: V$  shows  $x * \text{set}\{1\} = \text{lift } x x$ 
  by (subst mult) auto

lemma mult-2-right [simp]:
  fixes  $x :: V$  shows  $x * \text{set}\{0,1\} = x + x$ 
  by (subst mult) (auto simp: Sup-V-insert plus-eq-lift)

lemma Ord-mult [simp]:  $[[\text{Ord } y; \text{Ord } x]] \implies \text{Ord } (x * y)$ 
proof (induction y rule: Ord-induct3)

```

```

case 0
then show ?case
  by auto
next
case (succ k)
then show ?case
  by (simp add: mult-succ)
next
case (Limit k)
then have Ord (x *  $\sqcup$  (elts k))
  by (metis Ord-Sup imageE mult-Sup-distrib small-elts)
then show ?case
  using Limit.hyps Limit-eq-Sup-self by auto
qed

```

3.3.2 Proposition 4.4-5

proposition rank-mult-distrib: $\text{rank } (x*y) = \text{rank } x * \text{rank } y$

proof (induction y rule: eps-induct)

```

case (step y)
have rank (x*y) = ( $\sqcup$  y $\in$ elts ( $\sqcup$  u $\in$ elts y. lift (x * u) x). succ (rank y))
  by (metis rank-Sup mult)
also have ... = ( $\sqcup$  u $\in$ elts y.  $\sqcup$  r $\in$ elts x. succ (rank (x * u + r)))
  apply (simp add: lift-def image-image image-UN)
  apply (simp add: Sup-V-def)
done
also have ... = ( $\sqcup$  u $\in$ elts y.  $\sqcup$  r $\in$ elts x. succ (rank (x * u) + rank r))
  using rank-add-distrib by auto
also have ... = ( $\sqcup$  u $\in$ elts y.  $\sqcup$  r $\in$ elts x. succ (rank x * rank u + rank r))
  using step arg-cong [where f = Sup] by auto
also have ... = ( $\sqcup$  u $\in$ elts y. rank x * rank u + rank x)
proof (rule SUP-cong)
  show ( $\sqcup$  r $\in$ elts x. succ (rank x * rank u + rank r)) = rank x * rank u + rank
x
  if u  $\in$  elts y for u
  proof (cases x=0)
  case False
  have ( $\sqcup$  r $\in$ elts x. succ (rank x * rank u + rank r)) = rank x * rank u +
( $\sqcup$  y $\in$ elts x. succ (rank y))
  proof (rule order-antisym)
  show ( $\sqcup$  r $\in$ elts x. succ (rank x * rank u + rank r))  $\leq$  rank x * rank u +
( $\sqcup$  y $\in$ elts x. succ (rank y))
  by (auto simp: Sup-le-iff simp flip: plus-V-succ-right)
  have rank x * rank u + ( $\sqcup$  y $\in$ elts x. succ (rank y)) = ( $\sqcup$  y $\in$ elts x. rank x *
rank u + succ (rank y))
  by (simp add: add-Sup-distrib False)
also have ...  $\leq$  ( $\sqcup$  r $\in$ elts x. succ (rank x * rank u + rank r))
  using plus-V-succ-right by auto
  finally show rank x * rank u + ( $\sqcup$  y $\in$ elts x. succ (rank y))  $\leq$  ( $\sqcup$  r $\in$ elts x.

```

```

succ (rank x * rank u + rank r)) .
  qed
  also have ... = rank x * rank u + rank x
    by (metis rank-Sup)
  finally show ?thesis .
  qed auto
  qed auto
  also have ... = rank x * rank y
    by (simp add: rank-Sup [of y] mult-Sup-distrib mult-succ image-image)
  finally show ?case .
qed

```

lemma *mult-le1*:

```

fixes y::V assumes y ≠ 0 shows x ⊆ x * y
proof (cases x = 0)
  case False
  then obtain r where r: r ∈ elts x
    by fastforce
  from ⟨y ≠ 0⟩ show ?thesis
  proof (induction y rule: eps-induct)
    case (step y)
    show ?case
    proof (cases y = 1)
      case False
      with ⟨y ≠ 0⟩ obtain p where p: p ∈ elts y p ≠ 0
        by (metis V-equalityI elts-1 insertI1 singletonD trad-foundation)
      then have x*p + r ∈ elts (lift (x*p) x)
        by (simp add: lift-def r)
      moreover have lift (x*p) x ≤ x*y
        by (metis bdd-above-iff-small cSUP-upper2 order-refl ⟨p ∈ elts y⟩ replacement
small-elts mult)
      ultimately have x*p + r ∈ elts (x*y)
        by blast
      moreover have x*p ⊆ x*p + r
        by (metis TC-add' V-equalityI add.right-neutral eps-induct le-TC-refl
less-TC-iff less-imp-le-TC)
      ultimately show ?thesis
        using step.IH [OF p] le-TC-trans less-TC-iff by blast
    qed auto
  qed
  qed auto

```

lemma *mult-eq-0-iff* [simp]:

```

fixes y::V shows x * y = 0 ⟷ x=0 ∨ y=0
proof
  show x = 0 ∨ y = 0 if x * y = 0
    by (metis le-0 le-TC-def less-TC-imp-not-le mult-le1 that)
  qed auto

```

lemma *lift-lemma*:

assumes $x \neq 0 \ y \neq 0$ **shows** $\neg \text{lift } (x * y) \ x \leq x$

using *assms mult-le1* [of concl: $x \ y$]

by (*auto simp: le-TC-def TC-lift less-TC-def less-TC-imp-not-le*)

lemma *mult-le2*:

fixes $y::V$ **assumes** $x \neq 0 \ y \neq 0 \ y \neq 1$ **shows** $x \sqsubset x * y$

proof –

obtain v **where** $v: v \in \text{elts } y \ v \neq 0$

using *assms* **by** *fastforce*

have $x \neq x * y$

using *lift-lemma* [of $x \ v$]

by (*metis* $\langle x \neq 0 \rangle$ *bdd-above-iff-small cSUP-upper2 order-refl replacement small-elts mult v*)

then show *?thesis*

using *assms mult-le1* [of $y \ x$]

by (*auto simp: le-TC-def*)

qed

lemma *elts-mult- ωE* :

assumes $x \in \text{elts } (y * \omega)$

obtains n **where** $n \neq 0 \ x \in \text{elts } (y * \text{ord-of-nat } n) \wedge m. m < n \implies x \notin \text{elts } (y * \text{ord-of-nat } m)$

proof –

obtain k **where** $k: k \neq 0 \wedge x \in \text{elts } (y * \text{ord-of-nat } k)$

using *assms*

apply (*simp add: mult-Limit elts- ω*)

by (*metis mult-eq-0-iff elts-0 ex-in-conv ord-of-eq-0-iff that*)

define n **where** $n \equiv (\text{LEAST } k. k \neq 0 \wedge x \in \text{elts } (y * \text{ord-of-nat } k))$

show *thesis*

proof

show $n \neq 0 \ x \in \text{elts } (y * \text{ord-of-nat } n)$

unfolding *n-def* **by** (*metis (mono-tags, lifting) LeastI-ex k*)**+**

show $\wedge m. m < n \implies x \notin \text{elts } (y * \text{ord-of-nat } m)$

by (*metis (mono-tags, lifting) mult-eq-0-iff elts-0 empty-iff n-def not-less-Least ord-of-eq-0-iff*)

qed

qed

3.3.3 Theorem 4.6

theorem *mult-eq-imp-0*:

assumes $a * x = a * y + b \ b \sqsubset a$

shows $b = 0$

proof (*cases a=0 \vee x=0*)

case *True*

with *assms* **show** *?thesis*

by (*metis add-le-cancel-left mult-eq-0-iff eq-iff le-0*)

next

```

case False
then have  $a \neq 0$   $x \neq 0$ 
  by auto
then show ?thesis
proof (cases  $y=0$ )
  case True
  then show ?thesis
    using assms less-asym-TC mult-le2 by force
next
case False
have  $b=0$  if  $\text{Ord } \alpha$   $x \in \text{elts } (Vset \ \alpha)$   $y \in \text{elts } (Vset \ \alpha)$  for  $\alpha$ 
  using that assms
proof (induction  $\alpha$  arbitrary:  $x$   $y$   $b$  rule: Ord-induct3)
  case 0
  then show ?case by auto
next
case (succ  $k$ )
define  $\Phi$  where  $\Phi \equiv \lambda x y. \exists r. 0 \sqsubset r \wedge r \sqsubset a \wedge a*x = a*y + r$ 
show ?case
proof (rule ccontr)
  assume  $b \neq 0$ 
  then have  $0 \sqsubset b$ 
    by (metis nonzero-less-TC)
  then have  $\Phi \ x \ y$ 
    unfolding  $\Phi$ -def using succ.premis by blast
  then obtain  $x'$  where  $\Phi \ x' \ y \ x' \sqsubseteq x$  and  $\text{min: } \bigwedge x''. x'' \sqsubset x' \implies \neg \Phi \ x'' \ y$ 
    using less-TC-minimal [of  $\lambda x. \Phi \ x \ y \ x$ ] by blast
  then obtain  $b'$  where  $0 \sqsubset b' \ b' \sqsubset a$  and  $\text{eq: } a*x' = a*y + b'$ 
    using  $\Phi$ -def by blast
  have  $a*y \sqsubset a*x'$ 
    using TC-add'  $\langle 0 \sqsubset b' \rangle$  eq by auto
  then obtain  $p$  where  $p \in \text{elts } (a * x')$   $a * y \sqsubseteq p$ 
    using less-TC-iff by blast
  then have  $p \notin \text{elts } (a * y)$ 
    using less-TC-iff less-irrefl-TC by blast
  then have  $p \in \bigcup (\text{elts } \langle \lambda v. \text{lift } (a * v) \ a \rangle \text{ elts } x')$ 
    by (metis  $\langle p \in \text{elts } (a * x') \rangle$  elts-Sup replacement small-elts mult)
  then obtain  $u \ c$  where  $u \in \text{elts } x' \ c \in \text{elts } a \ p = a*u + c$ 
    using lift-def by auto
  then have  $p \in \text{elts } (\text{lift } (a*y) \ b')$ 
    using  $\langle p \in \text{elts } (a * x') \rangle \langle p \notin \text{elts } (a * y) \rangle$  eq plus-eq-lift by auto
  then obtain  $d$  where  $d: d \in \text{elts } b' \ p = a*y + d \ p = a*u + c$ 
    by (metis  $\langle p = a * u + c \rangle \langle p \in \text{elts } (a * x') \rangle \langle p \notin \text{elts } (a * y) \rangle$  eq
mem-plus-V-E)
  have noteq:  $a*y \neq a*u$ 
  proof
    assume  $a*y = a*u$ 
    then have  $\text{lift } (a*y) \ a = \text{lift } (a*u) \ a$ 
      by metis

```


also have $\dots \leq a*x'$
unfolding *mult [of - x]* **using** $\langle u \in \text{elts } x' \rangle$ **by** (*auto intro: cSUP-upper*)
also have $\dots = a*y \sqcup \text{lift } (a*y) b'$
by (*simp add: eq plus-eq-lift*)
finally have $\text{lift } (a*y) a \leq a*y \sqcup \text{lift } (a*y) b'$.
then have $\text{lift } (a*y) a \leq \text{lift } (a*y) b'$
using *add-le-cancel-left less-TC-imp-not-le plus-eq-lift* $\langle b' \sqsubset a \rangle$ **by** *auto*
then have $a \leq b'$
by (*simp add: le-iff-sup lift-eq-lift lift-sup-distrib*)
then show *False*
using $\langle b' \sqsubset a \rangle$ *less-TC-imp-not-le* **by** *auto*
qed
consider $a*y \triangleleft a*u \mid a*u \triangleleft a*y$
using *d comparable vle-comparable-def* **by** *auto*
then show *False*
proof cases
case 1
then obtain e **where** $e: a*u = a*y + e \ e \neq 0$
by (*metis add.right-neutral noteq vle-def*)
moreover have $e + c = d$
by (*metis e add-right-cancel* $\langle p = a * u + c \rangle \langle p = a * y + d \rangle$ *add.assoc*)
with $\langle d \in \text{elts } b' \rangle \langle b' \sqsubset a \rangle$ **have** $e \sqsubset a$
by (*meson less-TC-iff less-TC-trans vle2 vle-def*)
ultimately show *False*
— contradicts minimality of x'
using *min unfolding* Φ -def **by** (*meson* $\langle u \in \text{elts } x' \rangle$ *le-TC-def less-TC-iff nonzero-less-TC*)
next
case 2
then obtain e **where** $e: a*y = a*u + e \ e \neq 0$
by (*metis add.right-neutral noteq vle-def*)
moreover have $e + d = c$
by (*metis e add-right-cancel* $\langle p = a * u + c \rangle \langle p = a * y + d \rangle$ *add.assoc*)
with $\langle d \in \text{elts } b' \rangle \langle b' \sqsubset a \rangle$ **have** $e \sqsubset a$
by (*metis* $\langle c \in \text{elts } a \rangle$ *less-TC-iff vle2 vle-def*)
ultimately have $\Phi \ y \ u$
unfolding Φ -def **using** *nonzero-less-TC* **by** *blast*
then obtain y' **where** $\Phi \ y' \ u \ y' \sqsubseteq y$ **and** *min*: $\bigwedge x''. x'' \sqsubset y' \implies \neg \Phi$
 $x'' \ u$
using *less-TC-minimal* [of $\lambda x. \Phi \ x \ u \ y$] **by** *blast*
then obtain b' **where** $0 \sqsubset b' \ b' \sqsubset a$ **and** *eq*: $a*y' = a*u + b'$
using Φ -def **by** *blast*
have $u-k: u \in \text{elts } (Vset \ k)$
using $\langle u \in \text{elts } x' \rangle \langle x' \sqsubseteq x \rangle$ *succ Vset-succ-TC less-TC-iff less-le-TC-trans*
by *blast*
have $a*u \sqsubset a*y'$
using *TC-add'* $\langle 0 \sqsubset b' \rangle$ *eq* **by** *auto*
then obtain p **where** $p \in \text{elts } (a * y') \ a * u \sqsubseteq p$
using *less-TC-iff* **by** *blast*

then have $p \notin \text{elts } (a * u)$
using *less-TC-iff less-irrefl-TC* **by** *blast*
then have $p \in \bigcup (\text{elts } \langle \lambda v. \text{lift } (a * v) a \rangle \langle \text{elts } y' \rangle)$
by (*metis* $\langle p \in \text{elts } (a * y') \rangle$ *elts-Sup replacement small-elts mult*)
then obtain $v \ c$ **where** $v \in \text{elts } y' \ c \in \text{elts } a \ p = a*v + c$
using *lift-def* **by** *auto*
then have $p \in \text{elts } (\text{lift } (a*u) \ b')$
using $\langle p \in \text{elts } (a * y') \rangle \langle p \notin \text{elts } (a * u) \rangle$ *eq plus-eq-lift* **by** *auto*
then obtain d **where** $d: d \in \text{elts } b' \ p = a*u + d \ p = a*v + c$
by (*metis* $\langle p = a * v + c \rangle \langle p \in \text{elts } (a * y') \rangle \langle p \notin \text{elts } (a * u) \rangle$ *eq mem-plus-V-E*)
have $v\text{-}k: v \in \text{elts } (V\text{set } k)$
using *Vset-succ-TC* $\langle v \in \text{elts } y' \rangle \langle y' \sqsubseteq y \rangle$ *less-TC-iff less-le-TC-trans succ.hyps succ.premis(2)* **by** *blast*
have *noteq*: $a*u \neq a*v$
proof
assume $a*u = a*v$
then have $\text{lift } (a*v) \ a \leq a*y'$
unfolding *mult [of - y']* **using** $\langle v \in \text{elts } y' \rangle$ **by** (*auto intro: cSUP-upper*)
also have $\dots = a*u \sqcup \text{lift } (a*u) \ b'$
by (*simp add: eq plus-eq-lift*)
finally have $\text{lift } (a*v) \ a \leq a*u \sqcup \text{lift } (a*u) \ b'$.
then have $\text{lift } (a*u) \ a \leq \text{lift } (a*u) \ b'$
by (*metis* $\langle a * u = a * v \rangle$ *le-iff-sup lift-sup-distrib sup-left-commute sup-lift-eq-lift*)
then have $a \leq b'$
by (*simp add: le-iff-sup lift-eq-lift lift-sup-distrib*)
then show *False*
using $\langle b' \sqsubseteq a \rangle$ *less-TC-imp-not-le* **by** *auto*
qed
consider $a*u \trianglelefteq a*v \mid a*v \trianglelefteq a*u$
using *d comparable vle-comparable-def* **by** *auto*
then show *False*
proof cases
case 1
then obtain e **where** $e: a*v = a*u + e \ e \neq 0$
by (*metis add.right-neutral noteq vle-def*)
moreover have $e + c = d$
by (*metis add-right-cancel* $\langle p = a * u + d \rangle \langle p = a * v + c \rangle$ *add.assoc*)
e)
with $\langle d \in \text{elts } b' \rangle \langle b' \sqsubseteq a \rangle$ **have** $e \sqsubseteq a$
by (*meson less-TC-iff less-TC-trans vle2 vle-def*)
ultimately show *False*
using *succ.IH u-k v-k* **by** *blast*
next
case 2
then obtain e **where** $e: a*u = a*v + e \ e \neq 0$
by (*metis add.right-neutral noteq vle-def*)
moreover have $e + d = c$

```

      by (metis add-right-cancel add.assoc d e)
    with ⟨d ∈ elts b'⟩ ⟨b' ⊆ a⟩ have e ⊆ a
      by (metis ⟨c ∈ elts a⟩ less-TC-iff vle2 vle-def)
    ultimately show False
      using succ.IH u-k v-k by blast
  qed
qed
qed
next
case (Limit k)
obtain i j where k: i ∈ elts k j ∈ elts k
  and x: x ∈ elts (Vset i)
  and y: y ∈ elts (Vset j)
  using that Limit by (auto simp: Limit-Vfrom-eq)
show ?case
proof (rule Limit.IH [of i ⊔ j])
  show i ⊔ j ∈ elts k
    by (meson k x y Limit.hyps Limit-def Ord-in-Ord Ord-mem-iff-lt Ord-sup
union-less-iff)
  show x ∈ elts (Vset (i ⊔ j)) y ∈ elts (Vset (i ⊔ j))
    using x y by (auto simp: Vfrom-sup)
  qed (use Limit.prem1 in auto)
qed
then show ?thesis
  by (metis two-in-Vset Ord-rank Ord-VsetI rank-lt)
qed
qed

```

3.3.4 Theorem 4.7

lemma *mult-cancellation-half*:

assumes $a*x + r \leq a*y + s$ $r \subseteq a$ $s \subseteq a$

shows $x \leq y$

proof –

have $x \leq y$ if $\text{Ord } \alpha$ $x \in \text{elts } (Vset \alpha)$ $y \in \text{elts } (Vset \alpha)$ for α
 using that *assms*

proof (induction α arbitrary: x y r s rule: *Ord-induct3*)

case 0

then show ?case

by *auto*

next

case (*succ k*)

show ?case

proof

fix u

assume $u: u \in \text{elts } x$

have $u-k: u \in \text{elts } (Vset k)$

using *Vset-succ succ.hyps succ.prem1* u by *auto*

obtain r' where $r' \in \text{elts } a$ $r \subseteq r'$

```

    using less-TC-iff succ.prem(4) by blast
  have  $a * u + r' \in \text{elts } (\text{lift } (a * u) a)$ 
    by (simp add:  $\langle r' \in \text{elts } a \rangle$  lift-def)
  also have  $\dots \leq \text{elts } (a * x)$ 
    using u by (force simp: mult [of - x])
  also have  $\dots \leq \text{elts } (a * y + s)$ 
    using plus-eq-lift succ.prem(3) by auto
  also have  $\dots = \text{elts } (a * y) \cup \text{elts } (\text{lift } (a * y) s)$ 
    by (simp add: plus-eq-lift)
  finally have  $a * u + r' \in \text{elts } (a * y) \cup \text{elts } (\text{lift } (a * y) s)$  .
  then show  $u \in \text{elts } y$ 
  proof
    assume *:  $a * u + r' \in \text{elts } (a * y)$ 
    show  $u \in \text{elts } y$ 
    proof -
      obtain v e where  $v \in \text{elts } y$   $e \in \text{elts } a$   $a * u + r' = a * v + e$ 
        using * by (auto simp: mult [of - y] lift-def)
      then have v-k:  $v \in \text{elts } (Vset k)$ 
        using Vset-succ-TC less-TC-iff succ.prem(2) by blast
      then show ?thesis
        by (metis  $\langle r' \in \text{elts } a \rangle$  antisym le-TC-refl less-TC-iff order-refl succ.IH
          u-k v)
    qed
  next
    assume  $a * u + r' \in \text{elts } (\text{lift } (a * y) s)$ 
    then obtain t where  $t \in \text{elts } s$  and  $t: a * u + r' = a * y + t$ 
      using lift-def by auto
    have noteq:  $a * y \neq a * u$ 
    proof
      assume  $a * y = a * u$ 
      then have  $\text{lift } (a * y) a = \text{lift } (a * u) a$ 
        by metis
      also have  $\dots \leq a * x$ 
        unfolding mult [of - x] using  $\langle u \in \text{elts } x \rangle$  by (auto intro: cSUP-upper)
      also have  $\dots \leq a * y \sqcup \text{lift } (a * y) s$ 
        using  $\langle \text{elts } (a * x) \subseteq \text{elts } (a * y + s) \rangle$  plus-eq-lift by auto
      finally have  $\text{lift } (a * y) a \leq a * y \sqcup \text{lift } (a * y) s$  .
      then have  $\text{lift } (a * y) a \leq \text{lift } (a * y) s$ 
        using add-le-cancel-left less-TC-imp-not-le plus-eq-lift  $\langle s \sqsubseteq a \rangle$  by auto
      then have  $a \leq s$ 
        by (simp add: le-iff-sup lift-eq-lift lift-sup-distrib)
      then show False
        using  $\langle s \sqsubseteq a \rangle$  less-TC-imp-not-le by auto
    qed
  consider  $a * u \sqsubseteq a * y \mid a * y \sqsubseteq a * u$ 
    using t comparable vle-comparable-def by blast
  then have False
  proof cases
    case 1

```

```

then obtain  $c$  where  $a*y = a*u + c$ 
  by (metis vle-def)
then have  $c+t = r'$ 
  by (metis add-right-cancel add.assoc t)
then have  $c \sqsubset a$ 
  using  $\langle r' \in \text{elts } a \rangle$  less-TC-iff vle2 vle-def by force
moreover have  $c \neq 0$ 
  using  $\langle a * y = a * u + c \rangle$  noteq by auto
ultimately show ?thesis
  using  $\langle a * y = a * u + c \rangle$  mult-eq-imp-0 by blast
next
case 2
then obtain  $c$  where  $a*u = a*y + c$ 
  by (metis vle-def)
then have  $c+r' = t$ 
  by (metis add-right-cancel add.assoc t)
then have  $c \sqsubset a$ 
  by (metis \langle t \in \text{elts } s \rangle less-TC-iff less-TC-trans \langle s \sqsubset a \rangle vle2 vle-def)
moreover have  $c \neq 0$ 
  using  $\langle a * u = a * y + c \rangle$  noteq by auto
ultimately show ?thesis
  using  $\langle a * u = a * y + c \rangle$  mult-eq-imp-0 by blast
qed
then show  $u \in \text{elts } y$  ..
qed
qed
next
case (Limit k)
obtain  $i j$  where  $k: i \in \text{elts } k j \in \text{elts } k$ 
  and  $x: x \in \text{elts } (Vset\ i)$  and  $y: y \in \text{elts } (Vset\ j)$ 
  using that Limit by (auto simp: Limit-Vfrom-eq)
show ?case
proof (rule Limit.IH [of i \sqcup j])
  show  $i \sqcup j \in \text{elts } k$ 
  by (meson k x y Limit.hyps Limit-def Ord-in-Ord Ord-mem-iff-lt Ord-sup union-less-iff)
  show  $x \in \text{elts } (Vset\ (i \sqcup j))$   $y \in \text{elts } (Vset\ (i \sqcup j))$ 
  using  $x\ y$  by (auto simp: Vfrom-sup)
  show  $a * x + r \leq a * y + s$ 
  by (simp add: Limit.prem)
qed (auto simp: Limit.prem)
qed
then show ?thesis
  by (metis two-in-Vset Ord-rank Ord-VsetI rank-lt)
qed

theorem mult-cancellation-lemma:
assumes  $a*x + r = a*y + s$   $r \sqsubset a$   $s \sqsubset a$ 
shows  $x=y \wedge r=s$ 

```

by (metis assms leD less-V-def mult-cancellation-half odiff-add-cancel order-refl)

corollary *mult-cancellation* [simp]:

fixes $a::V$

assumes $a \neq 0$

shows $a*x = a*y \longleftrightarrow x=y$

by (metis assms nonzero-less-TC mult-cancellation-lemma)

corollary *mult-cancellation-less*:

assumes $lt: a*x + r < a*y + s$ and $r \sqsubset a \ s \sqsubset a$

obtains $x < y \mid x = y \ r < s$

proof –

have $x < y$

by (meson assms dual-order.strict-implies-order mult-cancellation-half)

then consider $x < y \mid x = y$

using *less-V-def* by blast

with lt that show ?thesis by blast

qed

corollary *lift-mult-TC-disjoint*:

fixes $x::V$

assumes $x \neq y$

shows $lift\ (a*x)\ (TC\ a) \sqcap lift\ (a*y)\ (TC\ a) = 0$

apply (rule *V-equalityI*)

using *assms*

by (auto simp: *less-TC-def inf-V-def lift-def image-iff dest: mult-cancellation-lemma*)

corollary *lift-mult-disjoint*:

fixes $x::V$

assumes $x \neq y$

shows $lift\ (a*x)\ a \sqcap lift\ (a*y)\ a = 0$

proof –

have $lift\ (a*x)\ a \sqcap lift\ (a*y)\ a \leq lift\ (a*x)\ (TC\ a) \sqcap lift\ (a*y)\ (TC\ a)$

by (metis *TC' inf-mono lift-sup-distrib sup-ge1*)

then show ?thesis

using *assms lift-mult-TC-disjoint* by auto

qed

lemma *mult-add-mem*:

assumes $a*x + r \in elts\ (a*y)\ r \sqsubset a$

shows $x \in elts\ y \ r \in elts\ a$

proof –

obtain $v\ s$ where $v: a * x + r = a * v + s \ v \in elts\ y \ s \in elts\ a$

using *assms unfolding mult [of a y] lift-def* by auto

then show $x \in elts\ y$

by (metis *arg-subset-TC assms(2) less-TC-def mult-cancellation-lemma vsubsetD*)

show $r \in elts\ a$

by (metis *arg-subset-TC assms(2) less-TC-def mult-cancellation-lemma v(1)*)

$v(\beta)$ *vsubsetD*
qed

lemma *mult-add-mem-0* [*simp*]: $a * x \in \text{elts } (a * y) \longleftrightarrow x \in \text{elts } y \wedge 0 \in \text{elts } a$
proof –
have $x \in \text{elts } y$
if $a * x \in \text{elts } (a * y) \wedge 0 \in \text{elts } a$
using *that* **using** *mult-add-mem* [*of a x 0*]
using *nonzero-less-TC* **by** *force*
moreover **have** $a * x \in \text{elts } (a * y)$
if $x \in \text{elts } y \wedge 0 \in \text{elts } a$
using *that* **by** (*force simp: image-iff mult* [*of a y*] *lift-def*)
ultimately show *?thesis*
by (*metis mult-eq-0-iff add.right-neutral mult-add-mem(2) nonzero-less-TC*)
qed

lemma *zero-mem-mult-iff*: $0 \in \text{elts } (x * y) \longleftrightarrow 0 \in \text{elts } x \wedge 0 \in \text{elts } y$
by (*metis Kirby.mult-zero-right mult-add-mem-0*)

lemma *zero-less-mult-iff* [*simp*]: $0 < x * y \longleftrightarrow 0 < x \wedge 0 < y$ **if** *Ord x*
using *Kirby.mult-eq-0-iff ZFC-in-HOL.neq0-conv* **by** *blast*

lemma *mult-cancel-less-iff* [*simp*]:
 $\llbracket \text{Ord } \alpha; \text{Ord } \beta; \text{Ord } \gamma \rrbracket \implies \alpha * \beta < \alpha * \gamma \longleftrightarrow \beta < \gamma \wedge 0 < \alpha$
using *mult-add-mem-0* [*of alpha beta gamma*]
by (*meson Ord-0 Ord-mem-iff-lt Ord-mult*)

lemma *mult-cancel-le-iff* [*simp*]:
 $\llbracket \text{Ord } \alpha; \text{Ord } \beta; \text{Ord } \gamma \rrbracket \implies \alpha * \beta \leq \alpha * \gamma \longleftrightarrow \beta \leq \gamma \vee \alpha = 0$
by (*metis Ord-linear2 Ord-mult eq-iff leD mult-cancel-less-iff mult-cancellation*)

lemma *mult-Suc-add-less*: $\llbracket \alpha < \gamma; \beta < \gamma; \text{Ord } \alpha; \text{Ord } \beta; \text{Ord } \gamma \rrbracket \implies \gamma * \text{ord-of-nat } m + \alpha < \gamma * \text{ord-of-nat } (\text{Suc } m) + \beta$
apply (*simp add: mult-succ add.assoc*)
by (*meson Ord-add Ord-linear2 le-less-trans not-add-less-right*)

lemma *mult-nat-less-add-less*:
assumes $m < n$ $\alpha < \gamma$ $\beta < \gamma$ **and** *ord: Ord alpha Ord beta Ord gamma*
shows $\gamma * \text{ord-of-nat } m + \alpha < \gamma * \text{ord-of-nat } n + \beta$
proof –
have $\text{Suc } m \leq n$
using $\langle m < n \rangle$ **by** *auto*
have $\gamma * \text{ord-of-nat } m + \alpha < \gamma * \text{ord-of-nat } (\text{Suc } m) + \beta$
using *assms mult-Suc-add-less* **by** *blast*
also **have** $\dots \leq \gamma * \text{ord-of-nat } n + \beta$
using *Ord-mult Ord-ord-of-nat add-right-mono* $\langle \text{Suc } m \leq n \rangle$ *ord mult-cancel-le-iff ord-of-nat-mono-iff* **by** *presburger*
finally show *?thesis* .
qed

```

lemma add-mult-less-add-mult:
  assumes  $x < y$   $x \in \text{elts } \beta$   $y \in \text{elts } \beta$   $\mu \in \text{elts } \alpha$   $\nu \in \text{elts } \alpha$  Ord  $\alpha$  Ord  $\beta$ 
  shows  $\alpha * x + \mu < \alpha * y + \nu$ 
proof –
  obtain Ord  $x$  Ord  $y$ 
  using Ord-in-Ord assms by blast
  then obtain  $\delta$  where  $0 \in \text{elts } \delta$   $y = x + \delta$ 
  by (metis add.right-neutral  $\langle x < y \rangle$  le-Ord-diff less-V-def mem-0-Ord)
  then show ?thesis
  apply (simp add: add-mult-distrib add.assoc)
  by (meson OrdmemD add-le-cancel-left0  $\langle \mu \in \text{elts } \alpha \rangle$   $\langle \text{Ord } \alpha \rangle$  less-le-trans
zero-imp-le-mult)
qed

lemma add-mult-less:
  assumes  $\gamma \in \text{elts } \alpha$   $\nu \in \text{elts } \beta$  Ord  $\alpha$  Ord  $\beta$ 
  shows  $\alpha * \nu + \gamma \in \text{elts } (\alpha * \beta)$ 
proof –
  have Ord  $\nu$ 
  using Ord-in-Ord assms by blast
  with assms show ?thesis
  by (metis Ord-mem-iff-lt Ord-succ add-mem-right-cancel mult-cancel-le-iff mult-succ
succ-le-iff vsubsetD)
qed

lemma Ord-add-mult-iff:
  assumes  $\beta \in \text{elts } \gamma$   $\beta' \in \text{elts } \gamma$  Ord  $\alpha$  Ord  $\alpha'$  Ord  $\gamma$ 
  shows  $\gamma * \alpha + \beta \in \text{elts } (\gamma * \alpha' + \beta') \longleftrightarrow \alpha \in \text{elts } \alpha' \vee \alpha = \alpha' \wedge \beta \in \text{elts } \beta'$ 
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof (cases  $\alpha \in \text{elts } \alpha'$ )
  case False
  with assms have  $\alpha = \alpha'$ 
  by (meson L Ord-linear Ord-mult Ord-trans add-mult-less not-add-mem-right)
  then show ?thesis
  using L less-V-def by auto
  qed auto
next
  assume R: ?rhs
  then show ?lhs
  proof
  assume  $\alpha \in \text{elts } \alpha'$ 
  then obtain  $\delta$  where  $\alpha' = \alpha + \delta$ 
  by (metis OrdmemD assms(3) assms(4) le-Ord-diff less-V-def)
  show ?lhs
  using assms

```


by (meson ⟨ $\alpha \in \text{elts } \alpha'$ ⟩ add-le-cancel-left0 add-mult-less vsubsetD)
 next
 assume $\alpha = \alpha' \wedge \beta \in \text{elts } \beta'$
 then show ?lhs
 using less-V-def by auto
 qed
 qed

lemma *vcard-mult*: $\text{vcard } (x * y) = \text{vcard } x \otimes \text{vcard } y$

proof –

have 1: $\text{elts } (\text{lift } (x * u) x) \approx \text{elts } x$ if $u \in \text{elts } y$ for u
 by (metis cardinal-epoll eqpoll-sym eqpoll-trans card-lift)
 have 2: pairwise ($\lambda u u'. \text{disjnt } (\text{elts } (\text{lift } (x * u) x)) (\text{elts } (\text{lift } (x * u') x))$) ($\text{elts } y$)
 by (simp add: pairwise-def disjnt-def) (metis V-disjoint-iff lift-mult-disjoint)
 have $x * y = (\bigsqcup_{u \in \text{elts } y} \text{lift } (x * u) x)$
 using mult by blast
 then have $\text{elts } (x * y) \approx (\bigcup_{u \in \text{elts } y} \text{elts } (\text{lift } (x * u) x))$
 by simp
 also have $\dots \approx \text{elts } y \times \text{elts } x$
 using Union-epoll-Times [OF 1 2] .
 also have $\dots \approx \text{elts } x \times \text{elts } y$
 by (simp add: times-commute-epoll)
 also have $\dots \approx \text{elts } (\text{vcard } x) \times \text{elts } (\text{vcard } y)$
 using cardinal-epoll eqpoll-sym times-epoll-cong by blast
 also have $\dots \approx \text{elts } (\text{vcard } x \otimes \text{vcard } y)$
 by (simp add: cmult-def elts-vcard-VSigma-epoll eqpoll-sym)
 finally have $\text{elts } (x * y) \approx \text{elts } (\text{vcard } x \otimes \text{vcard } y)$.
 then show ?thesis
 by (metis cadd-cmult-distrib cadd-def cardinal-cong cardinal-idem vsum-0-epoll)
 qed

proposition *TC-mult*: $\text{TC}(x * y) = (\bigsqcup_{r \in \text{elts } (\text{TC } x)} \bigsqcup_{u \in \text{elts } (\text{TC } y)} \text{set}\{x * u + r\})$

proof (cases $x = 0$)

case False
 have *: $\text{TC}(x * y) = (\bigsqcup_{u \in \text{elts } (\text{TC } y)} \text{lift } (x * u) (\text{TC } x))$ for y
 proof (induction y rule: eps-induct)
 case (step y)
 have $\text{TC}(x * y) = (\bigsqcup_{u \in \text{elts } y} \text{TC } (\text{lift } (x * u) x))$
 by (simp add: mult [of $x y$] TC-Sup-distrib image-image)
 also have $\dots = (\bigsqcup_{u \in \text{elts } y} \text{TC}(x * u) \sqcup \text{lift } (x * u) (\text{TC } x))$
 by (simp add: TC-lift False)
 also have $\dots = (\bigsqcup_{u \in \text{elts } y} (\bigsqcup_{z \in \text{elts } (\text{TC } u)} \text{lift } (x * z) (\text{TC } x)) \sqcup \text{lift } (x * u) (\text{TC } x))$
 by (simp add: step)
 also have $\dots = (\bigsqcup_{u \in \text{elts } (\text{TC } y)} \text{lift } (x * u) (\text{TC } x))$
 by (auto simp: TC' [of y] image-Un Sup-Un-distrib TC-Sup-distrib cSUP-UNION SUP-sup-distrib)

finally show *?case* .
qed
show *?thesis*
by (*force simp: * lift-def*)
qed *auto*

corollary *vcard-TC-mult*: $vcard (TC(x * y)) = vcard (TC x) \otimes vcard (TC y)$

proof –

have $(\bigcup_{u \in elts (TC x)}. \bigcup_{v \in elts (TC y)}. \{x * v + u\}) = (\bigcup_{u \in elts (TC x)}. (\lambda v. x * v + u) \text{ ‘ } elts (TC y))$

by (*simp add: UNION-singleton-eq-range*)

also have $\dots \approx (\bigcup_{x \in elts (TC x)}. elts (lift (TC y * x) (TC y)))$

proof (*rule UN-epoll-UN*)

show $(\lambda v. x * v + u) \text{ ‘ } elts (TC y) \approx elts (lift (TC y * u) (TC y))$

if $u \in elts (TC x)$ **for** u

proof –

have *inj-on* $(\lambda v. x * v + u) (elts (TC y))$

by (*meson inj-onI less-TC-def mult-cancellation-lemma that*)

then have $(\lambda v. x * v + u) \text{ ‘ } elts (TC y) \approx elts (TC y)$

by (*rule inj-on-image-epoll-self*)

also have $\dots \approx elts (lift (TC y * u) (TC y))$

by (*simp add: epoll-lift epoll-sym*)

finally show *?thesis* .

qed

show *pairwise* $(\lambda u ya. disjnt ((\lambda v. x * v + u) \text{ ‘ } elts (TC y)) ((\lambda v. x * v + u) \text{ ‘ } elts (TC y))) (elts (TC x))$

apply (*auto simp: pairwise-def disjnt-def*)

using *less-TC-def mult-cancellation-lemma* **by** *blast*

show *pairwise* $(\lambda u ya. disjnt (elts (lift (TC y * u) (TC y))) (elts (lift (TC y * u) (TC y)))) (elts (TC x))$

apply (*auto simp: pairwise-def disjnt-def*)

by (*metis Int-iff V-disjoint-iff empty-iff lift-mult-disjoint*)

qed

also have $\dots = elts (TC y * TC x)$

by (*metis elts-Sup image-image mult replacement small-elts*)

finally have $(\bigcup_{u \in elts (TC x)}. \bigcup_{v \in elts (TC y)}. \{x * v + u\}) \approx elts (TC y * TC x)$.

then show *?thesis*

apply (*subst cmult-commute*)

by (*simp add: TC-mult cardinal-cong flip: vcard-mult*)

qed

lemma *countable-mult*:

assumes *countable* $(elts A)$ *countable* $(elts B)$

shows *countable* $(elts (A * B))$

proof –

have $vcard A \leq \aleph_0$ $vcard B \leq \aleph_0$

using *assms countable-iff-le-Aleph0* **by** *blast+*

then have $\text{vcard } (A*B) \leq \aleph_0$
unfolding vcard-mult
by (*metis InfCard-csquare-eq cmult-le-mono Aleph-0 Card- ω InfCard-def order-refl*)
then show *?thesis*
by (*simp add: countable-iff-le-Aleph0*)
qed

3.4 Ordertype properties

lemma *ordertype-image-plus*:
assumes $\text{Ord } \alpha$
shows $\text{ordertype } ((+) u \text{ ' } \text{elts } \alpha) \text{ VWF} = \alpha$
proof (*subst ordertype-VWF-eq-iff*)
have $1: (u + x, u + y) \in \text{VWF}$ **if** $x \in \text{elts } \alpha$ $y \in \text{elts } \alpha$ $x < y$ **for** $x y$
using *that*
by (*meson Ord-in-Ord Ord-mem-iff-lt add-mem-right-cancel assms mem-imp-VWF*)
then have $2: x < y$
if $x \in \text{elts } \alpha$ $y \in \text{elts } \alpha$ $(u + x, u + y) \in \text{VWF}$ **for** $x y$
using *that* **by** (*metis Ord-in-Ord Ord-linear-lt VWF-asym assms*)
show $\exists f. \text{bij-betw } f ((+) u \text{ ' } \text{elts } \alpha) (\text{elts } \alpha) \wedge (\forall x \in (+) u \text{ ' } \text{elts } \alpha. \forall y \in (+) u \text{ ' } \text{elts } \alpha. (f x < f y) = ((x, y) \in \text{VWF}))$
using $1\ 2$ **unfolding** *bij-betw-def inj-on-def*
by (*rule-tac x= $\lambda x. \text{odiff } x u$ in exI*) (*auto simp: image-iff*)
qed (*use assms in auto*)

lemma *ordertype-diff*:
assumes $\beta + \delta = \alpha$ **and** $\alpha: \delta \in \text{elts } \alpha$ $\text{Ord } \alpha$
shows $\text{ordertype } (\text{elts } \alpha - \text{elts } \beta) \text{ VWF} = \delta$
proof –
have $*$: $\text{elts } \alpha - \text{elts } \beta = ((+)\beta) \text{ ' } \text{elts } \delta$
proof
show $\text{elts } \alpha - \text{elts } \beta \subseteq (+) \beta \text{ ' } \text{elts } \delta$
by *clarsimp (metis assms(1) image-iff mem-plus-V-E)*
show $(+) \beta \text{ ' } \text{elts } \delta \subseteq \text{elts } \alpha - \text{elts } \beta$
using *assms(1) not-add-mem-right* **by** *force*
qed
have $\text{ordertype } ((+) \beta \text{ ' } \text{elts } \delta) \text{ VWF} = \delta$
proof (*subst ordertype-VWF-inc-eq*)
show $\text{elts } \delta \subseteq \text{ON ordertype } (\text{elts } \delta) \text{ VWF} = \delta$
using α *elts-subset-ON ordertype-eq-Ord* **by** *blast+*
qed (*use * assms elts-subset-ON in auto*)
then show *?thesis*
by (*simp add: **)
qed

lemma *ordertype-interval-eq*:
assumes $\alpha: \text{Ord } \alpha$ **and** $\beta: \text{Ord } \beta$
shows $\text{ordertype } (\{\alpha ..< \alpha + \beta\} \cap \text{ON}) \text{ VWF} = \beta$

proof –
have $ON: (+) \alpha \text{ ' } elts \beta \subseteq ON$
using *assms Ord-add Ord-in-Ord* **by** *blast*
have $(\{\alpha ..< \alpha+\beta\} \cap ON) = (+) \alpha \text{ ' } elts \beta$
using *assms*
apply (*simp add: image-def set-eq-iff*)
by (*metis add-less-cancel-left Ord-add Ord-in-Ord Ord-linear2 Ord-mem-iff-lt*
le-Ord-diff not-add-less-right)
moreover have $ordertype (elts \beta) VWF = ordertype ((+) \alpha \text{ ' } elts \beta) VWF$
using $ON \beta elts-subset-ON ordertype-VWF-inc-eq$ **by** *auto*
ultimately show *?thesis*
using β **by** *auto*
qed

lemma *ordertype-Times*:

assumes *small A small B* **and** $r: wf \ r \ trans \ r \ total-on \ A \ r$ **and** $s: wf \ s \ trans \ s \ total-on \ B \ s$

shows $ordertype (A \times B) (r \ < *lex* > \ s) = ordertype \ B \ s * ordertype \ A \ r$ (**is** $= ?\beta * ?\alpha$)

proof (*subst ordertype-eq-iff*)

show $Ord \ (?\beta * ?\alpha)$

by (*intro wf-Ord-ordertype Ord-mult r s; simp*)

define f **where** $f \equiv \lambda(x,y). ?\beta * ordermap \ A \ r \ x + (ordermap \ B \ s \ y)$

show $\exists f. \text{bij-betw } f \ (A \times B) \ (elts \ (?\beta * ?\alpha)) \wedge (\forall x \in A \times B. \forall y \in A \times B. (f \ x < f \ y) = ((x, y) \in (r \ < *lex* > \ s)))$

unfolding *bij-betw-def*

proof (*intro exI conjI strip*)

show *inj-on* $f \ (A \times B)$

proof (*clarsimp simp: f-def inj-on-def*)

fix $x \ y \ x' \ y'$

assume $x \in A \ y \in B \ x' \in A \ y' \in B$

and $eq: ?\beta * ordermap \ A \ r \ x + ordermap \ B \ s \ y = ?\beta * ordermap \ A \ r \ x' + ordermap \ B \ s \ y'$

have $ordermap \ A \ r \ x = ordermap \ A \ r \ x' \wedge$

$ordermap \ B \ s \ y = ordermap \ B \ s \ y'$

proof (*rule mult-cancellation-lemma [OF eq]*)

show $ordermap \ B \ s \ y \sqsubset ?\beta$

using *ordermap-in-ordertype [OF <y ∈ B>, of s] less-TC-iff <small B>* **by** *blast*

show $ordermap \ B \ s \ y' \sqsubset ?\beta$

using *ordermap-in-ordertype [OF <y' ∈ B>, of s] less-TC-iff <small B>* **by** *blast*

qed

then show $x = x' \wedge y = y'$

using $\langle x \in A \rangle \langle x' \in A \rangle \langle y \in B \rangle \langle y' \in B \rangle r \ s \ \langle small \ A \rangle \ \langle small \ B \rangle$ **by** *auto*

qed

show $f \text{ ' } (A \times B) = elts \ (?\beta * ?\alpha)$ (**is** $?lhs = ?rhs$)

proof

show $f \text{ ' } (A \times B) \subseteq elts \ (?\beta * ?\alpha)$

```

apply (auto simp: f-def add-mult-less ordermap-in-ordertype wf-Ord-ordertype
r s)
  by (simp add: add-mult-less assms ordermap-in-ordertype wf-Ord-ordertype)
show elts (?β * ?α) ⊆ f ' (A × B)
proof (clarsimp simp: f-def image-iff elim !: elts-multE split: prod.split)
  fix u v
  assume u: u ∈ elts (?β) and v: v ∈ elts ?α
  have inv-into B (ordermap B s) u ∈ B
    by (simp add: inv-into-ordermap u)
  moreover have inv-into A (ordermap A r) v ∈ A
    by (simp add: inv-into-ordermap v)
  ultimately show ∃ x ∈ A. ∃ y ∈ B. ?β * v + u = ?β * ordermap A r x +
ordermap B s y
    by (metis ‹small A› ‹small B› bij-betw-inv-into-right ordermap-bij r(1)
r(3) s(1) s(3) u v)
  qed
qed
next
fix p q
assume p ∈ A × B and q ∈ A × B
then obtain u v x y where §: p = (u,v) u ∈ A v ∈ B q = (x,y) x ∈ A y ∈ B
  by blast
show ((f p) < f q) = ((p, q) ∈ (r < *lex* > s))
proof
  assume f p < f q
  with § assms have (u, x) ∈ r ∨ u=x ∧ (v, y) ∈ s
    apply (simp add: f-def)
  by (metis Ord-add Ord-add-mult-iff Ord-mem-iff-lt Ord-mult wf-Ord-ordermap
converse-ordermap-mono
ordermap-eq-iff ordermap-in-ordertype wf-Ord-ordertype)
  then show (p,q) ∈ (r < *lex* > s)
    by (simp add: §)
next
assume (p,q) ∈ (r < *lex* > s)
then have (u, x) ∈ r ∨ u = x ∧ (v, y) ∈ s
  by (simp add: §)
then show f p < f q
proof
  assume ux: (u, x) ∈ r
  have oo: ∧ x. Ord (ordermap A r x) ∧ y. Ord (ordermap B s y)
    by (simp-all add: r s)
  show f p < f q
proof (clarsimp simp: f-def split: prod.split)
  fix a b a' b'
  assume p = (a, b) and q = (a', b')
  then have ?β * ordermap A r a + ordermap B s b < ?β * ordermap A r
a'
    using ux assms §
    by (metis Ord-mult wf-Ord-ordermap OrdmemD Pair-inject add-mult-less

```

```

ordermap-in-ordertype ordermap-mono wf-Ord-ordertype)
  also have ... ≤ ?β * ordermap A r a' + ordermap B s b'
  by simp
  finally show ?β * ordermap A r a + ordermap B s b < ?β * ordermap A
r a' + ordermap B s b'.
  qed
next
  assume u = x ∧ (v, y) ∈ s
  then show f p < f q
    using § assms by (fastforce simp: f-def split: prod.split intro: or-
dermap-mono-less)
  qed
qed
qed
qed (use assms small-Times in auto)

end

```

4 Exponentiation of ordinals

```

theory Ordinal-Exp
  imports Kirby

```

```
begin
```

Source: Schlöder, Julian. Ordinal Arithmetic; available online at <http://www.math.uni-bonn.de/ag/logik/teaching/2012WS/Set%20theory/oa.pdf>

```

definition oexp :: [V, V] ⇒ V (infixr ‹↑› 80)
  where oexp a b ≡ transrec (λf x. if x=0 then 1
    else if Limit x then if a=0 then 0 else ⋂ξ ∈ elts x. f ξ
    else f (⋂(elts x) * a) b

```

$0 \uparrow \omega = 1$ if we don't make a special case for Limit ordinals and zero

```

lemma oexp-0-right [simp]:  $\alpha \uparrow 0 = 1$ 
  by (simp add: def-transrec [OF oexp-def])

```

```

lemma oexp-succ [simp]:  $\text{Ord } \beta \implies \alpha \uparrow (\text{succ } \beta) = \alpha \uparrow \beta * \alpha$ 
  by (simp add: def-transrec [OF oexp-def])

```

```

lemma oexp-Limit:  $\text{Limit } \beta \implies \alpha \uparrow \beta = (\text{if } \alpha=0 \text{ then } 0 \text{ else } \bigwedge \xi \in \text{elts } \beta. \alpha \uparrow \xi)$ 
  by (auto simp: def-transrec [OF oexp-def, of - β])

```

```

lemma oexp-1-right [simp]:  $\alpha \uparrow 1 = \alpha$ 
  using one-V-def oexp-succ by fastforce

```

```

lemma oexp-1 [simp]:  $\text{Ord } \alpha \implies 1 \uparrow \alpha = 1$ 
  by (induction rule: Ord-induct3) (use Limit-def oexp-Limit in auto)

```

lemma *oexp-0* [*simp*]: $\text{Ord } \alpha \implies 0 \uparrow \alpha = (\text{if } \alpha = 0 \text{ then } 1 \text{ else } 0)$
by (*induction rule: Ord-induct3*) (*use Limit-def oexp-Limit in auto*)

lemma *oexp-eq-0-iff* [*simp*]:
assumes $\text{Ord } \beta$ **shows** $\alpha \uparrow \beta = 0 \iff \alpha = 0 \wedge \beta \neq 0$
using $\langle \text{Ord } \beta \rangle$
proof (*induction rule: Ord-induct3*)
case (*Limit* μ)
then show *?case*
using *Limit-def oexp-Limit by auto*
qed *auto*

lemma *oexp-gt-0-iff* [*simp*]:
assumes $\text{Ord } \beta$ **shows** $\alpha \uparrow \beta > 0 \iff \alpha > 0 \vee \beta = 0$
by (*simp add: assms less-V-def*)

lemma *ord-of-nat-oexp*: $\text{ord-of-nat } (m \hat{\ } n) = \text{ord-of-nat } m \uparrow \text{ord-of-nat } n$
proof (*induction n*)
case (*Suc n*)
then show *?case*
by (*simp add: mult.commute [of m]*) (*simp add: ord-of-nat-mult*)
qed *auto*

lemma *omega-closed-oexp* [*intro*]:
assumes $\alpha \in \text{elts } \omega$ $\beta \in \text{elts } \omega$ **shows** $\alpha \uparrow \beta \in \text{elts } \omega$
proof –
obtain m n **where** $\alpha = \text{ord-of-nat } m$ $\beta = \text{ord-of-nat } n$
using *assms elts- ω by auto*
then have $\alpha \uparrow \beta = \text{ord-of-nat } (m \hat{\ } n)$
by (*simp add: ord-of-nat-oexp*)
then show *?thesis*
by (*simp add: ω -def*)
qed

lemma *Ord-oexp* [*simp*]:
assumes $\text{Ord } \alpha$ $\text{Ord } \beta$ **shows** $\text{Ord } (\alpha \uparrow \beta)$
using $\langle \text{Ord } \beta \rangle$
proof (*induction rule: Ord-induct3*)
case (*Limit* α)
then show *?case*
by (*auto simp: oexp-Limit image-iff intro: Ord-Sup*)
qed (*auto intro: Ord-mult assms*)

Lemma 3.19

lemma *le-oexp*:
assumes $\text{Ord } \alpha$ $\text{Ord } \beta$ $\beta \neq 0$ **shows** $\alpha \leq \alpha \uparrow \beta$
using $\langle \text{Ord } \beta \rangle$ $\langle \beta \neq 0 \rangle$
proof (*induction rule: Ord-induct3*)

```

case (succ  $\beta$ )
then show ?case
  by simp (metis  $\langle \text{Ord } \alpha \rangle$  le-0 le-mult mult.left-neutral oexp-0-right order-refl
order-trans)
next
  case (Limit  $\mu$ )
  then show ?case
    by (metis Limit-def Limit-eq-Sup-self ZFC-in-HOL.Sup-upper eq-iff image-eqI
image-ident oexp-1-right oexp-Limit replacement small-elts one-V-def)
qed auto

```

Lemma 3.20

lemma *le-oexp'*:

assumes *Ord* α $1 < \alpha$ *Ord* β **shows** $\beta \leq \alpha \uparrow \beta$

proof (*cases* $\beta = 0$)

case *True*

then show ?*thesis*

by *auto*

next

case *False*

show ?*thesis*

using $\langle \text{Ord } \beta \rangle$

proof (*induction rule: Ord-induct3*)

case *0*

then show ?*case*

by *auto*

next

case (*succ* γ)

then have $\alpha \uparrow \gamma * 1 < \alpha \uparrow \gamma * \alpha$

using $\langle \text{Ord } \alpha \rangle$ $\langle 1 < \alpha \rangle$

by (*metis* *le-mult* *less-V-def* *mult.right-neutral* *mult-cancellation* *not-less-0*
oexp-eq-0-iff *succ.hyps*)

then have $\gamma < \alpha \uparrow \text{succ } \gamma$

using *succ.IH* *succ.hyps* **by** *auto*

then show ?*case*

using *False* $\langle \text{Ord } \alpha \rangle$ $\langle 1 < \alpha \rangle$ *succ*

by (*metis* *Ord-mem-iff-lt* *Ord-oexp* *Ord-succ* *elts-succ* *insert-subset* *less-eq-V-def*
less-imp-le)

next

case (*Limit* μ)

with *False* $\langle 1 < \alpha \rangle$ **show** ?*case*

by (*force simp: Limit-def* *oexp-Limit* *intro: elts-succ*)

qed

qed

lemma *oexp-Limit-le*:

assumes $\beta < \gamma$ *Limit* γ *Ord* β $\alpha > 0$ **shows** $\alpha \uparrow \beta \leq \alpha \uparrow \gamma$

proof –


```

have Ord  $\gamma$ 
  using Limit-def assms(2) by blast
with assms show ?thesis
  using Ord-mem-iff-lt ZFC-in-HOL.Sup-upper oexp-Limit by auto
qed

proposition oexp-less:
  assumes  $\beta: \beta \in \text{elts } \gamma$  and Ord  $\gamma$  and  $\alpha: \alpha > 1$  Ord  $\alpha$  shows  $\alpha \uparrow \beta < \alpha \uparrow \gamma$ 
proof –
  obtain  $\beta < \gamma$  Ord  $\beta$ 
    using Ord-in-Ord OrdmemD assms by auto
  have gt0:  $\alpha \uparrow \beta > 0$ 
    using  $\langle \text{Ord } \beta \rangle \alpha$  dual-order.order-iff-strict by auto
  show ?thesis
    using  $\langle \text{Ord } \gamma \rangle \beta$ 
  proof (induction rule: Ord-induct3)
    case 0
    then show ?case
      by auto
  next
    case (succ  $\delta$ )
    then consider  $\beta = \delta \mid \beta < \delta$ 
      using OrdmemD elts-succ by blast
    then show ?case
  proof cases
    case 1
    then have  $(\alpha \uparrow \beta) * 1 < (\alpha \uparrow \delta) * \alpha$ 
      using Ord-1 Ord-oexp  $\alpha$  gt0 mult-cancel-less-iff succ.hyps by metis
    then show ?thesis
      by (simp add: succ.hyps)
  next
    case 2
    then have  $(\alpha \uparrow \delta) * 1 < (\alpha \uparrow \delta) * \alpha$ 
    by (meson Ord-1 Ord-mem-iff-lt Ord-oexp  $\langle \text{Ord } \beta \rangle \alpha$  gt0 less-trans mult-cancel-less-iff
succ)
    with 2 show ?thesis
      using Ord-mem-iff-lt  $\langle \text{Ord } \beta \rangle$  succ by auto
  qed
  next
    case (Limit  $\gamma$ )
    then obtain Ord  $\gamma$  succ  $\beta < \gamma$ 
      using Limit-def Ord-in-Ord OrdmemD assms by auto
    have  $\alpha \uparrow \beta = (\alpha \uparrow \beta) * 1$ 
      by simp
    also have  $\dots < (\alpha \uparrow \beta) * \alpha$ 
      using Ord-oexp  $\langle \text{Ord } \beta \rangle$  assms gt0 mult-cancel-less-iff by blast
    also have  $\dots = \alpha \uparrow \text{succ } \beta$ 
      by (simp add:  $\langle \text{Ord } \beta \rangle$ )
    also have  $\dots \leq (\bigsqcup \xi \in \text{elts } \gamma. \alpha \uparrow \xi)$ 

```

```

proof –
  have  $\text{succ } \beta \in \text{elts } \gamma$ 
    using Limit.hyps Limit.premis Limit-def by auto
  then show ?thesis
    by (simp add: ZFC-in-HOL.Sup-upper)
qed
finally
have  $\alpha \uparrow \beta < (\bigsqcup \xi \in \text{elts } \gamma. \alpha \uparrow \xi)$  .
then show ?case
  using Limit.hyps oexp-Limit  $\langle \alpha > 1 \rangle$  by auto
qed
qed

corollary oexp-less-iff:
  assumes  $\alpha > 0$  Ord  $\alpha$  Ord  $\beta$  Ord  $\gamma$  shows  $\alpha \uparrow \beta < \alpha \uparrow \gamma \iff \beta \in \text{elts } \gamma \wedge \alpha > 1$ 
proof safe
show  $\beta \in \text{elts } \gamma$   $1 < \alpha$ 
  if  $\alpha \uparrow \beta < \alpha \uparrow \gamma$ 
proof –
  show  $\alpha > 1$ 
  proof (rule ccontr)
    assume  $\neg \alpha > 1$ 
    then consider  $\alpha=0 \mid \alpha=1$ 
      using  $\langle \text{Ord } \alpha \rangle$  less-V-def mem-0-Ord by fastforce
    then show False
      by cases (use that  $\langle \alpha > 0 \rangle$   $\langle \text{Ord } \beta \rangle$   $\langle \text{Ord } \gamma \rangle$  in auto split: if-split-asm)
  qed
show  $\beta: \beta \in \text{elts } \gamma$ 
proof (rule ccontr)
  assume  $\beta \notin \text{elts } \gamma$ 
  then have  $\gamma \leq \beta$ 
    by (meson Ord-linear-le Ord-mem-iff-lt assms less-le-not-le)
  then consider  $\gamma = \beta \mid \gamma < \beta$ 
    using less-V-def by blast
  then show False
proof cases
  case 1
    then show ?thesis
      using that by blast
  next
  case 2
    with  $\langle \alpha > 1 \rangle$  have  $\alpha \uparrow \gamma < \alpha \uparrow \beta$ 
      by (simp add: Ord-mem-iff-lt assms oexp-less)
    with that show ?thesis
      by auto
  qed
qed
qed
show  $\alpha \uparrow \beta < \alpha \uparrow \gamma$  if  $\beta \in \text{elts } \gamma$   $1 < \alpha$ 

```

using *that* by (*simp add: assms oexp-less*)
 qed

lemma ω -*oexp-iff* [*simp*]: $\llbracket \text{Ord } \alpha; \text{Ord } \beta \rrbracket \implies \omega \uparrow \alpha = \omega \uparrow \beta \iff \alpha = \beta$
 by (*metis Ord- ω Ord-linear ω -gt1 less-irrefl oexp-less*)

lemma *Limit-oexp*:
 assumes *Limit* γ *Ord* α $\alpha > 1$ **shows** *Limit* $(\alpha \uparrow \gamma)$
 unfolding *Limit-def*
proof *safe*
 show $O\alpha\gamma$: *Ord* $(\alpha \uparrow \gamma)$
 using *Limit-def Ord-oexp* $\langle \text{Limit } \gamma \rangle$ *assms(2)* **by** *blast*
 show 0 : $0 \in \text{elts } (\alpha \uparrow \gamma)$
 using *Limit-def oexp-Limit* $\langle \text{Limit } \gamma \rangle$ $\langle \alpha > 1 \rangle$ **by** *fastforce*
 have *Ord* γ
 using *Limit-def* $\langle \text{Limit } \gamma \rangle$ **by** *blast*
fix x
 assume x : $x \in \text{elts } (\alpha \uparrow \gamma)$
with $\langle \text{Limit } \gamma \rangle$ $\langle \alpha > 1 \rangle$
obtain β **where** $\beta < \gamma$ *Ord* β *Ord* x **and** $x\beta$: $x \in \text{elts } (\alpha \uparrow \beta)$
 apply (*simp add: oexp-Limit split: if-split-asm*)
 using *Ord-in-Ord OrdmemD* $\langle \text{Ord } \gamma \rangle$ $O\alpha\gamma$ x **by** *blast*
then have $O\alpha\beta$: *Ord* $(\alpha \uparrow \beta)$
 using *Ord-oexp assms(2)* **by** *blast*
 have $\beta \in \text{elts } \gamma$
 by (*simp add: Ord-mem-iff-lt* $\langle \text{Ord } \beta \rangle$ $\langle \text{Ord } \gamma \rangle$ $\langle \beta < \gamma \rangle$)
moreover have $\alpha \neq 0$
 using $\langle \alpha > 1 \rangle$ **by** *blast*
ultimately have $\alpha\beta\gamma$: $\alpha \uparrow \beta \leq \alpha \uparrow \gamma$
 by (*simp add: Sup-upper oexp-Limit* $\langle \text{Limit } \gamma \rangle$)
 have *succ* $x \leq \alpha \uparrow \beta$
 by (*simp add: OrdmemD O $\alpha\beta$ $\langle \text{Ord } x \rangle$ succ-le-iff $x\beta$*)
then consider *succ* $x < \alpha \uparrow \beta$ | *succ* $x = \alpha \uparrow \beta$
 using *le-neq-trans* **by** *blast*
then show *succ* $x \in \text{elts } (\alpha \uparrow \gamma)$
proof *cases*
 case 1
with $\alpha\beta\gamma$ **show** *?thesis*
 using $O\alpha\beta$ *Ord-mem-iff-lt* $\langle \text{Ord } x \rangle$ **by** *blast*
next
 case 2
then have *succ* $\beta < \gamma$
 using *Limit-def OrdmemD* $\langle \beta \in \text{elts } \gamma \rangle$ *assms(1)* **by** *auto*
 have *ge1*: $1 \leq \alpha \uparrow \beta$
 by (*metis 2 Ord-0* $\langle \text{Ord } x \rangle$ *le-0 le-succ-iff one-V-def*)
 have *succ* $x < \text{succ } (\alpha \uparrow \beta)$
 using 2 $O\alpha\beta$ *succ-le-iff* **by** *auto*
also have $\dots \leq (\alpha \uparrow \beta) + (\alpha \uparrow \beta)$
 using *ge1* **by** (*simp add: succ-eq-add1*)

also have $\dots = (\alpha \uparrow \beta) * \text{succ} (\text{succ } 0)$
by (*simp add: mult-succ*)
also have $\dots \leq (\alpha \uparrow \beta) * \alpha$
using $O\alpha\beta$ *Ord-succ* *assms(2)* *assms(3)* *one-V-def succ-le-iff* **by** *auto*
also have $\dots = \alpha \uparrow \text{succ } \beta$
by (*simp add: Ord beta*)
also have $\dots \leq \alpha \uparrow \gamma$
by (*meson Limit-def* $\langle \beta \in \text{elts } \gamma \rangle$ *assms dual-order.order-iff-strict oexp-less*)
finally show *?thesis*
by (*simp add: 2 Oalpha beta Oalpha gamma Ord-mem-iff-lt*)
qed
qed

lemma *oexp-mono*:
assumes α : *Ord* $\alpha \neq 0$ **and** β : *Ord* $\beta \sqsubseteq \beta$ **shows** $\alpha \uparrow \gamma \leq \alpha \uparrow \beta$
using β
proof (*induction rule: Ord-induct3*)
case 0
then show *?case*
by *simp*
next
case (*succ* β)
with α *le-mult* **show** *?case*
by (*auto simp: le-TC-succ*)
next
case (*Limit* μ)
then have $\alpha \uparrow \gamma \leq \bigsqcup ((\uparrow) \alpha \text{ 'elts } \mu)$
using *Limit.hyps* *Ord-less-TC-mem* $\langle \alpha \neq 0 \rangle$ *le-TC-def* **by** (*auto simp: oexp-Limit*
Limit-def)
then show *?case*
using α **by** (*simp add: oexp-Limit Limit.hyps*)
qed

lemma *oexp-mono-le*:
assumes $\gamma \leq \beta$ $\alpha \neq 0$ *Ord* α *Ord* β *Ord* γ **shows** $\alpha \uparrow \gamma \leq \alpha \uparrow \beta$
by (*simp add: assms oexp-mono vle2 vle-iff-le-Ord*)

lemma *oexp-sup*:
assumes $\alpha \neq 0$ *Ord* α *Ord* β *Ord* γ **shows** $\alpha \uparrow (\beta \sqcup \gamma) = \alpha \uparrow \beta \sqcup \alpha \uparrow \gamma$
by (*metis Ord-linear-le assms oexp-mono-le sup.absorb2 sup.orderE*)

lemma *oexp-Sup*:
assumes $\alpha \neq 0$ *Ord* α **and** X : $X \subseteq ON$ *small* X $X \neq \{\}$ **shows** $\alpha \uparrow \bigsqcup X =$
 $\bigsqcup ((\uparrow) \alpha \text{ ' } X)$
proof (*rule order-antisym*)
show $\bigsqcup ((\uparrow) \alpha \text{ ' } X) \leq \alpha \uparrow \bigsqcup X$
by (*metis ON-imp-Ord Ord-Sup ZFC-in-HOL.Sup-upper assms cSUP-least*)

```

oexp-mono-le)
next
  have Ord (Sup X)
  using Ord-Sup X by auto
  then show  $\alpha \uparrow \bigsqcup X \leq \bigsqcup ((\uparrow) \alpha \text{ ' } X)$ 
  proof (cases rule: Ord-cases)
  case 0
  then show ?thesis
  using X dual-order.antisym by fastforce
next
  case (succ  $\beta$ )
  then show ?thesis
  using ZFC-in-HOL.Sup-upper X succ-in-Sup-Ord by auto
next
  case limit
  show ?thesis
  proof (clarsimp simp: assms oexp-Limit limit)
  fix x y z
  assume x:  $x \in \text{elts } (\alpha \uparrow y)$  and z:  $z \in X \ y \in \text{elts } z$ 
  then have  $\alpha \uparrow y \leq \alpha \uparrow z$ 
  by (meson ON-imp-Ord Ord-in-Ord OrdmemD  $\alpha \text{ ' } X \subseteq ON$ ) le-less oexp-mono-le)
  with x have  $x \in \text{elts } (\alpha \uparrow z)$  by blast
  then show  $\exists u \in X. x \in \text{elts } (\alpha \uparrow u)$ 
  using  $\langle z \in X \rangle$  by blast
  qed
qed
qed

```

```

lemma omega-le-Limit:
  assumes Limit  $\mu$  shows  $\omega \leq \mu$ 
proof
  fix  $\rho$ 
  assume  $\rho \in \text{elts } \omega$ 
  then obtain n where  $\rho = \text{ord-of-nat } n$ 
  using elts- $\omega$  by auto
  have  $\text{ord-of-nat } n \in \text{elts } \mu$ 
  by (induction n) (use Limit-def assms in auto)
  then show  $\rho \in \text{elts } \mu$ 
  using  $\langle \rho = \text{ord-of-nat } n \rangle$  by auto
qed

```

```

lemma finite-omega-power [simp]:
  assumes  $1 < n \ n \in \text{elts } \omega$  shows  $n \uparrow \omega = \omega$ 
proof (rule order-antisym)
  have  $\bigsqcup ((\uparrow) (\text{ord-of-nat } k) \text{ ' } \text{elts } \omega) \leq \omega$  for k
  proof (induction k)
  case 0
  then show ?case

```

```

    by auto
  next
  case (Suc k)
  then show ?case
    by (metis Ord- $\omega$  OrdmemD Sup-eq-0-iff ZFC-in-HOL.SUP-le-iff le-0 le-less
omega-closed-oexp ord-of-nat- $\omega$ )
  qed
  then show  $n\uparrow\omega \leq \omega$ 
    using assms
    by (simp add: elts- $\omega$  oexp-Limit) metis
  show  $\omega \leq n\uparrow\omega$ 
    using Ord-in-Ord assms le-oexp' by blast
qed

```

proposition *oexp-add*:

assumes $Ord\ \alpha\ Ord\ \beta\ Ord\ \gamma$ shows $\alpha\uparrow(\beta + \gamma) = \alpha\uparrow\beta * \alpha\uparrow\gamma$

proof (cases $\langle\alpha = 0\rangle$)

case True

then show ?thesis

using assms by simp

next

case False

show ?thesis

using $\langle Ord\ \gamma\rangle$

proof (induction rule: Ord-induct3)

case 0

then show ?case

by auto

next

case (succ ξ)

then show ?case

using $\langle Ord\ \beta\rangle$ by (auto simp: plus-V-succ-right mult.assoc)

next

case (Limit μ)

have $\alpha\uparrow(\beta + (\bigsqcup \xi \in elts\ \mu.\ \xi)) = (\bigsqcup \xi \in elts\ (\beta + \mu).\ \alpha\uparrow\xi)$

by (simp add: Limit.hyps oexp-Limit assms False)

also have $\dots = (\bigsqcup \xi \in \{\xi.\ Ord\ \xi \wedge \beta + \xi < \beta + \mu\}.\ \alpha\uparrow(\beta + \xi))$

proof (rule Sup-eq-Sup)

show $(\lambda\xi.\ \alpha\uparrow(\beta + \xi))\ \text{'}\ \{\xi.\ Ord\ \xi \wedge \beta + \xi < \beta + \mu\} \subseteq (\uparrow)\ \alpha\ \text{'}\ elts\ (\beta + \mu)$

using Limit.hyps Limit-def Ord-mem-iff-lt imageI by blast

fix x

assume $x \in (\uparrow)\ \alpha\ \text{'}\ elts\ (\beta + \mu)$

then obtain ξ where $\xi: \xi \in elts\ (\beta + \mu)$ and $x: x = \alpha\uparrow\xi$

by auto

have $\exists\gamma.\ Ord\ \gamma \wedge \gamma < \mu \wedge \alpha\uparrow\xi \leq \alpha\uparrow(\beta + \gamma)$

proof (rule mem-plus-V-E [OF ξ])

assume $\xi \in elts\ \beta$

then have $\alpha\uparrow\xi \leq \alpha\uparrow\beta$

```

      by (meson arg-subset-TC assms False le-TC-def less-TC-def oexp-mono
vsubsetD)
    with zero-less-Limit [OF ‹Limit μ›]
    show  $\exists \gamma. \text{Ord } \gamma \wedge \gamma < \mu \wedge \alpha \uparrow \xi \leq \alpha \uparrow (\beta + \gamma)$ 
      by force
  next
  fix  $\delta$ 
  assume  $\delta \in \text{elts } \mu$  and  $\xi = \beta + \delta$ 
  have  $\text{Ord } \delta$ 
    using Limit.hyps Limit-def Ord-in-Ord ‹ $\delta \in \text{elts } \mu$ › by blast
  moreover have  $\delta < \mu$ 
    using Limit.hyps Limit-def OrdmemD ‹ $\delta \in \text{elts } \mu$ › by auto
  ultimately show  $\exists \gamma. \text{Ord } \gamma \wedge \gamma < \mu \wedge \alpha \uparrow \xi \leq \alpha \uparrow (\beta + \gamma)$ 
    using ‹ $\xi = \beta + \delta$ › by blast
qed
then show  $\exists y \in (\lambda \xi. \alpha \uparrow (\beta + \xi)) \cdot \{ \xi. \text{Ord } \xi \wedge \beta + \xi < \beta + \mu \}. x \leq y$ 
  using x by auto
qed auto
also have  $\dots = (\bigsqcup \xi \in \text{elts } \mu. \alpha \uparrow (\beta + \xi))$ 
  using ‹Limit μ›
  by (simp add: Ord-Collect-lt Limit-def)
also have  $\dots = (\bigsqcup \xi \in \text{elts } \mu. \alpha \uparrow \beta * \alpha \uparrow \xi)$ 
  using Limit.IH by auto
also have  $\dots = \alpha \uparrow \beta * \alpha \uparrow (\bigsqcup \xi \in \text{elts } \mu. \xi)$ 
  using ‹ $\alpha \neq 0$ › Limit.hyps
  by (simp add: image-image oexp-Limit mult-Sup-distrib)
finally show ?case .
qed
qed

proposition oexp-mult:
  assumes  $\text{Ord } \alpha \text{ Ord } \beta \text{ Ord } \gamma$  shows  $\alpha \uparrow (\beta * \gamma) = (\alpha \uparrow \beta) \uparrow \gamma$ 
proof (cases  $\alpha = 0 \vee \beta = 0$ )
  case True
  then show ?thesis
    by (auto simp: ‹Ord β› ‹Ord γ›)
  next
  case False
  show ?thesis
    using ‹Ord γ›
  proof (induction rule: Ord-induct3)
  case 0
  then show ?case
    by auto
  next
  case succ
  then show ?case
    using assms by (auto simp: mult-succ oexp-add)
  next

```

```

case (Limit  $\mu$ )
have Lim: Limit ( $\bigsqcup ((*) \beta \text{ ' elts } \mu)$ )
  unfolding Limit-def
proof (intro conjI allI impI)
  show Ord ( $\bigsqcup ((*) \beta \text{ ' elts } \mu)$ )
    using Limit.hyps Limit-def Ord-in-Ord  $\langle \text{Ord } \beta \rangle$  by (auto intro: Ord-Sup)
  have succ 0  $\in$  elts  $\mu$ 
    using Limit.hyps Limit-def by blast
  then show 0  $\in$  elts ( $\bigsqcup ((*) \beta \text{ ' elts } \mu)$ )
    using False  $\langle \text{Ord } \beta \rangle$  mem-0-Ord by force
  show succ y  $\in$  elts ( $\bigsqcup ((*) \beta \text{ ' elts } \mu)$ )
    if y  $\in$  elts ( $\bigsqcup ((*) \beta \text{ ' elts } \mu)$ ) for y
    using that False Limit.hyps
    apply (clarsimp simp: Limit-def)
    by (metis Ord-in-Ord Ord-linear Ord-mem-iff-lt Ord-mult Ord-succ assms(2)
less-V-def mult-cancellation mult-succ not-add-mem-right succ-le-iff succ-ne-self)
  qed
  have  $\alpha \uparrow (\beta * (\bigsqcup \xi \in \text{elts } \mu. \xi)) = \alpha \uparrow \bigsqcup ((*) \beta \text{ ' elts } \mu)$ 
    by (simp add: mult-Sup-distrib)
  also have ... =  $\bigsqcup (\bigcup x \in \text{elts } \mu. (\uparrow) \alpha \text{ ' elts } (\beta * x))$ 
    using False Lim oexp-Limit by fastforce
  also have ... = ( $\bigsqcup x \in \text{elts } \mu. \alpha \uparrow (\beta * x)$ )
  proof (rule Sup-eq-Sup)
    show  $(\lambda x. \alpha \uparrow (\beta * x)) \text{ ' elts } \mu \subseteq (\bigcup x \in \text{elts } \mu. (\uparrow) \alpha \text{ ' elts } (\beta * x))$ 
      using  $\langle \text{Ord } \alpha \rangle \langle \text{Ord } \beta \rangle$  False Limit
      applyclarsimp
      by (metis Limit-def elts-succ imageI insertI1 mem-0-Ord mult-add-mem-0)
    show  $\exists y \in (\lambda x. \alpha \uparrow (\beta * x)) \text{ ' elts } \mu. x \leq y$ 
      if x  $\in (\bigcup x \in \text{elts } \mu. (\uparrow) \alpha \text{ ' elts } (\beta * x))$  for x
      using that  $\langle \text{Ord } \alpha \rangle \langle \text{Ord } \beta \rangle$  False Limit
      byclarsimp (metis Limit-def Ord-in-Ord Ord-mult VWO-TC-le mem-imp-VWO
oexp-mono)
  qed auto
  also have ... =  $\bigsqcup ((\uparrow) (\alpha \uparrow \beta) \text{ ' elts } (\bigsqcup \xi \in \text{elts } \mu. \xi))$ 
    using Limit.IH Limit.hyps by auto
  also have ... =  $(\alpha \uparrow \beta) \uparrow (\bigsqcup \xi \in \text{elts } \mu. \xi)$ 
    using False Limit.hyps oexp-Limit  $\langle \text{Ord } \beta \rangle$  by auto
  finally show ?case .
  qed
qed

lemma Limit-omega-oexp:
  assumes Ord  $\delta$   $\delta \neq 0$ 
  shows Limit  $(\omega \uparrow \delta)$ 
  using assms
proof (cases  $\delta$  rule: Ord-cases)
  case 0
  then show ?thesis
    using assms(2) by blast

```



```

next
case (succ l)
have *: succ β ∈ elts (ω↑l * n + ω↑l)
  if n: n ∈ elts ω and β: β ∈ elts (ω↑l * n) for n β
proof -
  obtain Ord n Ord β
  by (meson Ord-ω Ord-in-Ord Ord-mult Ord-oezp β n succ(1))
  obtain oo: Ord (ω↑l) Ord (ω↑l * n)
  by (simp add: ⟨Ord n⟩ succ(1))
  moreover have f4: β < ω↑l * n
  using oo Ord-mem-iff-lt ⟨Ord β⟩ ⟨β ∈ elts (ω↑l * n)⟩ by blast
  moreover have f5: Ord (succ β)
  using ⟨Ord β⟩ by blast
  moreover have ω↑l ≠ 0
  using oexp-eq-0-iff omega-nonzero succ(1) by blast
  ultimately show ?thesis
  by (metis add-less-cancel-left Ord-ω Ord-add Ord-mem-iff-lt OrdmemD ⟨Ord β⟩
  add.right-neutral dual-order.strict-trans2 oexp-gt-0-iff succ(1) succ-le-iff zero-in-omega)
qed
show ?thesis
  using succ
  apply (clarsimp simp: Limit-def mem-0-Ord)
  apply (simp add: mult-Limit)
  by (metis * mult-succ succ-in-omega)
next
case limit
then show ?thesis
  by (metis Limit-oezp Ord-ω OrdmemD one-V-def succ-in-omega zero-in-omega)
qed

lemma oexp-mult-commute:
  fixes j::nat
  assumes Ord α
  shows (α ↑ j) * α = α * (α ↑ j)
proof -
  have (α ↑ j) * α = α ↑ (1 + ord-of-nat j)
  by (simp add: one-V-def)
  also have ... = α * (α ↑ j)
  by (simp add: assms oexp-add)
  finally show ?thesis .
qed

lemma oexp-ω-Limit: Limit β ⇒ ω↑β = (⊔ ξ ∈ elts β. ω↑ξ)
  by (simp add: oexp-Limit)

lemma ω-power-succ-gtr: Ord α ⇒ ω ↑ α * ord-of-nat n < ω ↑ succ α
  by (simp add: OrdmemD)

lemma countable-oezp:

```

```

assumes  $\nu: \alpha \in \text{elts } \omega 1$ 
shows  $\omega \uparrow \alpha \in \text{elts } \omega 1$ 
proof –
  have  $\text{Ord } \alpha$ 
    using  $\text{Ord-}\omega 1 \text{ Ord-in-Ord assms}$  by blast
  then show ?thesis
    using assms
  proof (induction rule: Ord-induct3)
    case 0
      then show ?case
        by (simp add: Ord-mem-iff-lt)
    next
      case (succ  $\alpha$ )
      then have countable ( $\text{elts } (\omega \uparrow \alpha * \omega)$ )
        by (simp add: succ-in-Limit-iff countable-mult less-}\omega 1\text{-imp-countable})
      then show ?case
        using Ord-mem-iff-lt countable-iff-less-}\omega 1\text{ succ.hyps} by auto
    next
      case (Limit  $\alpha$ )
      with  $\text{Ord-}\omega 1$  have countable ( $\bigcup \beta \in \text{elts } \alpha. \text{elts } (\omega \uparrow \beta)$ )  $\text{Ord } (\omega \uparrow \bigsqcup (\text{elts } \alpha))$ 
        by (force simp: Limit-def intro: Ord-trans less-}\omega 1\text{-imp-countable})+
      then have  $\omega \uparrow \bigsqcup (\text{elts } \alpha) < \omega 1$ 
        using Limit.hyps countable-iff-less-}\omega 1\text{ oexp-Limit} by fastforce
      then show ?case
        using Limit.hyps Limit-def Ord-mem-iff-lt by auto
  qed
qed

end

```

5 Cantor Normal Form

```

theory Cantor-NF
  imports Ordinal-Exp
begin

```

5.1 Cantor normal form

Lemma 5.1

```

lemma cnf-1:
  assumes  $\alpha: \alpha \in \text{elts } \beta \text{ Ord } \beta$  and  $m > 0$ 
  shows  $\omega \uparrow \alpha * \text{ord-of-nat } n < \omega \uparrow \beta * \text{ord-of-nat } m$ 
proof –
  have  $\dagger: \omega \uparrow \text{succ } \alpha \leq \omega \uparrow \beta$ 
    using Ord-mem-iff-less-TC assms oexp-mono succ-le-TC-iff by auto
  have  $\omega \uparrow \alpha * \text{ord-of-nat } n < \omega \uparrow \alpha * \omega$ 
    using Ord-in-Ord OrdmemD assms by auto
  also have  $\dots = \omega \uparrow \text{succ } \alpha$ 

```

using *Ord-in-Ord* α **by** *auto*
also have $\dots \leq \omega \uparrow \beta$
using \dagger **by** *blast*
also have $\dots \leq \omega \uparrow \beta * \text{ord-of-nat } m$
using $\langle m > 0 \rangle$ *le-mult* **by** *auto*
finally show *?thesis* .
qed

fun *Cantor-sum* **where**

Cantor-sum-Nil: *Cantor-sum* $[]$ *ms* = 0
| *Cantor-sum-Nil2*: *Cantor-sum* $(\alpha \# \alpha s)$ $[]$ = 0
| *Cantor-sum-Cons*: *Cantor-sum* $(\alpha \# \alpha s)$ $(m \# ms)$ = $(\omega \uparrow \alpha) * \text{ord-of-nat } m + \text{Cantor-sum } \alpha s$ *ms*

abbreviation *Cantor-dec* :: *V list* \Rightarrow *bool* **where**

Cantor-dec \equiv *sorted-wrt* $(>)$

lemma *Ord-Cantor-sum*:

assumes *List.set* $\alpha s \subseteq ON$

shows *Ord* (*Cantor-sum* αs *ms*)

using *assms*

proof (*induction* αs *arbitrary*: *ms*)

case (*Cons* *a* αs *ms*)

then show *?case*

by (*cases* *ms*) *auto*

qed *auto*

lemma *Cantor-dec-Cons-iff* [*simp*]: *Cantor-dec* $(\alpha \# \beta \# \beta s) \longleftrightarrow \beta < \alpha \wedge \text{Cantor-dec } (\beta \# \beta s)$

by *auto*

Lemma 5.2. The second and third premises aren't really necessary, but their removal requires quite a lot of work.

lemma *cnf-2*:

assumes *List.set* $(\alpha \# \alpha s) \subseteq ON$ *list.set* *ms* $\subseteq \{0 < ..\}$ *length* $\alpha s = \text{length } ms$

and *Cantor-dec* $(\alpha \# \alpha s)$

shows $\omega \uparrow \alpha > \text{Cantor-sum } \alpha s$ *ms*

using *assms*

proof (*induction* *ms* *arbitrary*: α αs)

case *Nil*

then obtain $\alpha 0$ **where** $\alpha 0: (\alpha \# \alpha s) = [\alpha 0]$

by (*metis* *length-0-conv*)

then have *Ord* $\alpha 0$

using *Nil.premis*(1) **by** *auto*

then show *?case*

using $\alpha 0$ *zero-less-Limit* **by** *auto*

next

```

case (Cons m1 ms)
then obtain  $\alpha 0 \alpha 1 \alpha s'$  where  $\alpha 0 1: (\alpha \# \alpha s) = \alpha 0 \# \alpha 1 \# \alpha s'$ 
  by (metis (no-types, lifting) Cons.premis(3) Suc-length-conv)
then have Ord  $\alpha 0$  Ord  $\alpha 1$ 
  using Cons.premis(1)  $\alpha 0 1$  by auto
have *:  $\omega \uparrow \alpha 0 * \text{ord-of-nat } 1 > \omega \uparrow \alpha 1 * \text{ord-of-nat } m1$ 
proof (rule cnf-1)
  show  $\alpha 1 \in \text{elts } \alpha 0$ 
    using Cons.premis  $\alpha 0 1$  by (simp add: Ord-mem-iff-lt  $\langle \text{Ord } \alpha 0 \rangle \langle \text{Ord } \alpha 1 \rangle$ )
qed (use  $\langle \text{Ord } \alpha 0 \rangle$  in auto)
show ?case
proof (cases ms)
  case Nil
  then show ?thesis
    using * one-V-def Cons.premis(3)  $\alpha 0 1$  by auto
next
  case (Cons m2 ms')
  then obtain  $\alpha 2 \alpha s''$  where  $\alpha 0 2: (\alpha \# \alpha s) = \alpha 0 \# \alpha 1 \# \alpha 2 \# \alpha s''$ 
    by (metis Cons.premis(3) Suc-length-conv  $\alpha 0 1$  length-tl list.sel(3))
  then have Ord  $\alpha 2$ 
    using Cons.premis(1) by auto
  have  $m1 > 0 \ m2 > 0$ 
    using Cons.premis Cons by auto
  have  $\omega \uparrow \alpha 1 * \text{ord-of-nat } m1 + \omega \uparrow \alpha 1 * \text{ord-of-nat } m2 = (\omega \uparrow \alpha 1 * \text{ord-of-nat } m1) * \text{ord-of-nat } 2$ 
    by (simp add: mult-succ eval-nat-numeral)
  also have ...  $< \omega \uparrow \alpha 0$ 
    using cnf-1 [of concl:  $\alpha 1 \ m1 * 2 \ \alpha 0 \ 1$ ] Cons.premis  $\alpha 0 1$  one-V-def
    by (simp add: mult.assoc ord-of-nat-mult Ord-mem-iff-lt)
  finally have II:  $\omega \uparrow \alpha 1 * \text{ord-of-nat } m1 + \omega \uparrow \alpha 1 * \text{ord-of-nat } m2 < \omega \uparrow \alpha 0$ 
    by simp
  have Cantor-sum (tl  $\alpha s$ ) ms  $< \omega \uparrow \text{hd } \alpha s$ 
  proof (rule Cons.IH)
    show Cantor-dec (hd  $\alpha s \#$  tl  $\alpha s$ )
      using  $\langle \text{Cantor-dec } (\alpha \# \alpha s) \rangle \ \alpha 0 1$  by auto
  qed (use Cons.premis  $\alpha 0 1$  in auto)
  then have Cantor-sum ( $\alpha 2 \# \alpha s''$ ) ms  $< \omega \uparrow \alpha 1$ 
    using  $\alpha 0 2$  by auto
  also have ...  $\leq \omega \uparrow \alpha 1 * \text{ord-of-nat } m1$ 
    by (simp add:  $\langle 0 < m1 \rangle$  le-mult)
  finally show ?thesis
    using II  $\alpha 0 2$  dual-order.strict-trans by fastforce
qed
qed

```

proposition Cantor-nf-exists:

assumes Ord α

obtains $\alpha s \ ms$ where List.set $\alpha s \subseteq ON$ list.set ms $\subseteq \{0 < ..\}$ length $\alpha s =$ length ms

```

    and Cantor-dec  $\alpha$ s
    and  $\alpha = \text{Cantor-sum } \alpha s \ ms$ 
using assms
proof (induction  $\alpha$  arbitrary: thesis rule: Ord-induct)
case (step  $\alpha$ )
show ?case
proof (cases  $\alpha = 0$ )
case True
have Cantor-sum [] [] = 0
by simp
with True show ?thesis
using length-pos-if-in-set step.premis subset-eq
by (metis length-0-conv not-gr-zero sorted-wrt.simps(1))
next
case False
define  $\alpha\text{hat}$  where  $\alpha\text{hat} \equiv \text{Sup } \{\gamma \in ON. \omega \uparrow \gamma \leq \alpha\}$ 
then have Ord  $\alpha\text{hat}$ 
using Ord-Sup assms by fastforce
have  $\bigwedge \xi. \llbracket \text{Ord } \xi; \omega \uparrow \xi \leq \alpha \rrbracket \implies \xi \leq \omega \uparrow \alpha$ 
by (metis Ord- $\omega$  OrdmemD le-oexp' order-trans step.hyps one-V-def succ-in-omega
zero-in-omega)
then have  $\{\gamma \in ON. \omega \uparrow \gamma \leq \alpha\} \subseteq \text{elts } (\text{succ } (\omega \uparrow \alpha))$ 
using Ord-mem-iff-lt step.hyps by force
then have sma: small  $\{\gamma \in ON. \omega \uparrow \gamma \leq \alpha\}$ 
by (meson down)
have  $le: \omega \uparrow \alpha\text{hat} \leq \alpha$ 
proof (rule ccontr)
assume  $\neg \omega \uparrow \alpha\text{hat} \leq \alpha$ 
then have  $\dagger: \alpha \in \text{elts } (\omega \uparrow \alpha\text{hat})$ 
by (meson Ord- $\omega$  Ord-linear2 Ord-mem-iff-lt Ord-oexp <Ord  $\alpha\text{hat}$ > step.hyps)
obtain  $\gamma$  where Ord  $\gamma$   $\omega \uparrow \gamma \leq \alpha$   $\alpha < \gamma$ 
using <Ord  $\alpha\text{hat}$ >
proof (cases  $\alpha\text{hat}$  rule: Ord-cases)
case 0
with  $\dagger$  show thesis
by (auto simp: False)
next
case (succ  $\beta$ )
have  $\text{succ } \beta \in \{\gamma \in ON. \omega \uparrow \gamma \leq \alpha\}$ 
by (rule succ-in-Sup-Ord) (use succ  $\alpha\text{hat}$ -def sma in auto)
then have  $\omega \uparrow \text{succ } \beta \leq \alpha$ 
by blast
with  $\dagger$  show thesis
using < $\neg \omega \uparrow \alpha\text{hat} \leq \alpha$ > succ by blast
next
case limit
with  $\dagger$  show thesis
apply (clarsimp simp: oexp-Limit  $\alpha\text{hat}$ -def)
by (meson Ord- $\omega$  Ord-in-Ord Ord-linear-le mem-not-refl oexp-mono-le

```

```

omega-nonzero vsubsetD)
  qed
  then show False
    by (metis Ord- $\omega$  OrdmemD leD le-less-trans le-oe $\omega$ ' one-V-def succ-in-omega
zero-in-omega)
  qed
  have False if  $\nexists M. \alpha < \omega \uparrow \alpha \text{hat} * \text{ord-of-nat } M$ 
  proof -
    have  $\dagger: \omega \uparrow \alpha \text{hat} * \text{ord-of-nat } M \leq \alpha$  for  $M$ 
      by (meson that Ord- $\omega$  Ord-linear2 Ord-mult Ord-oe $\omega$  Ord-ord-of-nat  $\langle \text{Ord } \alpha \text{hat} \rangle$ 
step.hyps)
    have  $\neg \omega \uparrow \text{succ } \alpha \text{hat} \leq \alpha$ 
      using Ord-mem-iff-lt  $\alpha \text{hat-def } \langle \text{Ord } \alpha \text{hat} \rangle$  sma elts-succ by blast
    then have  $\alpha < \omega \uparrow \text{succ } \alpha \text{hat}$ 
      by (meson Ord- $\omega$  Ord-linear2 Ord-oe $\omega$  Ord-succ  $\langle \text{Ord } \alpha \text{hat} \rangle$  step.hyps)
    also have  $\dots = \omega \uparrow \alpha \text{hat} * \omega$ 
      using  $\langle \text{Ord } \alpha \text{hat} \rangle$  oe $\omega$ -succ by blast
    also have  $\dots = \text{Sup } (\text{range } (\lambda m. \omega \uparrow \alpha \text{hat} * \text{ord-of-nat } m))$ 
      by (simp add: mult-Limit) (auto simp:  $\omega$ -def image-image)
    also have  $\dots \leq \alpha$ 
      using  $\dagger$  by blast
    finally show False
      by simp
  qed
  then obtain  $M$  where  $M: \omega \uparrow \alpha \text{hat} * \text{ord-of-nat } M > \alpha$ 
    by blast
  have bound:  $i \leq M$  if  $\omega \uparrow \alpha \text{hat} * \text{ord-of-nat } i \leq \alpha$  for  $i$ 
  proof -
    have  $\omega \uparrow \alpha \text{hat} * \text{ord-of-nat } i < \omega \uparrow \alpha \text{hat} * \text{ord-of-nat } M$ 
      using  $M$  dual-order.strict-trans2 that by blast
    then show ?thesis
      using  $\langle \text{Ord } \alpha \text{hat} \rangle$  less-V-def by auto
  qed
  define  $m \text{hat}$  where  $m \text{hat} \equiv \text{Greatest } (\lambda m. \omega \uparrow \alpha \text{hat} * \text{ord-of-nat } m \leq \alpha)$ 
  have  $m \text{hat-ge}: m \leq m \text{hat}$  if  $\omega \uparrow \alpha \text{hat} * \text{ord-of-nat } m \leq \alpha$  for  $m$ 
    unfolding  $m \text{hat-def}$ 
    by (metis (mono-tags, lifting) Greatest-le-nat bound that)
  have  $m \text{hat}: \omega \uparrow \alpha \text{hat} * \text{ord-of-nat } m \text{hat} \leq \alpha$ 
    unfolding  $m \text{hat-def}$ 
    by (rule GreatestI-nat [where  $k=0$  and  $b=M$ ]) (use bound in auto)
  then obtain  $\xi$  where Ord  $\xi \xi \leq \alpha$  and  $\xi: \alpha = \omega \uparrow \alpha \text{hat} * \text{ord-of-nat } m \text{hat} + \xi$ 
    by (metis Ord- $\omega$  Ord-mult Ord-oe $\omega$  Ord-ord-of-nat  $\langle \text{Ord } \alpha \text{hat} \rangle$  step.hyps
le-Ord-diff)
  have False if  $\xi = \alpha$ 
  proof -
    have  $\xi \geq \omega \uparrow \alpha \text{hat}$ 
      by (simp add: le that)
    then obtain  $\zeta$  where Ord  $\zeta \zeta \leq \xi$  and  $\zeta: \xi = \omega \uparrow \alpha \text{hat} + \zeta$ 
      by (metis Ord- $\omega$  Ord-oe $\omega$   $\langle \text{Ord } \alpha \text{hat} \rangle$   $\langle \text{Ord } \xi \rangle$  le-Ord-diff)

```

```

then have  $\alpha = \omega \uparrow \alpha \text{hat} * \text{ord-of-nat } m\text{hat} + \omega \uparrow \alpha \text{hat} + \zeta$ 
  by (simp add:  $\xi$  add.assoc)
then have  $\omega \uparrow \alpha \text{hat} * \text{ord-of-nat } (\text{Suc } m\text{hat}) \leq \alpha$ 
by (metis add-le-cancel-left add.right-neutral le-0 mult-succ ord-of-nat.simps(2))
then show False
  using Suc-n-not-le-n mhat-ge by blast
qed
then have  $\xi \text{in } \alpha: \xi \in \text{elts } \alpha$ 
  using Ord-mem-iff-lt  $\langle \text{Ord } \xi \rangle$   $\langle \xi \leq \alpha \rangle$  less-V-def step.hyps by auto
show thesis
proof (cases  $\xi = 0$ )
  case True
  show thesis
  proof (rule step.prems)
    show  $\alpha = \text{Cantor-sum } [\alpha \text{hat}] [m\text{hat}]$ 
    by (simp add: True  $\xi$ )
  qed (use  $\xi$  True  $\langle \alpha \neq 0 \rangle$   $\langle \text{Ord } \alpha \text{hat} \rangle$  in auto)
next
case False
obtain  $\beta s$   $ns$  where sub: List.set  $\beta s \subseteq ON$  list.set  $ns \subseteq \{0 < ..\}$ 
  and len-eq: length  $\beta s = \text{length } ns$ 
  and dec: Cantor-dec  $\beta s$ 
  and  $\xi \text{eq}: \xi = \text{Cantor-sum } \beta s$  ns
  using step.IH [OF  $\xi \text{in } \alpha]$  by blast
then have length  $\beta s > 0$  length  $ns > 0$ 
  using False Cantor-sum.simps(1)  $\langle \xi = \text{Cantor-sum } \beta s$  ns  $\rangle$  by auto
then obtain  $\beta 0$   $n0$   $\beta s'$   $ns'$  where  $\beta s = \beta 0 \# \beta s'$  and Ord  $\beta 0$ 
  and  $ns = n0 \# ns'$  and  $n0 > 0$ 
  using sub by (auto simp: neq-Nil-conv)
moreover have False if  $\beta 0 > \alpha \text{hat}$ 
proof –
  have  $\omega \uparrow \beta 0 \leq \omega \uparrow \beta 0 * \text{ord-of-nat } n0 + u$  for  $u$ 
    using  $\langle n0 > 0 \rangle$ 
  by (metis add-le-cancel-left Ord-ord-of-nat add.right-neutral dual-order.trans
gr-implies-not-zero le-0 le-mult ord-of-eq-0-iff)
  moreover have  $\omega \uparrow \beta 0 > \alpha$ 
    using that  $\langle \text{Ord } \beta 0 \rangle$ 
  by (metis (no-types, lifting) Ord- $\omega$  Ord-linear2 Ord-oexp Sup-upper  $\alpha \text{hat-def}$ 
leD mem-Collect-eq sma step.hyps)
  ultimately have  $\xi \geq \omega \uparrow \beta 0$ 
    by (simp add:  $\xi \text{eq } \beta 0$  n0)
  then show ?thesis
    using  $\langle \alpha < \omega \uparrow \beta 0 \rangle$   $\langle \xi \leq \alpha \rangle$  by auto
qed
ultimately have  $\beta 0 \leq \alpha \text{hat}$ 
  using Ord-linear2  $\langle \text{Ord } \alpha \text{hat} \rangle$  by auto
then consider  $\beta 0 < \alpha \text{hat} \mid \beta 0 = \alpha \text{hat}$ 
  using dual-order.order-iff-strict by auto
then show ?thesis

```

```

proof cases
  case 1
  show ?thesis
  proof (rule step.prem)
    show list.set (αhat#βs) ⊆ ON
      using sub by (auto simp: ⟨Ord αhat⟩)
    show list.set (mhat#ns) ⊆ {0::nat<..}
      using sub using ⟨ξ = α ⇒ False⟩ ξ by fastforce
    show Cantor-dec (αhat#βs)
      using that ⟨β0 < αhat⟩ ⟨Ord αhat⟩ ⟨Ord β0⟩ Ord-mem-iff-lt β0 dec
less-Suc-eq-0-disj
      by (force simp: β0 n0)
    show length (αhat#βs) = length (mhat#ns)
      by (auto simp: len-eq)
    show α = Cantor-sum (αhat#βs) (mhat#ns)
      by (simp add: ξ ξeq β0 n0)
  qed
next
  case 2
  show ?thesis
  proof (rule step.prem)
    show list.set βs ⊆ ON
      by (simp add: sub(1))
    show list.set ((n0+mhat)#ns') ⊆ {0::nat<..}
      using n0 sub(2) by auto
    show length (βs::V list) = length ((n0+mhat)#ns')
      by (simp add: len-eq n0)
    show Cantor-dec βs
      using that β0 dec by auto
    show α = Cantor-sum βs ((n0+mhat)#ns')
      using 2
      by (simp add: add-mult-distrib β0 ξ ξeq add.assoc add commute n0
ord-of-nat-add)
  qed
  qed
  qed
  qed
qed

```

lemma Cantor-sum-0E:

```

  assumes Cantor-sum αs ms = 0 List.set αs ⊆ ON list.set ms ⊆ {0<..} length
αs = length ms
  shows αs = []
  using assms
proof (induction αs arbitrary: ms)
  case Nil
  then show ?case
    by auto
next

```


case (*Cons a list*)
then obtain $m\ ms'$ **where** $ms = m\#\ms'$ $m \neq 0$ *list.set* $ms' \subseteq \{0<..\}$
by *simp* (*metis Suc-length-conv greaterThan-iff insert-subset list.set(2)*)
with Cons show ?*case* **by** *auto*
qed

lemma *Cantor-nf-unique-aux:*

assumes *Ord* α
and $\alpha s ON$: *List.set* $\alpha s \subseteq ON$
and $\beta s ON$: *List.set* $\beta s \subseteq ON$
and ms : *list.set* $ms \subseteq \{0<..\}$
and ns : *list.set* $ns \subseteq \{0<..\}$
and $mseq$: *length* $\alpha s = \text{length } ms$
and $nseq$: *length* $\beta s = \text{length } ns$
and $\alpha sdec$: *Cantor-dec* αs
and $\beta sdec$: *Cantor-dec* βs
and αseq : $\alpha = \text{Cantor-sum } \alpha s\ ms$
and βseq : $\alpha = \text{Cantor-sum } \beta s\ ns$
shows $\alpha s = \beta s \wedge ms = ns$
using *assms*
proof (*induction* α *arbitrary:* $\alpha s\ ms\ \beta s\ ns$ *rule: Ord-induct*)
case (*step* α)
show ?*case*
proof (*cases* $\alpha = 0$)
case *True*
then show ?*thesis*
using *step.prem*s **by** (*metis Cantor-sum-0E length-0-conv*)
next
case *False*
then obtain $\alpha 0\ \alpha s'\ \beta 0\ \beta s'$ **where** αs : $\alpha s = \alpha 0\ \#\ \alpha s'$ **and** βs : $\beta s = \beta 0\ \#\$
 $\beta s'$
by (*metis Cantor-sum.simps(1) min-list.cases step.prem*s(9,10))
then have ON : *Ord* $\alpha 0$ *list.set* $\alpha s' \subseteq ON$ *Ord* $\beta 0$ *list.set* $\beta s' \subseteq ON$
using $\alpha s\ \beta s$ *step.prem*s(1,2) **by** *auto*
then obtain $m 0\ ms'\ n 0\ ns'$ **where** ms : $ms = m 0\ \#\ ms'$ **and** ns : $ns = n 0\ \#\$
 ns'
by (*metis* $\alpha s\ \beta s$ *length-0-conv list.distinct(1) list.exhaust step.prem*s(5,6))
then have nz : $m 0 \neq 0$ *list.set* $ms' \subseteq \{0<..\}$ $n 0 \neq 0$ *list.set* $ns' \subseteq \{0<..\}$
using $ms\ ns$ *step.prem*s(3,4) **by** *auto*
have *False* **if** $\beta 0 < \alpha 0$
proof –
have *Ord*: *Ord* (*Cantor-sum* $\beta s\ ns$) *Ord* ($\omega \uparrow \alpha 0$)
using *Ord-oe*xp $\langle \text{Ord } \alpha 0 \rangle$ *step.hyps* *step.prem*s(10) **by** *blast+*
have $*$: *Cantor-sum* $\beta s\ ns < \omega \uparrow \alpha 0$
using *step.prem*s(2–6) $\langle \text{Ord } \alpha 0 \rangle$ $\langle \text{Cantor-dec } \beta s \rangle$ *that* βs *cnf-2*
by (*metis Cantor-dec-Cons-iff insert-subset list.set(2) mem-Collect-eq*)
then show *False*
by (*metis Cantor-sum-Cons Ord-mem-iff-lt Ord-ord-of-nat Ord*c αs $\langle m 0 \neq$

$0 \rangle * \text{le-mult } ms \text{ not-add-mem-right ord-of-eq-0 step.prem}(9,10) \text{ vsubset}D$
qed
moreover
have *False* **if** $\alpha 0 < \beta 0$
proof –
have *Ord*: *Ord* (*Cantor-sum* αs *ms*) *Ord* ($\omega \uparrow \beta 0$)
using *Ord-oexp* $\langle \text{Ord } \beta 0 \rangle$ *step.hyps* *step.prem}(9)* **by** *blast+*
have $*$: *Cantor-sum* αs *ms* $< \omega \uparrow \beta 0$
using *step.prem}(1–5)* $\langle \text{Ord } \beta 0 \rangle$ $\langle \text{Cantor-dec } \alpha s \rangle$ *that* αs *cnf-2*
by (*metis* *Cantor-dec-Cons-iff* βs *insert-subset* *list.set}(2))
then show *False*
by (*metis* *Cantor-sum-Cons* *Ord-mem-iff-lt* *Ord-ord-of-nat* *Ord* βs $\langle n 0 \neq$
 $0 \rangle * \text{le-mult not-add-mem-right } ns \text{ ord-of-eq-0 step.prem}(9,10) \text{ vsubset}D$
qed
ultimately have 1 : $\alpha 0 = \beta 0$
using *Ord-linear-lt* $\langle \text{Ord } \alpha 0 \rangle$ $\langle \text{Ord } \beta 0 \rangle$ **by** *blast*
have *False* **if** $m 0 < n 0$
proof –
have $\omega \uparrow \alpha 0 > \text{Cantor-sum } \alpha s' \text{ } ms'$
using αs $\langle \text{list.set } ms' \subseteq \{0 < ..\} \rangle$ *cnf-2* *ms* *step.prem}(1,5,7)* **by** *auto*
then have $\alpha < \omega \uparrow \alpha 0 * \text{ord-of-nat } m 0 + \omega \uparrow \alpha 0$
by (*simp* *add*: αs *ms* *step.prem}(9))
also have $\dots = \omega \uparrow \alpha 0 * \text{ord-of-nat } (\text{Suc } m 0)$
by (*simp* *add*: *mult-succ*)
also have $\dots \leq \omega \uparrow \alpha 0 * \text{ord-of-nat } n 0$
by (*meson* *Ord- ω* *Ord-oexp* *Ord-ord-of-nat* *Suc-leI* $\langle \text{Ord } \alpha 0 \rangle$ *mult-cancel-le-iff*
ord-of-nat-mono-iff *that*)
also have $\dots \leq \alpha$
by (*metis* *Cantor-sum-Cons* *add-le-cancel-left* βs $\langle \alpha 0 = \beta 0 \rangle$ *add.right-neutral*
le-0 *ns* *step.prem}(10))
finally show *False*
by *blast*
qed
moreover have *False* **if** $n 0 < m 0$
proof –
have $\omega \uparrow \beta 0 > \text{Cantor-sum } \beta s' \text{ } ns'$
using βs $\langle \text{list.set } ns' \subseteq \{0 < ..\} \rangle$ *cnf-2* *ns* *step.prem}(2,6,8)* **by** *auto*
then have $\alpha < \omega \uparrow \beta 0 * \text{ord-of-nat } n 0 + \omega \uparrow \beta 0$
by (*simp* *add*: βs *ns* *step.prem}(10))
also have $\dots = \omega \uparrow \beta 0 * \text{ord-of-nat } (\text{Suc } n 0)$
by (*simp* *add*: *mult-succ*)
also have $\dots \leq \omega \uparrow \beta 0 * \text{ord-of-nat } m 0$
by (*meson* *Ord- ω* *Ord-oexp* *Ord-ord-of-nat* *Suc-leI* $\langle \text{Ord } \beta 0 \rangle$ *mult-cancel-le-iff*
ord-of-nat-mono-iff *that*)
also have $\dots \leq \alpha$
by (*metis* *Cantor-sum-Cons* *add-le-cancel-left* αs $\langle \alpha 0 = \beta 0 \rangle$ *add.right-neutral*
le-0 *ms* *step.prem}(9))
finally show *False*
by *blast******

qed
ultimately have 2: $m0 = n0$
using *nat-neq-iff* **by** *blast*
have $\alpha s' = \beta s' \wedge ms' = ns'$
proof (*rule step.IH*)
have *Cantor-sum* $\alpha s' ms' < \omega \uparrow \alpha 0$
using αs *cnf-2* ms *nz(2)* *step.prem(1)* *step.prem(5)* *step.prem(7)* **by** *auto*
also have $\dots \leq \text{Cantor-sum } \alpha s ms$
apply (*simp add: \alpha s \beta s ms ns*)
by (*metis Cantor-sum-Cons add-less-cancel-left ON(1) Ord-\omega Ord-linear^2*
Ord-oexp Ord-ord-of-nat \alpha s add.right-neutral dual-order.strict-trans1 le-mult ms
not-less-0 nz(1) ord-of-eq-0 step.hyps step.prem(9))
finally show *Cantor-sum* $\alpha s' ms' \in \text{elts } \alpha$
using *ON(2)* *Ord-Cantor-sum* *Ord-mem-iff-lt* *step.hyps* *step.prem(9)* **by**
blast
show $\text{length } \alpha s' = \text{length } ms' \text{ length } \beta s' = \text{length } ns'$
using $\alpha s ms \beta s ns$ *step.prem* **by** *auto*
show *Cantor-dec* $\alpha s' \text{Cantor-dec } \beta s'$
using $\alpha s \beta s$ *step.prem(7,8)* **by** *auto*
have *Cantor-sum* $\alpha s ms = \text{Cantor-sum } \beta s ns$
using *step.prem(9,10)* **by** *auto*
then show *Cantor-sum* $\alpha s' ms' = \text{Cantor-sum } \beta s' ns'$
using 1 2 **by** (*simp add: \alpha s \beta s ms ns*)
qed (*use ON nz in auto*)
then show *?thesis*
using 1 2 **by** (*simp add: \alpha s \beta s ms ns*)
qed
qed

proposition *Cantor-nf-unique*:
assumes *Cantor-sum* $\alpha s ms = \text{Cantor-sum } \beta s ns$
and $\alpha s ON: \text{List.set } \alpha s \subseteq ON$
and $\beta s ON: \text{List.set } \beta s \subseteq ON$
and $ms: \text{list.set } ms \subseteq \{0<..\}$
and $ns: \text{list.set } ns \subseteq \{0<..\}$
and $mseq: \text{length } \alpha s = \text{length } ms$
and $nseq: \text{length } \beta s = \text{length } ns$
and $\alpha sdec: \text{Cantor-dec } \alpha s$
and $\beta sdec: \text{Cantor-dec } \beta s$
shows $\alpha s = \beta s \wedge ms = ns$
using *Cantor-nf-unique-aux* *Ord-Cantor-sum* *assms* **by** *auto*

lemma *less-\omega-power*:
assumes *Ord* $\alpha 1$ *Ord* β
and $\alpha 2: \alpha 2 \in \text{elts } \alpha 1$ **and** $\beta: \beta < \omega \uparrow \alpha 2$
and $m1 > 0$ $m2 > 0$
shows $\omega \uparrow \alpha 2 * \text{ord-of-nat } m2 + \beta < \omega \uparrow \alpha 1 * \text{ord-of-nat } m1 + (\omega \uparrow \alpha 2 * \text{ord-of-nat } m2)$

$m2 + \beta$
 (is ?lhs < ?rhs)
proof –
obtain oo: Ord ($\omega \uparrow \alpha 1$) Ord ($\omega \uparrow \alpha 2$)
using Ord-in-Ord Ord-oevp *assms* **by** blast
moreover obtain ord-of-nat $m2 \neq 0$
using *assms* ord-of-eq-0 **by** blast
ultimately have $\beta < \omega \uparrow \alpha 2 * \text{ord-of-nat } m2$
by (meson Ord-ord-of-nat β dual-order.strict-trans1 le-mult)
with oo *assms* **have** ?lhs \neq ?rhs
by (metis Ord-mult Ord-ord-of-nat add-strict-mono add.assoc cnf-1 not-add-less-right oo)
then show ?thesis
by (simp add: add-le-left ‹Ord β › less-V-def oo)
qed

lemma Cantor-sum-ge:
assumes List.set ($\alpha \# \alpha s$) \subseteq ON list.set $ms \subseteq \{0 < ..\}$ length $ms > 0$
shows $\omega \uparrow \alpha \leq \text{Cantor-sum } (\alpha \# \alpha s) \ ms$
proof –
obtain m ns **where** $ms: ms = \text{Cons } m \ ns$
by (meson *assms*(β) list.set-cases nth-mem)
then have $\omega \uparrow \alpha \leq \omega \uparrow \alpha * \text{ord-of-nat } m$
using *assms*(β) le-mult **by** auto
then show ?thesis
using dual-order.trans ms **by** auto
qed

5.2 Simplified Cantor normal form

No coefficients, and the exponents decreasing non-strictly

fun ω -sum **where**
 ω -sum-Nil: ω -sum [] = 0
 ω -sum-Cons: ω -sum ($\alpha \# \alpha s$) = ($\omega \uparrow \alpha$) + ω -sum αs

abbreviation ω -dec :: V list \Rightarrow bool **where**
 ω -dec \equiv sorted-wrt (\geq)

lemma Ord- ω -sum [simp]: List.set $\alpha s \subseteq$ ON \Longrightarrow Ord (ω -sum αs)
by (induction αs) auto

lemma ω -dec-Cons-iff [simp]: ω -dec ($\alpha \# \beta \# \beta s$) \longleftrightarrow $\beta \leq \alpha \wedge \omega$ -dec ($\beta \# \beta s$)
by auto

lemma ω -sum-0E:
assumes ω -sum $\alpha s = 0$ List.set $\alpha s \subseteq$ ON
shows $\alpha s = []$
using *assms*
by (induction αs) auto

```

fun  $\omega$ -of-Cantor where
   $\omega$ -of-Cantor-Nil:  $\omega$ -of-Cantor []  $ms$  = []
|  $\omega$ -of-Cantor-Nil2:  $\omega$ -of-Cantor ( $\alpha\#\alpha s$ ) [] = []
|  $\omega$ -of-Cantor-Cons:  $\omega$ -of-Cantor ( $\alpha\#\alpha s$ ) ( $m\#ms$ ) = replicate  $m$   $\alpha$  @  $\omega$ -of-Cantor
 $\alpha s$   $ms$ 

```

```

lemma  $\omega$ -sum-append [simp]:  $\omega$ -sum ( $xs$  @  $ys$ ) =  $\omega$ -sum  $xs$  +  $\omega$ -sum  $ys$ 
by (induction  $xs$ ) (auto simp: add.assoc)

```

```

lemma  $\omega$ -sum-replicate [simp]:  $\omega$ -sum (replicate  $m$   $a$ ) =  $\omega$   $\uparrow$   $a$  * ord-of-nat  $m$ 
by (induction  $m$ ) (auto simp: mult-succ simp flip: replicate-append-same)

```

```

lemma  $\omega$ -sum-of-Cantor [simp]:  $\omega$ -sum ( $\omega$ -of-Cantor  $\alpha s$   $ms$ ) = Cantor-sum  $\alpha s$ 
 $ms$ 
proof (induction  $\alpha s$  arbitrary: ms)
case (Cons  $a$   $\alpha s$   $ms$ )
then show ?case
by (cases  $ms$ ) auto
qed auto

```

```

lemma  $\omega$ -of-Cantor-subset: List.set ( $\omega$ -of-Cantor  $\alpha s$   $ms$ )  $\subseteq$  List.set  $\alpha s$ 
proof (induction  $\alpha s$  arbitrary: ms)
case (Cons  $a$   $\alpha s$   $ms$ )
then show ?case
by (cases  $ms$ ) auto
qed auto

```

```

lemma  $\omega$ -dec-replicate:  $\omega$ -dec (replicate  $m$   $\alpha$  @  $\alpha s$ ) = (if  $m=0$  then  $\omega$ -dec  $\alpha s$  else
 $\omega$ -dec ( $\alpha\#\alpha s$ ))
by (induction  $m$  arbitrary:  $\alpha s$ ) (simp-all flip: replicate-append-same)

```

```

lemma  $\omega$ -dec-of-Cantor-aux:
assumes Cantor-dec ( $\alpha\#\alpha s$ ) length  $\alpha s$  = length  $ms$ 
shows  $\omega$ -dec ( $\omega$ -of-Cantor ( $\alpha\#\alpha s$ ) ( $m\#ms$ ))
using assms
proof (induction  $\alpha s$  arbitrary: ms)
case Nil
then show ?case
using sorted-wrt-iff-nth-less by fastforce
next
case (Cons  $a$   $\alpha s$   $ms$ )
then obtain  $n$   $ns$  where  $ms$  =  $n\#ns$ 
by (metis length-Suc-conv)
then have  $a \leq \alpha$ 
using Cons.prem1 order.strict-implies-order by auto

```

moreover have $\forall x \in \text{list.set } (\omega\text{-of-Cantor } \alpha s \text{ } ns). x \leq a$
using *Cons ns <a ≤ α>*
apply (*simp add: ω-dec-replicate*)
by (*meson ω-of-Cantor-subset order.strict-implies-order subsetD*)
ultimately show *?case*
using *Cons ns by (force simp: ω-dec-replicate)*
qed

lemma *ω-dec-of-Cantor*:
assumes *Cantor-dec α s length α s = length m s*
shows *ω-dec (ω-of-Cantor α s m s)*
proof (*cases α s*)
case *Nil*
then have *m s = []*
using *assms by auto*
with *Nil show ?thesis*
by *simp*
next
case (*Cons a list*)
then show *?thesis*
by (*metis ω-dec-of-Cantor-aux assms length-Suc-conv*)
qed

proposition *ω-nf-exists*:
assumes *Ord α*
obtains *α s where List.set α s ⊆ ON and ω-dec α s and α = ω-sum α s*
proof –
obtain *α s m s where List.set α s ⊆ ON list.set m s ⊆ {0<..} and length: length α s = length m s*
and *Cantor-dec α s*
and *α: α = Cantor-sum α s m s*
using *Cantor-nf-exists assms by blast*
then show *thesis*
by (*metis ω-dec-of-Cantor ω-of-Cantor-subset ω-sum-of-Cantor order-trans that*)
qed

lemma *ω-sum-take-drop*: *ω-sum α s = ω-sum (take k α s) + ω-sum (drop k α s)*
proof (*induction k arbitrary: α s*)
case *0*
then show *?case*
by *simp*
next
case (*Suc k*)
then show *?case*
proof (*cases α s*)
case *Nil*
then show *?thesis*
by *simp*

```

next
  case (Cons a list)
  with Suc.premis show ?thesis
  by (simp add: add.assoc flip: Suc.IH)
qed
qed

lemma in-elts- $\omega$ -sum:
  assumes  $\delta \in \text{elts } (\omega\text{-sum } \alpha s)$ 
  shows  $\exists k < \text{length } \alpha s. \exists \gamma \in \text{elts } (\omega \uparrow (\alpha s!k)). \delta = \omega\text{-sum } (\text{take } k \alpha s) + \gamma$ 
  using assms
proof (induction  $\alpha s$  arbitrary:  $\delta$ )
  case (Cons  $\alpha \alpha s$ )
  then have  $\delta \in \text{elts } (\omega \uparrow \alpha + \omega\text{-sum } \alpha s)$ 
  by simp
  then show ?case
  proof (rule mem-plus-V-E)
    fix  $\eta$ 
    assume  $\eta: \eta \in \text{elts } (\omega\text{-sum } \alpha s)$  and  $\delta: \delta = \omega \uparrow \alpha + \eta$ 
    then obtain  $k \gamma$  where  $k < \text{length } \alpha s$   $\gamma \in \text{elts } (\omega \uparrow (\alpha s!k))$   $\eta = \omega\text{-sum } (\text{take } k$ 
 $\alpha s) + \gamma$ 
    using Cons.IH by blast
    then show ?case
    by (rule-tac  $x = \text{Suc } k$  in exI) (simp add:  $\delta$  add.assoc)
  qed auto
qed auto

lemma  $\omega$ -le- $\omega$ -sum:  $\llbracket k < \text{length } \alpha s; \text{List.set } \alpha s \subseteq \text{ON} \rrbracket \implies \omega \uparrow (\alpha s!k) \leq \omega\text{-sum } \alpha s$ 
proof (induction  $\alpha s$  arbitrary:  $k$ )
  case (Cons a  $\alpha s$ )
  then obtain Ord a list.set  $\alpha s \subseteq \text{ON}$ 
  by simp
  with Cons.IH have  $\bigwedge k x. k < \text{length } \alpha s \implies \omega \uparrow \alpha s!k \leq \omega \uparrow a + \omega\text{-sum } \alpha s$ 
  by (meson Ord- $\omega$  Ord- $\omega$ -sum Ord-oexp add-le-left order-trans)
  then show ?case
  using Cons by (simp add: nth-Cons split: nat.split)
qed auto

lemma  $\omega$ -sum-less-self:
  assumes List.set  $(\alpha \# \alpha s) \subseteq \text{ON}$  and  $\omega\text{-dec } (\alpha \# \alpha s)$ 
  shows  $\omega\text{-sum } \alpha s < \omega \uparrow \alpha + \omega\text{-sum } \alpha s$ 
  using assms
proof (induction  $\alpha s$  arbitrary:  $\alpha$ )
  case Nil
  then show ?case
  using ZFC-in-HOL.neq0-conv by fastforce
next
  case (Cons  $\alpha 1 \alpha s$ )

```

then show *?case*
by (*simp add: add-right-strict-mono oexp-mono-le*)
qed

Something like Lemma 5.2 for ω -sum

lemma *ω -sum-less- ω -power:*
assumes *ω -dec* ($\alpha \# \alpha s$) *List.set* ($\alpha \# \alpha s$) $\subseteq ON$
shows *ω -sum* $\alpha s < \omega \uparrow \alpha * \omega$
using *assms*
proof (*induction* αs)
case *Nil*
then show *?case*
by (*simp add: ω -gt0*)
next
case (*Cons* $\beta \alpha s$)
then have *Ord* α
by *auto*
have *ω -sum* $\alpha s < \omega \uparrow \alpha * \omega$
using *Cons* **by** *force*
then have $\omega \uparrow \beta + \omega$ -sum $\alpha s < \omega \uparrow \alpha + \omega \uparrow \alpha * \omega$
using *Cons.prem*s *add-right-strict-mono oexp-mono-le* **by** *auto*
also have $\dots = \omega \uparrow \alpha * \omega$
by (*metis Kirby.add-mult-distrib mult.right-neutral one-plus- ω -equals- ω*)
finally show *?case*
by *simp*
qed

lemma *ω -sum-nf-unique-aux:*
assumes *Ord* α
and $\alpha s ON$: *List.set* $\alpha s \subseteq ON$
and $\beta s ON$: *List.set* $\beta s \subseteq ON$
and $\alpha s dec$: *ω -dec* αs
and $\beta s dec$: *ω -dec* βs
and αseq : $\alpha = \omega$ -sum αs
and βseq : $\alpha = \omega$ -sum βs
shows $\alpha s = \beta s$
using *assms*
proof (*induction* α *arbitrary: $\alpha s \beta s$ rule: Ord-induct*)
case (*step* α)
show *?case*
proof (*cases* $\alpha = 0$)
case *True*
then show *?thesis*
using *step.prem*s **by** (*metis ω -sum-0E*)
next
case *False*
then obtain $\alpha 0 \alpha s' \beta 0 \beta s'$ **where** αs : $\alpha s = \alpha 0 \# \alpha s'$ **and** βs : $\beta s = \beta 0 \# \beta s'$
 $\beta s'$


```

    by (metis  $\omega$ -sum.elims step.prem(5,6))
  then have ON: Ord  $\alpha 0$  list.set  $\alpha s' \subseteq ON$  Ord  $\beta 0$  list.set  $\beta s' \subseteq ON$ 
    using  $\alpha s \beta s$  step.prem(1,2) by auto
  have False if  $\beta 0 < \alpha 0$ 
  proof -
    have Ord $\alpha$ : Ord ( $\omega$ -sum  $\beta s$ ) Ord ( $\omega \uparrow \alpha 0$ )
      using Ord-oe $\alpha$   $\langle$ Ord  $\alpha 0$  $\rangle$  step.hyps step.prem(6) by blast+
    have  $\omega$ -sum  $\beta s < \omega \uparrow \beta 0 * \omega$ 
      by (rule  $\omega$ -sum-less- $\omega$ -power) (use  $\beta s$  step.prem ON in auto)
    also have  $\dots \leq \omega \uparrow \alpha 0$ 
      using ON by (metis Ord- $\omega$  Ord-succ oe $\alpha$ -mono-le oe $\alpha$ -succ omega-nonzero
succ-le-iff that)
    finally show False
      using  $\alpha s$  leD step.prem(5,6) by auto
  qed
  moreover
  have False if  $\alpha 0 < \beta 0$ 
  proof -
    have Ord $\alpha$ : Ord ( $\omega$ -sum  $\alpha s$ ) Ord ( $\omega \uparrow \beta 0$ )
      using Ord-oe $\alpha$   $\langle$ Ord  $\beta 0$  $\rangle$  step.hyps step.prem(5) by blast+
    have  $\omega$ -sum  $\alpha s < \omega \uparrow \alpha 0 * \omega$ 
      by (rule  $\omega$ -sum-less- $\omega$ -power) (use  $\alpha s$  step.prem ON in auto)
    also have  $\dots \leq \omega \uparrow \beta 0$ 
      using ON by (metis Ord- $\omega$  Ord-succ oe $\alpha$ -mono-le oe $\alpha$ -succ omega-nonzero
succ-le-iff that)
    finally show False
      using  $\beta s$  leD step.prem(5,6)
      by (simp add:  $\langle$  $\alpha = \omega$ -sum  $\alpha s$  $\rangle$  leD)
  qed
  ultimately have  $\dagger$ :  $\alpha 0 = \beta 0$ 
    using Ord-linear-lt  $\langle$ Ord  $\alpha 0$  $\rangle$   $\langle$ Ord  $\beta 0$  $\rangle$  by blast
  moreover have  $\alpha s' = \beta s'$ 
  proof (rule step.IH)
    show  $\omega$ -sum  $\alpha s' \in$  elts  $\alpha$ 
      using step.prem  $\alpha s$ 
      by (simp add: Ord-mem-iff-lt  $\omega$ -sum-less-self)
    show  $\omega$ -dec  $\alpha s' \omega$ -dec  $\beta s'$ 
      using  $\alpha s \beta s$  step.prem(3,4) by auto
    have  $\omega$ -sum  $\alpha s = \omega$ -sum  $\beta s$ 
      using step.prem(5,6) by auto
    then show  $\omega$ -sum  $\alpha s' = \omega$ -sum  $\beta s'$ 
      by (simp add:  $\dagger$   $\alpha s \beta s$ )
  qed (use ON in auto)
  ultimately show ?thesis
    by (simp add:  $\alpha s \beta s$ )
  qed
qed

```

5.3 Indecomposable ordinals

Cf exercise 5 on page 43 of Kunen

definition *indecomposable*

where *indecomposable* $\alpha \equiv \text{Ord } \alpha \wedge (\forall \beta \in \text{elts } \alpha. \forall \gamma \in \text{elts } \alpha. \beta + \gamma \in \text{elts } \alpha)$

lemma *indecomposableD*:

$\llbracket \text{indecomposable } \alpha; \beta < \alpha; \gamma < \alpha; \text{Ord } \beta; \text{Ord } \gamma \rrbracket \implies \beta + \gamma < \alpha$

by (*meson Ord-mem-iff-lt OrdmemD indecomposable-def*)

lemma *indecomposable-imp-Ord*:

indecomposable $\alpha \implies \text{Ord } \alpha$

using *indecomposable-def* **by** *blast*

lemma *indecomposable-1: indecomposable 1*

by (*auto simp: indecomposable-def*)

lemma *indecomposable-0: indecomposable 0*

by (*auto simp: indecomposable-def*)

lemma *indecomposable-succ* [*simp*]: *indecomposable* (*succ* α) $\longleftrightarrow \alpha = 0$

using *not-add-mem-right*

apply (*auto simp: indecomposable-def*)

apply (*metis add-right-cancel add.right-neutral*)

done

lemma *indecomposable-alt*:

assumes *ord*: *Ord* α *Ord* β **and** $\beta < \alpha$ **and** *minor*: $\bigwedge \beta \gamma. \llbracket \beta < \alpha; \gamma < \alpha; \text{Ord } \gamma \rrbracket \implies \beta + \gamma < \alpha$

shows $\beta + \alpha = \alpha$

proof –

have $\neg \beta + \alpha < \alpha$

by (*simp add: add-le-left ord leD*)

moreover have $\neg \alpha < \beta + \alpha$

by (*metis assms le-Ord-diff less-V-def*)

ultimately show *?thesis*

by (*simp add: add-le-left less-V-def ord*)

qed

lemma *indecomposable-imp-eq*:

assumes *indecomposable* α *Ord* β $\beta < \alpha$

shows $\beta + \alpha = \alpha$

by (*metis assms indecomposableD indecomposable-def le-Ord-diff less-V-def less-irrefl*)

lemma *indecomposable2*:

assumes *y*: $y < x$ **and** *z*: $z < x$ **and** *minor*: $\bigwedge y::V. y < x \implies y + x = x$

shows $y + z < x$

by (*metis add-less-cancel-left y z minor*)

```

lemma indecomposable-imp-Limit:
  assumes indec: indecomposable  $\alpha$  and  $\alpha > 1$ 
  shows Limit  $\alpha$ 
  using indecomposable-imp-Ord [OF indec]
proof (cases rule: Ord-cases)
  case (succ  $\beta$ )
  then show ?thesis
    using assms one-V-def by auto
qed (use assms in auto)

lemma eq-imp-indecomposable:
  assumes Ord  $\alpha \wedge \beta::V. \beta \in \text{elts } \alpha \implies \beta + \alpha = \alpha$ 
  shows indecomposable  $\alpha$ 
  by (metis add-mem-right-cancel assms indecomposable-def)

lemma indecomposable- $\omega$ -power:
  assumes Ord  $\delta$ 
  shows indecomposable  $(\omega \uparrow \delta)$ 
  unfolding indecomposable-def
proof (intro conjI ballI)
  show Ord  $(\omega \uparrow \delta)$ 
    by (simp add: <Ord  $\delta$ >)
next
  fix  $\beta \gamma$ 
  assume asm:  $\beta \in \text{elts } (\omega \uparrow \delta) \ \gamma \in \text{elts } (\omega \uparrow \delta)$ 
  then obtain ord: Ord  $\beta$  Ord  $\gamma$  and  $\beta < \omega \uparrow \delta$  and  $\gamma < \omega \uparrow \delta$ 
    by (meson Ord- $\omega$  Ord-in-Ord Ord-oexp OrdmemD <Ord  $\delta$ >)
  show  $\beta + \gamma \in \text{elts } (\omega \uparrow \delta)$ 
    using <Ord  $\delta$ >
  proof (cases  $\delta$  rule: Ord-cases)
  case 0
  then show ?thesis
    using <Ord  $\delta$ > asm by auto
next
  case (succ  $l$ )
  have  $\exists x \in \text{elts } \omega. \beta + \gamma \in \text{elts } (\omega \uparrow l * x)$ 
    if  $x \in \text{elts } \omega \ \beta \in \text{elts } (\omega \uparrow l * x)$  and  $y \in \text{elts } \omega \ \gamma \in \text{elts } (\omega \uparrow l * y)$ 
    for  $x \ y$ 
  proof –
    obtain Ord  $x$  Ord  $y$  Ord  $(\omega \uparrow l * x)$  Ord  $(\omega \uparrow l * y)$ 
      using Ord- $\omega$  Ord-mult Ord-oexp x y nat-into-Ord succ(1) by presburger
    then have  $\beta + \gamma \in \text{elts } (\omega \uparrow l * (x+y))$ 
      using add-mult-distrib Ord-add Ord-mem-iff-lt add-strict-mono ord x y by
presburger
    then show ?thesis
      using  $x \ y$  by blast
  qed
  then show ?thesis
    using <Ord  $\delta$ > succ ord  $\beta \gamma$  by (auto simp: mult-Limit simp flip: Ord-mem-iff-lt)

```

```

next
  case limit
  have Ord ( $\omega\uparrow\delta$ )
    by (simp add:  $\langle \text{Ord } \delta \rangle$ )
  then obtain x y where x:  $x \in \text{elts } \delta$  Ord x  $\beta \in \text{elts } (\omega\uparrow x)$ 
    and y:  $y \in \text{elts } \delta$  Ord y  $\gamma \in \text{elts } (\omega\uparrow y)$ 
    using  $\langle \text{Ord } \delta \rangle$  limit ord  $\beta$   $\gamma$  oexp-Limit
    by (auto simp flip: Ord-mem-iff-lt intro: Ord-in-Ord)
  then have succ ( $x \sqcup y$ )  $\in \text{elts } \delta$ 
    by (metis Limit-def Ord-linear-le limit sup.absorb2 sup.orderE)
  moreover have  $\beta + \gamma \in \text{elts } (\omega\uparrow \text{succ } (x \sqcup y))$ 
  proof -
    have oxy: Ord ( $x \sqcup y$ )
      using Ord-sup x y by blast
    then obtain  $\omega\uparrow x \leq \omega\uparrow(x \sqcup y)$   $\omega\uparrow y \leq \omega\uparrow(x \sqcup y)$ 
      by (metis Ord- $\omega$  Ord-linear-le Ord-mem-iff-less-TC Ord-mem-iff-lt le-TC-def
less-le-not-le oexp-mono omega-nonzero sup.absorb2 sup.orderE x(2) y(2))
    then have  $\beta \in \text{elts } (\omega\uparrow(x \sqcup y))$   $\gamma \in \text{elts } (\omega\uparrow(x \sqcup y))$ 
      using x y by blast+
    then have  $\beta + \gamma \in \text{elts } (\omega\uparrow(x \sqcup y) * \text{succ } (\text{succ } 0))$ 
      by (metis Ord- $\omega$  Ord-add Ord-mem-iff-lt Ord-oexp Ord-sup add-strict-mono
mult.right-neutral mult-succ ord one-V-def x(2) y(2))
    then show ?thesis
      apply (simp add: oxy)
      using Ord- $\omega$  Ord-mult Ord-oexp Ord-trans mem-0-Ord mult-add-mem-0
oexp-eq-0-iff omega-nonzero oxy succ-in-omega by presburger
    qed
    ultimately show ?thesis
      using ord  $\langle \text{Ord } (\omega\uparrow\delta) \rangle$  limit oexp-Limit by auto
  qed
qed

```

lemma *ω -power-imp-eq*:

```

  assumes  $\beta < \omega\uparrow\delta$  Ord  $\beta$  Ord  $\delta$   $\delta \neq 0$ 
  shows  $\beta + \omega\uparrow\delta = \omega\uparrow\delta$ 
  by (simp add: assms indecomposable- $\omega$ -power indecomposable-imp-eq)

```

lemma *mult-oexp-indec*: $\llbracket \text{Ord } \alpha; \text{Limit } \mu; \text{indecomposable } \mu \rrbracket \implies \alpha * (\alpha \uparrow \mu) = (\alpha \uparrow \mu)$

```

  by (metis Limit-def Ord-1 OrdmemD indecomposable-imp-eq oexp-1-right oexp-add
one-V-def)

```

lemma *mult-oexp- ω* : *Ord* $\alpha \implies \alpha * (\alpha \uparrow \omega) = (\alpha \uparrow \omega)$

```

  by (metis Ord-1 Ord- $\omega$  oexp-1-right oexp-add one-plus- $\omega$ -equals- $\omega$ )

```

lemma *type-imp-indecomposable*:

```

  assumes  $\alpha$ : Ord  $\alpha$ 
  and minor:  $\bigwedge X. X \subseteq \text{elts } \alpha \implies \text{ordertype } X \text{ VWF} = \alpha \vee \text{ordertype } (\text{elts } \alpha - X) \text{ VWF} = \alpha$ 

```

```

shows indecomposable  $\alpha$ 
unfolding indecomposable-def
proof (intro conjI ballI)
  fix  $\beta \ \gamma$ 
  assume  $\beta: \beta \in \text{elts } \alpha$  and  $\gamma: \gamma \in \text{elts } \alpha$ 
  then obtain  $\beta\gamma: \text{elts } \beta \subseteq \text{elts } \alpha \ \text{elts } \gamma \subseteq \text{elts } \alpha \ \text{Ord } \beta \ \text{Ord } \gamma$ 
    using  $\alpha \ \text{Ord-in-Ord} \ \text{Ord-trans}$  by blast
  then have oeq: ordertype (elts  $\beta$ ) VWF =  $\beta$ 
    by auto
  show  $\beta + \gamma \in \text{elts } \alpha$ 
  proof (rule ccontr)
    assume  $\beta + \gamma \notin \text{elts } \alpha$ 
    then obtain  $\delta$  where  $\delta: \text{Ord } \delta \ \beta + \delta = \alpha$ 
      by (metis Ord-ordertype  $\beta\gamma(1)$  le-Ord-diff less-eq-V-def minor oeq)
    then have  $\delta \in \text{elts } \alpha$ 
      using Ord-linear  $\beta\gamma \ \langle \beta + \gamma \notin \text{elts } \alpha \rangle$  by blast
    then have ordertype (elts  $\alpha - \text{elts } \beta$ ) VWF =  $\delta$ 
      using  $\delta$  ordertype-diff Limit-def  $\alpha \ \langle \text{Ord } \beta \rangle$  by blast
    then show False
      by (metis  $\beta \ \langle \delta \in \text{elts } \alpha \rangle \ \langle \text{elts } \beta \subseteq \text{elts } \alpha \rangle$  oeq mem-not-refl minor)
  qed
qed (use assms in auto)

```

This proof uses Cantor normal form, yet still is rather long

```

proposition indecomposable-is- $\omega$ -power:
  assumes inc: indecomposable  $\mu$ 
  obtains  $\mu = 0 \mid \delta$  where Ord  $\delta \ \mu = \omega \uparrow \delta$ 
proof (cases  $\mu = 0$ )
  case True
  then show thesis
    by (simp add: that)
next
  case False
  obtain Ord  $\mu$ 
    using Limit-def assms indecomposable-def by blast
  then obtain  $\alpha s \ ms$  where Cantor: List.set  $\alpha s \subseteq \text{ON list.set } ms \subseteq \{0 < ..\}$ 
    length  $\alpha s = \text{length } ms$  Cantor-dec  $\alpha s$ 
    and  $\mu: \mu = \text{Cantor-sum } \alpha s \ ms$ 
    using Cantor-nf-exists by blast
  consider (0) length  $\alpha s = 0 \mid$  (1) length  $\alpha s = 1 \mid$  (2) length  $\alpha s \geq 2$ 
    by linarith
  then show thesis
proof cases
  case 0
  then show ?thesis
    using  $\mu$  assms False indecomposable-def by auto
next
  case 1
  then obtain  $\alpha \ m$  where  $\alpha m: \alpha s = [\alpha] \ ms = [m]$ 

```

by (*metis One-nat-def* $\langle \text{length } \alpha s = \text{length } ms \rangle$ *length-0-conv* *length-Suc-conv*)
then obtain $\text{Ord } \alpha \ m \neq 0 \ \text{Ord } (\omega \uparrow \alpha)$
using $\langle \text{list.set } \alpha s \subseteq \text{ON} \rangle \langle \text{list.set } ms \subseteq \{0 < ..\} \rangle$ **by** *auto*
have $\mu: \mu = \omega \uparrow \alpha * \text{ord-of-nat } m$
using αm **by** (*simp add:* μ *)*
moreover have $m = 1$
proof (*rule ccontr*)
assume $m \neq 1$
then have $2: m \geq 2$
using $\langle m \neq 0 \rangle$ **by** *linarith*
then have $m = \text{Suc } 0 + \text{Suc } 0 + (m-2)$
by *simp*
then have $\text{ord-of-nat } m = 1 + 1 + \text{ord-of-nat } (m-2)$
by (*metis add.left-neutral mult.left-neutral mult-succ ord-of-nat.simps*
ord-of-nat-add)
then have $\mu \text{eq}: \mu = \omega \uparrow \alpha + \omega \uparrow \alpha + \omega \uparrow \alpha * \text{ord-of-nat } (m-2)$
using μ **by** (*simp add: add-mult-distrib*)
moreover have *less:* $\omega \uparrow \alpha < \mu$
by (*metis Ord- ω OrdmemD $\mu \text{eq} \langle \text{Ord } \alpha \rangle$ add-le-cancel-left0 add-less-cancel-left0*
le-less-trans less-V-def oexp-gt-0-iff zero-in-omega)
moreover have $\omega \uparrow \alpha + \omega \uparrow \alpha * \text{ord-of-nat } (m-2) < \mu$
using $2 \ \mu \langle \text{Ord } \alpha \rangle$ *assms less indecomposableD less-V-def* **by** *auto*
ultimately show *False*
using *indecomposableD [OF inc, of $\omega \uparrow \alpha \ \omega \uparrow \alpha + \omega \uparrow \alpha * \text{ord-of-nat } (m-2)$]*
by (*simp add:* $\langle \text{Ord } (\omega \uparrow \alpha) \rangle$ *add.assoc*)
qed
moreover have $\text{Ord } \alpha$
using $\langle \text{List.set } \alpha s \subseteq \text{ON} \rangle$ **by** (*simp add:* $\langle \alpha s = [\alpha] \rangle$ *)*
ultimately show *?thesis*
by (*metis One-nat-def mult.right-neutral ord-of-nat.simps one-V-def that(2)*)
next
case 2
then obtain $\alpha 1 \ \alpha 2 \ \alpha s' \ m 1 \ m 2 \ ms'$ **where** $\alpha m: \alpha s = \alpha 1 \# \alpha 2 \# \alpha s' \ ms =$
 $m 1 \# m 2 \# ms'$
by (*metis Cantor(3) One-nat-def Suc-1 impossible-Cons length-Cons list.size(3)*
not-numeral-le-zero remdups-adj.cases)
then obtain $\text{Ord } \alpha 1 \ \text{Ord } \alpha 2 \ m 1 \neq 0 \ m 2 \neq 0 \ \text{Ord } (\omega \uparrow \alpha 1) \ \text{Ord } (\omega \uparrow \alpha 2)$
 $\text{list.set } \alpha s' \subseteq \text{ON} \ \text{list.set } ms' \subseteq \{0 < ..\}$
using $\langle \text{list.set } \alpha s \subseteq \text{ON} \rangle \langle \text{list.set } ms \subseteq \{0 < ..\} \rangle$ **by** *auto*
have $oCs: \text{Ord } (\text{Cantor-sum } \alpha s' \ ms')$
by (*simp add: Ord-Cantor-sum* $\langle \text{list.set } \alpha s' \subseteq \text{ON} \rangle$)
have $\alpha 2 1: \alpha 2 \in \text{elts } \alpha 1$
using *Cantor-dec-Cons-iff* $\alpha m(1) \langle \text{Cantor-dec } \alpha s \rangle$
by (*simp add: Ord-mem-iff-lt* $\langle \text{Ord } \alpha 1 \rangle \langle \text{Ord } \alpha 2 \rangle$)
have $\omega \uparrow \alpha 2 \neq 0$
by (*simp add:* $\langle \text{Ord } \alpha 2 \rangle$ *)*
then have $*$: $(\omega \uparrow \alpha 2 * \text{ord-of-nat } m 2 + \text{Cantor-sum } \alpha s' \ ms') > 0$
by (*simp add: OrdmemD* $\langle \text{Ord } (\omega \uparrow \alpha 2) \rangle \langle m 2 \neq 0 \rangle \text{mem-0-Ord } oCs$)
have $\mu: \mu = \omega \uparrow \alpha 1 * \text{ord-of-nat } m 1 + (\omega \uparrow \alpha 2 * \text{ord-of-nat } m 2 + \text{Cantor-sum}$

```

 $\alpha s' ms'$ )
  (is  $\mu = ?\alpha + ?\beta$ )
  using  $\alpha m$  by (simp add:  $\mu$ )
  moreover
  have  $\omega^{\uparrow\alpha 2} * \text{ord-of-nat } m2 + \text{Cantor-sum } \alpha s' ms' < \omega^{\uparrow\alpha 1} * \text{ord-of-nat } m1$ 
+ ( $\omega^{\uparrow\alpha 2} * \text{ord-of-nat } m2 + \text{Cantor-sum } \alpha s' ms'$ )
  if  $\alpha 2 \in \text{elts } \alpha 1$ 
  proof (rule less- $\omega$ -power)
  show  $\text{Cantor-sum } \alpha s' ms' < \omega^{\uparrow\alpha 2}$ 
  using  $\alpha m$  Cantor cnf-2 by auto
  qed (use oCs  $\langle \text{Ord } \alpha 1 \rangle \langle m1 \neq 0 \rangle \langle m2 \neq 0 \rangle$  that in auto)
  then have  $?\beta < \mu$ 
  using  $\alpha 2 1$  by (simp add:  $\mu \alpha m$ )
  moreover have less:  $?\alpha < \mu$ 
  using oCs by (metis  $\mu * \text{add-less-cancel-left add.right-neutral}$ )
  ultimately have False
  using indecomposableD [OF inc, of  $?\alpha ?\beta$ ]
  by (simp add:  $\langle \text{Ord } (\omega^{\uparrow\alpha 1}) \rangle \langle \text{Ord } (\omega^{\uparrow\alpha 2}) \rangle$  oCs)
  then show ?thesis ..
  qed
qed

```

corollary *indecomposable-iff- ω -power*:

indecomposable $\mu \longleftrightarrow \mu = 0 \vee (\exists \delta. \mu = \omega^{\uparrow\delta} \wedge \text{Ord } \delta)$

by (*meson indecomposable-0 indecomposable- ω -power indecomposable-is- ω -power*)

theorem *indecomposable-imp-type*:

fixes $X :: \text{bool} \Rightarrow V \text{ set}$

assumes γ : *indecomposable* γ

and $\bigwedge b. \text{ordertype } (X b) \text{ VWF} \leq \gamma \wedge b. \text{small } (X b) \wedge b. X b \subseteq \text{ON}$

and $\text{elts } \gamma \subseteq (\text{UN } b. X b)$

shows $\exists b. \text{ordertype } (X b) \text{ VWF} = \gamma$

using γ [THEN *indecomposable-imp-Ord*] *assms*

proof (*induction arbitrary: X*)

case (*succ* β)

show ?*case*

proof (*cases* $\beta = 0$)

case *True*

then have $\exists b. 0 \in X b$

using *succ.prem*s(5) by *blast*

then have $\exists b. \text{ordertype } (X b) \text{ VWF} \neq 0$

using *succ.prem*s(3) by *auto*

then have $\exists b. \text{ordertype } (X b) \text{ VWF} \geq \text{succ } 0$

by (*meson Ord-0 Ord-linear2 Ord-ordertype less-eq-V-0-iff succ-le-iff*)

then show ?*thesis*

using *True succ.prem*s(2) by *blast*

next

case *False*

then show ?*thesis*

```

    using succ.premis by auto
  qed
next
case (Limit  $\gamma$ )
then obtain  $\delta$  where  $\delta: \gamma = \omega \uparrow \delta$  and  $\delta \neq 0$  Ord  $\delta$ 
  by (metis Limit-eq-Sup-self image-ident indecomposable-is- $\omega$ -power not-succ-Limit
  oexp-0-right one-V-def zero-not-Limit)
  show ?case
  proof (cases Limit  $\delta$ )
    case True
    have ot:  $\exists b. \text{ordertype } (X b \cap \text{elts } (\omega \uparrow \alpha)) \text{ VWF} = \omega \uparrow \alpha$ 
      if  $\alpha \in \text{elts } \delta$  for  $\alpha$ 
    proof (rule Limit.IH)
      have Ord  $\alpha$ 
        using Ord-in-Ord  $\langle \text{Ord } \delta \rangle$  that by blast
      then show  $\omega \uparrow \alpha \in \text{elts } \gamma$ 
        by (simp add: Ord-mem-iff-lt  $\delta$   $\omega$ -gt1  $\langle \text{Ord } \delta \rangle$  oexp-less that)
      show indecomposable  $(\omega \uparrow \alpha)$ 
        using  $\langle \text{Ord } \alpha \rangle$  indecomposable-1 indecomposable- $\omega$ -power by fastforce
      show small  $(X b \cap \text{elts } (\omega \uparrow \alpha))$  for  $b$ 
        by (meson down inf-le2)
      show ordertype  $(X b \cap \text{elts } (\omega \uparrow \alpha)) \text{ VWF} \leq \omega \uparrow \alpha$  for  $b$ 
        by (simp add:  $\langle \text{Ord } \alpha \rangle$  ordertype-le-Ord)
      show  $X b \cap \text{elts } (\omega \uparrow \alpha) \subseteq \text{ON}$  for  $b$ 
        by (simp add: Limit.premis inf.coboundedI1)
      show  $\text{elts } (\omega \uparrow \alpha) \subseteq (\bigcup b. X b \cap \text{elts } (\omega \uparrow \alpha))$ 
        using Limit.premis Limit.hyps  $\langle \omega \uparrow \alpha \in \text{elts } \gamma \rangle$ 
        by clarsimp (metis Ord-trans UN-E indecomposable-imp-Ord subset-eq)
    qed
  define A where  $A \equiv \lambda b. \{ \alpha \in \text{elts } \delta. \text{ordertype } (X b \cap \text{elts } (\omega \uparrow \alpha)) \text{ VWF} \geq \omega \uparrow \alpha \}$ 
  have A small: small  $(A b)$  for  $b$ 
    by (simp add: A-def)
  have A ON:  $A b \subseteq \text{ON}$  for  $b$ 
    using A-def  $\langle \text{Ord } \delta \rangle$  elts-subset-ON by blast
  have eq:  $\text{elts } \delta = (\bigcup b. A b)$ 
    by (auto simp: A-def) (metis ot eq-refl)
  then obtain  $b$  where  $b: \text{Sup } (A b) = \delta$ 
    using  $\langle \text{Limit } \delta \rangle$ 
    apply (auto simp: UN-bool-eq)
    by (metis A ON ON-imp-Ord Ord-Sup Ord-linear-le Limit-eq-Sup-self Sup-Un-distrib
  A small sup.absorb2 sup.orderE)
  have  $\omega \uparrow \alpha \leq \text{ordertype } (X b) \text{ VWF}$  if  $\alpha \in A b$  for  $\alpha$ 
  proof -
    have  $(\omega \uparrow \alpha) = \text{ordertype } ((X b) \cap \text{elts } (\omega \uparrow \alpha)) \text{ VWF}$ 
      using  $\langle \text{Ord } \delta \rangle$  that by (simp add: A-def Ord-in-Ord dual-order.antisym
  ordertype-le-Ord)
    also have  $\dots \leq \text{ordertype } (X b) \text{ VWF}$ 
      by (simp add: Limit.premis ordertype-VWF-mono)
  
```



```

    finally show ?thesis .
  qed
  then have ordertype (X b) VWF  $\geq$  Sup (( $\lambda\alpha. \omega\uparrow\alpha$ ) ' A b)
    by blast
  moreover have Sup (( $\lambda\alpha. \omega\uparrow\alpha$ ) ' A b) =  $\omega \uparrow$  Sup (A b)
    by (metis b Ord- $\omega$  ZFC-in-HOL.Sup-empty AON  $\langle \delta \neq 0 \rangle$  Asmall oexp-Sup
    omega-nonzero)
  ultimately show ?thesis
    using Limit.hyps Limit.premis  $\delta$  b by auto
  next
  case False
  then obtain  $\beta$  where  $\beta: \delta = \text{succ } \beta$  Ord  $\beta$ 
    using Ord-cases  $\langle \delta \neq 0 \rangle$   $\langle$  Ord  $\delta \rangle$  by auto
  then have Ord $\omega\beta$ : Ord ( $\omega\uparrow\beta$ )
    using Ord-oexp by blast
  have subX12: elts ( $\omega\uparrow\beta * \omega$ )  $\subseteq$  ( $\bigcup$  b. X b)
    using Limit  $\beta$   $\delta$  by auto
  define E where  $E \equiv \lambda n. \{\omega\uparrow\beta * \text{ord-of-nat } n .. < \omega\uparrow\beta * \text{ord-of-nat } (\text{Suc } n)\}$ 
 $\cap$  ON
  have EON:  $E n \subseteq ON$  for  $n$ 
    using E-def by blast
  have E-imp-less:  $x < y$  if  $i < j$   $x \in E i$   $y \in E j$  for  $x y i j$ 
  proof -
    have succ ( $i$ )  $\leq$  ord-of-nat  $j$ 
      using that(1) by force
    then have  $\neg y \leq x$ 
      using that
    apply (auto simp: E-def)
    by (metis Ord $\omega\beta$  Ord-ord-of-nat leD mult-cancel-le-iff ord-of-nat.simps(2)
    order-trans)
    with that show ?thesis
      by (meson EON ON-imp-Ord Ord-linear2)
  qed
  then have djE: disjnt (E  $i$ ) (E  $j$ ) if  $i \neq j$  for  $i j$ 
    using that nat-neq-iff unfolding disjnt-def by auto
  have less-imp-E:  $i \leq j$  if  $x < y$   $x \in E i$   $y \in E j$  for  $x y i j$ 
    using that E-imp-less [OF -  $\langle y \in E j \rangle \langle x \in E i \rangle$ ] leI less-asm by blast
  have inc: indecomposable ( $\omega\uparrow\beta$ )
    using  $\beta$  indecomposable-1 indecomposable- $\omega$ -power by fastforce
  have in-En:  $\omega\uparrow\beta * \text{ord-of-nat } n + x \in E n$  if  $x \in \text{elts } (\omega\uparrow\beta)$  for  $x n$ 
    using that Ord $\omega\beta$  Ord-in-Ord [OF Ord $\omega\beta$ ] by (auto simp: E-def Ord $\omega\beta$ 
    OrdmemD mult-succ)
  have *: elts  $\gamma = \bigcup$  (range E)
  proof
    have  $\exists m. \omega\uparrow\beta * m \leq x \wedge x < \omega\uparrow\beta * \text{succ } (\text{ord-of-nat } m)$ 
      if  $x \in \text{elts } (\omega\uparrow\beta * \text{ord-of-nat } n)$  for  $x n$ 
      using that
      apply (clarsimp simp add: mult [of - ord-of-nat  $n$ ] lift-def)
      by (metis add-less-cancel-left OrdmemD inc indecomposable-imp-Ord mult-succ

```

plus sup-ge1)
moreover have $\text{Ord } x$ **if** $x \in \text{elts } (\omega \uparrow \beta * \text{ord-of-nat } n)$ **for** $x \ n$
by (*meson Ord $\omega\beta$ Ord-in-Ord Ord-mult Ord-ord-of-nat that*)
ultimately show $\text{elts } \gamma \subseteq \bigcup (\text{range } E)$
by (*auto simp: $\delta \ \beta$ E-def mult-Limit elts- ω*)
have $x \in \text{elts } (\omega \uparrow \beta * \text{succ}(\text{ord-of-nat } n))$
if $\text{Ord } x \ x < \omega \uparrow \beta * \text{succ } (n)$ **for** $x \ n$
by (*metis that Ord-mem-iff-lt Ord-mult Ord-ord-of-nat inc indecomposable-imp-Ord ord-of-nat.simps(2)*)
then show $\bigcup (\text{range } E) \subseteq \text{elts } \gamma$
by (*force simp: $\delta \ \beta$ E-def Limit.premis mult-Limit*)
qed
have *smE*: $\text{small } (E \ n)$ **for** n
by (*metis * complete-lattice-class.Sup-upper down rangeI*)
have *otE*: $\text{ordertype } (E \ n) \ \text{VWF} = \omega \uparrow \beta$ **for** n
by (*simp add: E-def inc indecomposable-imp-Ord mult-succ ordertype-interval-eq*)

define *cut* **where** $\text{cut} \equiv \lambda n \ x. \ \text{odiff } x \ (\omega \uparrow \beta * \text{ord-of-nat } n)$
have *cutON*: $\text{cut } n \ ' \ X \subseteq \text{ON}$ **if** $X \subseteq \text{ON}$ **for** $n \ X$
using *that* **by** (*simp add: image-subset-iff cut-def ON-imp-Ord Ord $\omega\beta$ Ord-odiff*)
have *cut* [*simp*]: $\text{cut } n \ (\omega \uparrow \beta * \text{ord-of-nat } n + x) = x$ **for** $x \ n$
by (*auto simp: cut-def*)
have *cuteq*: $x \in \text{cut } n \ ' \ (X \cap E \ n) \longleftrightarrow \omega \uparrow \beta * \text{ord-of-nat } n + x \in X$
if $x: x \in \text{elts } (\omega \uparrow \beta)$ **for** $x \ X \ n$
proof
show $\omega \uparrow \beta * \text{ord-of-nat } n + x \in X$ **if** $x \in \text{cut } n \ ' \ (X \cap E \ n)$
using *E-def Ord $\omega\beta$ Ord-odiff-eq image-iff local.cut-def* **that** **by** *auto*
show $x \in \text{cut } n \ ' \ (X \cap E \ n)$
if $\omega \uparrow \beta * \text{ord-of-nat } n + x \in X$
by (*metis (full-types) IntI cut image-iff in-En that x*)
qed
have *ot-cuteq*: $\text{ordertype } (\text{cut } n \ ' \ (X \cap E \ n)) \ \text{VWF} = \text{ordertype } (X \cap E \ n)$
VWF **for** $n \ X$
proof (*rule ordertype-VWF-inc-eq*)
show $X \cap E \ n \subseteq \text{ON}$
using *E-def* **by** *blast*
then show $\text{cut } n \ ' \ (X \cap E \ n) \subseteq \text{ON}$
by (*simp add: cutON*)
show $\text{small } (X \cap E \ n)$
by (*meson Int-lower2 smE smaller-than-small*)
show $\text{cut } n \ x < \text{cut } n \ y$
if $x \in X \cap E \ n \ y \in X \cap E \ n \ x < y$ **for** $x \ y$
using *that* $\langle X \cap E \ n \subseteq \text{ON} \rangle$ **by** (*simp add: E-def Ord $\omega\beta$ Ord-odiff-less-odiff local.cut-def*)
qed

define *N* **where** $N \equiv \lambda b. \ \{n. \ \text{ordertype } (X \ b \cap E \ n) \ \text{VWF} = \omega \uparrow \beta\}$
have $\exists b. \ \text{infinite } (N \ b)$
proof (*rule ccontr*)

```

assume  $\nexists b. \text{infinite } (N b)$ 
then obtain  $n$  where  $\bigwedge b. n \notin N b$ 
  apply (simp add: ex-bool-eq)
  by (metis (full-types) finite-nat-set-iff-bounded not-less-iff-gr-or-eq)
moreover
have  $\exists b. \text{ordertype } (\text{cut } n \text{ ' } (X b \cap E n)) \text{ VWF} = \omega \uparrow \beta$ 
proof (rule Limit.IH)
  show  $\omega \uparrow \beta \in \text{elts } \gamma$ 
    by (metis Limit.hyps Limit-def Limit-omega Ord-mem-iff-less-TC  $\beta$   $\delta$ 
mult-le2 not-succ-Limit oexp-succ omega-nonzero one-V-def)
  show indecomposable  $(\omega \uparrow \beta)$ 
    by (simp add: inc)
  show ordertype  $(\text{cut } n \text{ ' } (X b \cap E n)) \text{ VWF} \leq \omega \uparrow \beta$  for  $b$ 
    by (metis otE inf-le2 ordertype-VWF-mono ot-cuteq smE)
  show small  $(\text{cut } n \text{ ' } (X b \cap E n))$  for  $b$ 
    using smE subset-iff-less-eq-V
    by (meson inf-le2 replacement)
  show  $\text{cut } n \text{ ' } (X b \cap E n) \subseteq \text{ON}$  for  $b$ 
    using E-def cutON by auto
  have  $\text{elts } (\omega \uparrow \beta * \text{succ } n) \subseteq \bigcup (\text{range } X)$ 
    by (metis Ord $\omega$  $\beta$  Ord- $\omega$  Ord-ord-of-nat less-eq-V-def mult-cancel-le-iff
ord-of-nat.simps(2) ord-of-nat-le-omega order-trans subX12)
  then show  $\text{elts } (\omega \uparrow \beta) \subseteq \bigcup b. \text{cut } n \text{ ' } (X b \cap E n)$ 
    by (auto simp: mult-succ mult-Limit UN-subset-iff cuteq UN-bool-eq)
qed
then have  $\exists b. \text{ordertype } (X b \cap E n) \text{ VWF} = \omega \uparrow \beta$ 
  by (simp add: ot-cuteq)
ultimately show False
  by (simp add: N-def)
qed
then obtain  $b$  where  $b: \text{infinite } (N b)$ 
  by blast
  then obtain  $\varphi :: \text{nat} \Rightarrow \text{nat}$  where  $\varphi: \text{bij-betw } \varphi \text{ UNIV } (N b)$  and mono:
strict-mono  $\varphi$ 
    by (meson bij-enumerate enumerate-mono strict-mono-def)
  then have ordertype  $(X b \cap E (\varphi n)) \text{ VWF} = \omega \uparrow \beta$  for  $n$ 
    using N-def bij-betw-imp-surj-on by blast
  moreover have small  $(X b \cap E (\varphi n))$  for  $n$ 
    by (meson inf-le2 smE subset-iff-less-eq-V)
  ultimately have  $\exists f. \text{bij-betw } f (X b \cap E (\varphi n)) (\text{elts } (\omega \uparrow \beta)) \wedge (\forall x \in X b \cap E (\varphi n). \forall y \in X b \cap E (\varphi n). f x < f y \iff (x,y) \in \text{VWF})$ 
    for  $n$  by (metis Ord-ordertype ordertype-VWF-eq-iff)
  then obtain  $F$  where bijF:  $\bigwedge n. \text{bij-betw } (F n) (X b \cap E (\varphi n)) (\text{elts } (\omega \uparrow \beta))$ 
    and  $F: \bigwedge n. \forall x \in X b \cap E (\varphi n). \forall y \in X b \cap E (\varphi n). F n x < F n y \iff (x,y) \in \text{VWF}$ 
    by metis
  then have F-bound:  $\bigwedge n. \forall x \in X b \cap E (\varphi n). F n x < \omega \uparrow \beta$ 
    by (metis Ord- $\omega$  Ord-oexp OrdmemD  $\beta$ (2) bij-betw-imp-surj-on image-eqI)
  have F-Ord:  $\bigwedge n. \forall x \in X b \cap E (\varphi n). \text{Ord } (F n x)$ 

```

by (metis otE ON-imp-Ord Ord-ordertype bijF bij-betw-def elts-subset-ON imageI)

have inc: $\varphi n \geq n$ for n
by (simp add: mono strict-mono-imp-increasing)

have dj φ : disjnt (E (φi)) (E (φj)) if $i \neq j$ for $i j$
by (rule djE) (use φ that in \langle auto simp: bij-betw-def inj-def \rangle)

define Y where $Y \equiv (\bigcup n. E (\varphi n))$

have $\exists n. y \in E (\varphi n)$ if $y \in Y$ for y
using Y-def that by blast

then obtain ι where $\iota: \bigwedge y. y \in Y \implies y \in E (\iota y)$
by metis

have $Y \subseteq ON$
by (auto simp: Y-def E-def)

have ι le: $\iota x \leq \iota y$ if $x < y$ $x \in Y$ $y \in Y$ for $x y$
using less-imp-E strict-mono-less-eq that ι [OF $\langle x \in Y \rangle$] ι [OF $\langle y \in Y \rangle$]

mono

unfolding Y-def by blast

have equ: $x \in E (\varphi k) \implies \iota x = k$ for $x k$
using ι unfolding Y-def

by (meson UN-I disjnt-iff dj φ iso-tuple-UNIV-I)

have upper: $\omega \uparrow \beta * \text{ord-of-nat } (\iota x) \leq x$ if $x \in Y$ for x
using that

proof (clarsimp simp add: Y-def equ)

fix $u v$

assume $u: u \in \text{elts } (\omega \uparrow \beta * \text{ord-of-nat } v)$ and $v: x \in E (\varphi v)$

then have $u < \omega \uparrow \beta * \text{ord-of-nat } v$

by (simp add: OrdmemD $\beta(2)$)

also have $\dots \leq \omega \uparrow \beta * \text{ord-of-nat } (\varphi v)$

by (simp add: $\beta(2)$ inc)

also have $\dots \leq x$

using v by (simp add: E-def)

finally show $u \in \text{elts } x$

using $\langle Y \subseteq ON \rangle$

by (meson ON-imp-Ord Ord- ω Ord-in-Ord Ord-mem-iff-lt Ord-mult Ord-oe xp Ord-ord-of-nat $\beta(2)$ that u)

qed

define G where $G \equiv \lambda x. \omega \uparrow \beta * \text{ord-of-nat } (\iota x) + F (\iota x) x$

have G-strict-mono: $G x < G y$ if $x < y$ $x \in X$ $b \cap Y$ $y \in X$ $b \cap Y$ for $x y$

proof (cases $\iota x = \iota y$)

case True

then show ?thesis

using that unfolding G-def

by (metis F Int-iff add-less-cancel-left Limit.prem $s(4)$ ON-imp-Ord

VWF-iff-Ord-less ι)

next

```

case False
then have  $\iota x < \iota y$ 
  by (meson IntE ile le-less that)
then show ?thesis
using that by (simp add: G-def F-Ord F-bound Ord $\omega\beta$   $\iota$  mult-nat-less-add-less)
qed

have ordertype  $(X b \cap Y) VWF = (\omega \uparrow \beta) * \omega$ 
proof (rule ordertype-VWF-eq-iff [THEN iffD2])
  show Ord  $(\omega \uparrow \beta * \omega)$ 
    by (simp add:  $\beta$ )
  show small  $(X b \cap Y)$ 
    by (meson Limit.premis(3) inf-le1 subset-iff-less-eq-V)
  have bij-betw  $G (X b \cap Y) (elts (\omega \uparrow \beta * \omega))$ 
proof (rule bij-betw-imageI)
  show inj-on  $G (X b \cap Y)$ 
proof (rule linorder-inj-onI)
  fix  $x y$ 
  assume  $xy: x < y \ x \in (X b \cap Y) \ y \in (X b \cap Y)$ 
  show  $G x \neq G y$ 
    using G-strict-mono xy by force
next
  show  $x \leq y \vee y \leq x$ 
    if  $x \in (X b \cap Y) \ y \in (X b \cap Y)$  for  $x y$ 
      using that  $\langle X b \subseteq ON \rangle$  by (clarsimp simp: Y-def) (metis ON-imp-Ord
Ord-linear Ord-trans)
qed
show  $G \text{ ` } (X b \cap Y) = elts (\omega \uparrow \beta * \omega)$ 
proof
  show  $G \text{ ` } (X b \cap Y) \subseteq elts (\omega \uparrow \beta * \omega)$ 
    using  $\langle X b \subseteq ON \rangle$ 
    apply (clarsimp simp: G-def mult-Limit Y-def eqI)
    by (metis IntI add-mem-right-cancel bijF bij-betw-imp-surj-on image-eqI
mult-succ ord-of-nat- $\omega$  succ-in-omega)
  show  $elts (\omega \uparrow \beta * \omega) \subseteq G \text{ ` } (X b \cap Y)$ 
proof
  fix  $x$ 
  assume  $x: x \in elts (\omega \uparrow \beta * \omega)$ 
  then obtain  $k$  where  $n: x \in elts (\omega \uparrow \beta * ord\text{-of-nat} (Suc\ k))$ 
    and minim:  $\bigwedge m. m < Suc\ k \implies x \notin elts (\omega \uparrow \beta * ord\text{-of-nat}$ 
m)
    using elts-mult- $\omega E$ 
    by (metis old.nat.exhaust)
  then obtain  $y$  where  $y: y \in elts (\omega \uparrow \beta)$  and xeq:  $x = \omega \uparrow \beta * ord\text{-of-nat}$ 
k + y
    using  $x$  by (auto simp: mult-succ elim: mem-plus-V-E)
  then have  $1: inv\text{-into} (X b \cap E (\varphi\ k)) (F\ k) \ y \in (X b \cap E (\varphi\ k))$ 
    by (metis bijF bij-betw-def inv-into-into)
  then have  $(inv\text{-into} (X b \cap E (\varphi\ k)) (F\ k) \ y) \in X b \cap Y$ 

```

by (force simp: Y-def)
 moreover have $G (\text{inv-into } (X \text{ b } \cap E (\varphi \text{ k})) (F \text{ k}) \text{ y}) = x$
 by (metis 1 G-def Int-iff bijF bij-betw-inv-into-right eqt xeq y)
 ultimately show $x \in G '(X \text{ b } \cap Y)$
 by blast
 qed
 qed
 qed
 moreover have $(x, y) \in VWF$
 if $x \in X \text{ b } x \in Y \text{ y} \in X \text{ b } y \in Y \text{ G } x < \text{ G } y$ for $x \text{ y}$
 proof –
 have $x < y$
 using that by (metis G-strict-mono Int-iff Limit.prem(4) ON-imp-Ord
 Ord-linear-lt less-asm)
 then show ?thesis
 using ON-imp-Ord $\langle Y \subseteq ON \rangle$ that by auto
 qed
 moreover have $G x < G y$
 if $x \in X \text{ b } x \in Y \text{ y} \in X \text{ b } y \in Y (x, y) \in VWF$ for $x \text{ y}$
 proof –
 have $x < y$
 using that ON-imp-Ord $\langle Y \subseteq ON \rangle$ by auto
 then show ?thesis
 by (simp add: G-strict-mono that)
 qed
 ultimately show $\exists f. \text{bij-betw } f (X \text{ b } \cap Y) (\text{elts } (\omega \uparrow \beta * \omega)) \wedge (\forall x \in (X \text{ b } \cap Y). \forall y \in (X \text{ b } \cap Y). f x < f y \longleftrightarrow ((x, y) \in VWF))$
 by blast
 qed
 moreover have $\text{ordertype } (\bigcup n. X \text{ b } \cap E (\varphi \text{ n})) VWF \leq \text{ordertype } (X \text{ b}) VWF$
 using Limit.prem(3) ordertype-VWF-mono by auto
 ultimately have $\text{ordertype } (X \text{ b}) VWF = (\omega \uparrow \beta) * \omega$
 using Limit.hyps Limit.prem(2) $\beta \delta$
 using Y-def by auto
 then show ?thesis
 using Limit.hyps $\beta \delta$ by auto
 qed
 qed auto

corollary *indecomposable-imp-type2:*

assumes $\alpha: \text{indecomposable } \gamma \text{ X } \subseteq \text{elts } \gamma$
 shows $\text{ordertype } X VWF = \gamma \vee \text{ordertype } (\text{elts } \gamma - X) VWF = \gamma$
 proof –
 have $\text{Ord } \gamma$
 using assms indecomposable-imp-Ord by blast
 have $\exists b. \text{ordertype } (\text{if } b \text{ then } X \text{ else } \text{elts } \gamma - X) VWF = \gamma$
 proof (rule indecomposable-imp-type)
 show $\text{ordertype } (\text{if } b \text{ then } X \text{ else } \text{elts } \gamma - X) VWF \leq \gamma$ for b

by (simp add: ⟨Ord γ⟩ assms ordertype-le-Ord)
 show (if b then X else elts γ - X) ⊆ ON for b
 using ⟨Ord γ⟩ assms elts-subset-ON by auto
 qed (use assms down in auto)
 then show ?thesis
 by (metis (full-types))
 qed

5.4 From ordinals to order types

lemma *indecomposable-ordertype-eq*:

assumes *indec*: indecomposable α and α: ordertype A VWF = α and A: B ⊆ A small A
 shows ordertype B VWF = α ∨ ordertype (A-B) VWF = α
proof (rule ccontr)
 assume ¬ (ordertype B VWF = α ∨ ordertype (A - B) VWF = α)
 moreover have ordertype (ordermap A VWF ‘ B) VWF = ordertype B VWF
 using ⟨B ⊆ A⟩ by (auto intro: ordertype-image-ordermap [OF ⟨small A⟩])
 moreover have ordertype (elts α - ordermap A VWF ‘ B) VWF = ordertype (A - B) VWF
 by (metis ordertype-map-image α A elts-of-set ordertype-def replacement)
 moreover have ordermap A VWF ‘ B ⊆ elts α
 using α A by blast
 ultimately show False
 using *indecomposable-imp-type2* [OF ⟨indecomposable α⟩] ⟨small A⟩ by metis
 qed

lemma *indecomposable-ordertype-ge*:

assumes *indec*: indecomposable α and α: ordertype A VWF ≥ α and small: small A small B
 shows ordertype B VWF ≥ α ∨ ordertype (A-B) VWF ≥ α
proof –
 obtain A' where A' ⊆ A ordertype A' VWF = α
 by (meson α ⟨small A⟩ *indec indecomposable-def le-ordertype-obtains-subset*)
 then have ordertype (B ∩ A') VWF = α ∨ ordertype (A'-B) VWF = α
 by (metis Diff-Diff-Int Diff-subset Int-commute ⟨small A⟩ *indecomposable-ordertype-eq indec smaller-than-small*)
 moreover have ordertype (B ∩ A') VWF ≤ ordertype B VWF
 by (meson Int-lower1 small ordertype-VWF-mono smaller-than-small)
 moreover have ordertype (A'-B) VWF ≤ ordertype (A-B) VWF
 by (meson Diff-mono Diff-subset ⟨A' ⊆ A⟩ ⟨small A⟩ order-refl ordertype-VWF-mono smaller-than-small)
 ultimately show ?thesis
 by blast
 qed

now for finite partitions

lemma *indecomposable-ordertype-finite-eq*:

assumes *indecomposable* α

```

    and  $\mathcal{A}$ : finite  $\mathcal{A}$  pairwise disjoint  $\mathcal{A} \cup \mathcal{A} = A$   $\mathcal{A} \neq \{\}$  ordertype  $A$   $VWF = \alpha$ 
small  $A$ 
  shows  $\exists X \in \mathcal{A}$ . ordertype  $X$   $VWF = \alpha$ 
  using  $\mathcal{A}$ 
proof (induction arbitrary:  $A$ )
  case (insert  $X$   $\mathcal{A}$ )
  show ?case
  proof (cases  $\mathcal{A} = \{\}$ )
    case True
    then show ?thesis
      using insert.prem by blast
  next
  case False
  have  $smA$ : small  $(\bigcup \mathcal{A})$ 
    using insert.prem by auto
  show ?thesis
  proof (cases  $\exists X \in \mathcal{A}$ . ordertype  $X$   $VWF = \alpha$ )
    case True
    then show ?thesis
      using insert.prem by blast
  next
  case False
  have  $X = A - \bigcup \mathcal{A}$ 
    using insert.hyps insert.prem by (auto simp: pairwise-insert disjoint-iff)
  then have ordertype  $X$   $VWF = \alpha$ 
    using indecomposable-ordertype-eq assms insert False
    by (metis Union-mono cSup-singleton pairwise-insert  $smA$  subset-insertI)
  then show ?thesis
    using insert.prem by blast
  qed
qed
qed auto

```

```

lemma indecomposable-ordertype-finite-ge:
  assumes indec: indecomposable  $\alpha$ 
    and  $\mathcal{A}$ : finite  $\mathcal{A}$   $A \subseteq \bigcup \mathcal{A}$   $\mathcal{A} \neq \{\}$  ordertype  $A$   $VWF \geq \alpha$  small  $(\bigcup \mathcal{A})$ 
  shows  $\exists X \in \mathcal{A}$ . ordertype  $X$   $VWF \geq \alpha$ 
  using  $\mathcal{A}$ 
proof (induction arbitrary:  $A$ )
  case (insert  $X$   $\mathcal{A}$ )
  show ?case
  proof (cases  $\mathcal{A} = \{\}$ )
    case True
    then have  $\alpha \leq$  ordertype  $X$   $VWF$ 
      using insert.prem
      by (simp add: order.trans ordertype-VWF-mono)
    then show ?thesis
      using insert.prem by blast
  next

```



```

case False
show ?thesis
proof (cases  $\exists X \in \mathcal{A}. \text{ordertype } X \text{ VWF} \geq \alpha$ )
  case True
  then show ?thesis
    using insert.prems by blast
  next
  case False
  moreover have small ( $X \cup \bigcup \mathcal{A}$ )
    using insert.prems by auto
  moreover have ordertype ( $\bigcup (\text{insert } X \ \mathcal{A})$ ) VWF  $\geq \alpha$ 
    using insert.prems ordertype-VWF-mono by blast
  ultimately have ordertype  $X$  VWF  $\geq \alpha$ 
    using indecomposable-ordertype-ge [OF indec]
    by (metis Diff-subset-conv Sup-insert cSup-singleton insert.IH small-sup-iff
subset-refl)
  then show ?thesis
    using insert.prems by blast
  qed
qed
qed auto

end

```

6 Type Classes for ZFC

```

theory ZFC-Typeclasses
  imports ZFC-Cardinals Complex-Main

```

```

begin

```

6.1 The class of embeddable types

```

class embeddable =
  assumes ex-inj:  $\exists V\text{-of} :: 'a \Rightarrow V. \text{inj } V\text{-of}$ 

context countable
begin

  subclass embeddable
  proof –
    have inj (ord-of-nat  $\circ$  to-nat) if inj to-nat
      for to-nat  $:: 'a \Rightarrow \text{nat}$ 
      using that by (simp add: inj-compose inj-ord-of-nat)
    then show class.embeddable TYPE('a)
      by intro-classes (meson local.ex-inj)
  qed

end

```

```

instance unit :: embeddable ..
instance bool :: embeddable ..
instance nat :: embeddable ..
instance int :: embeddable ..
instance rat :: embeddable ..
instance char :: embeddable ..
instance String.literal :: embeddable ..
instance typerep :: embeddable ..

```

```

lemma embeddable-classI:
  fixes f :: 'a ⇒ V
  assumes  $\bigwedge x y. f x = f y \implies x = y$ 
  shows OFCLASS('a, embeddable-class)
proof (intro-classes, rule exI)
  show inj f
  by (rule injI [OF assms]) assumption
qed

```

```

instance V :: embeddable
  by (intro-classes) (meson inj-on-id)

```

```

instance prod :: (embeddable,embeddable) embeddable
proof –
  have inj ( $\lambda(x,y). \langle V\text{-of1 } x, V\text{-of2 } y \rangle$ ) if inj V-of1 inj V-of2
  for V-of1 :: 'a ⇒ V and V-of2 :: 'b ⇒ V
  using that by (auto simp: inj-on-def)
  then show OFCLASS('a × 'b, embeddable-class)
  by intro-classes (meson embeddable-class.ex-inj)
qed

```

```

instance sum :: (embeddable,embeddable) embeddable
proof –
  have inj (case-sum (Inl ∘ V-of1) (Inr ∘ V-of2)) if inj V-of1 inj V-of2
  for V-of1 :: 'a ⇒ V and V-of2 :: 'b ⇒ V
  using that by (auto simp: inj-on-def split: sum.split-asm)
  then show OFCLASS('a + 'b, embeddable-class)
  by intro-classes (meson embeddable-class.ex-inj)
qed

```

```

instance option :: (embeddable) embeddable
proof –
  have inj (case-option 0 ( $\lambda x. \text{ZFC-in-HOL.set}\{V\text{-of } x\}$ )) if inj V-of
  for V-of :: 'a ⇒ V
  using that by (auto simp: inj-on-def split: option.split-asm)
  then show OFCLASS('a option, embeddable-class)
  by intro-classes (meson embeddable-class.ex-inj)
qed

```

```

primrec V-of-list where
  V-of-list V-of Nil = 0
| V-of-list V-of (x#xs) = ⟨V-of x, V-of-list V-of xs⟩

lemma inj-V-of-list:
  assumes inj V-of
  shows inj (V-of-list V-of)
proof –
  note inj-eq [OF assms, simp]
  have x = y if V-of-list V-of x = V-of-list V-of y for x y
    using that
  proof (induction x arbitrary: y)
    case Nil
    then show ?case
      by (cases y auto)
    next
    case (Cons a x)
    then show ?case
      by (cases y auto)
  qed
  then show ?thesis
    by (auto simp: inj-on-def)
qed

instance list :: (embeddable) embeddable
proof –
  have inj (rec-list 0 (λx xs r. ⟨V-of x, r⟩)) (is inj ?f)
    if V-of: inj V-of for V-of :: 'a ⇒ V
  proof –
  note inj-eq [OF V-of, simp]
  have x = y if ?f x = ?f y for x y
    using that
  proof (induction x arbitrary: y)
    case Nil
    then show ?case
      by (cases y auto)
    next
    case (Cons a x)
    then show ?case
      by (cases y auto)
  qed
  then show ?thesis
    by (auto simp: inj-on-def)
qed
  then show OFCLASS('a list, embeddable-class)
    by intro-classes (meson embeddable-class.ex-inj)
qed

```

6.2 The class of small types

```
class small =
  assumes small: small (UNIV::'a set)
begin

subclass embeddable
  by intro-classes (meson local.small small-def)

lemma TC-small [iff]:
  fixes A :: 'a set
  shows small A
  using small smaller-than-small by blast

end

context countable
begin

subclass small
proof -
  have *: inj (ord-of-nat o to-nat) if inj to-nat
    for to-nat :: 'a ⇒ nat
    using that by (simp add: inj-compose inj-ord-of-nat)
  then show class.small TYPE('a)
    by intro-classes (metis small-image-nat local.ex-inj the-inv-into-onto)
qed

end

lemma lepoll-UNIV-imp-small:  $X \lesssim (UNIV::'a::small\ set) \implies small\ X$ 
  by (meson lepoll-iff replacement small smaller-than-small)

lemma lepoll-imp-small:
  fixes A :: 'a::small set
  assumes  $X \lesssim A$ 
  shows small X
  by (metis lepoll-UNIV-imp-small UNIV-I assms lepoll-def subsetI)

instance unit :: small ..
instance bool :: small ..
instance nat :: small ..
instance int :: small ..
instance rat :: small ..
instance char :: small ..
instance String.literal :: small ..
instance typerep :: small ..

instance prod :: (small,small) small
proof -
```

```

have inj ( $\lambda(x,y). \langle V\text{-of1 } x, V\text{-of2 } y \rangle$ )
  range ( $\lambda(x,y). \langle V\text{-of1 } x, V\text{-of2 } y \rangle$ )  $\leq$  elts (VSigma A ( $\lambda x. B$ ))
if inj V-of1 inj V-of2 range V-of1  $\leq$  elts A range V-of2  $\leq$  elts B
for V-of1 :: 'a  $\Rightarrow$  V and V-of2 :: 'b  $\Rightarrow$  V and A B
using that by (auto simp: inj-on-def)
with small [where 'a='a] small [where 'a='b]
show OFCLASS('a  $\times$  'b, small-class)
  by intro-classes (smt (verit) down-raw f-inv-into-f set-eq-subset small-def)
qed

```

```

instance sum :: (small,small) small
proof –
  have inj (case-sum (Inl  $\circ$  V-of1) (Inr  $\circ$  V-of2))
    range (case-sum (Inl  $\circ$  V-of1) (Inr  $\circ$  V-of2))  $\leq$  elts (A  $\uplus$  B)
  if inj V-of1 inj V-of2 range V-of1  $\leq$  elts A range V-of2  $\leq$  elts B
  for V-of1 :: 'a  $\Rightarrow$  V and V-of2 :: 'b  $\Rightarrow$  V and A B
  using that by (force simp: inj-on-def split: sum.split)+
  with small [where 'a='a] small [where 'a='b]
  show OFCLASS('a + 'b, small-class)
    by intro-classes (metis down-raw replacement set-eq-subset small-def small-iff)
qed

```

```

instance option :: (small) small
proof –
  have inj ( $\lambda x. \text{case } x \text{ of } \text{None} \Rightarrow 0 \mid \text{Some } x \Rightarrow \text{ZFC-in-HOL.set } \{V\text{-of } x\}$ )
    range ( $\lambda x. \text{case } x \text{ of } \text{None} \Rightarrow 0 \mid \text{Some } x \Rightarrow \text{ZFC-in-HOL.set } \{V\text{-of } x\}$ )  $\leq$ 
  insert 0 (elts (VPow A))
  if inj V-of range V-of  $\leq$  elts A
  for V-of :: 'a  $\Rightarrow$  V and A
  using that by (auto simp: inj-on-def split: option.split-asm)
  with small [where 'a='a]
  show OFCLASS('a option, small-class)
    by intro-classes (smt (verit) down order.refl ex-inj small-iff small-image-iff
  small-insert)
qed

```

```

instance list :: (small) small
proof –
  have small (range (V-of-list V-of))
  if inj V-of range V-of  $\leq$  elts A
  for V-of :: 'a  $\Rightarrow$  V and A
proof –
  have range (V-of-list V-of)  $\approx$  (UNIV :: 'a list set)
    using that by (simp add: inj-V-of-list)
  also have ...  $\approx$  lists (UNIV :: 'a set)
    by simp
  also have ...  $\lesssim$  (UNIV :: 'a set)  $\times$  (UNIV :: nat set)
proof (cases finite (range (V-of::'a  $\Rightarrow$  V)))
  case True

```

```

    then have lists (UNIV :: 'a set)  $\lesssim$  (UNIV :: nat set)
    using countable-iff-lepoll countable-image-inj-on that(1) uncountable-infinite
  by blast
  then show ?thesis
    by (blast intro: lepoll-trans [OF - lepoll-times2])
  next
  case False
  then have lists (UNIV :: 'a set)  $\lesssim$  (UNIV :: 'a set)
    using  $\langle$ infinite (range V-of) $\rangle$  eqpoll-imp-lepoll infinite-epoll-lists by blast
  then show ?thesis
    using lepoll-times1 lepoll-trans by fastforce
  qed
  finally show ?thesis
    by (simp add: lepoll-imp-small)
  qed
  with small [where 'a='a]
  show OFCLASS('a list, small-class)
    by intro-classes (metis inj-V-of-list order.refl small-def small-iff small-iff-range)
  qed

instance fun :: (small,embeddable) embeddable
proof -
  have inj ( $\lambda f. VLambda A (\lambda x. V-of2 (f (inv V-of1 x)))$ )
    if *: inj V-of1 inj V-of2 range V-of1  $\leq$  elts A
    for V-of1 :: 'a  $\Rightarrow$  V and V-of2 :: 'b  $\Rightarrow$  V and A
  proof -
    have f u = f' u
      if VLambda A ( $\lambda u. V-of2 (f (inv V-of1 u))$ ) = VLambda A ( $\lambda x. V-of2 (f' (inv V-of1 x))$ )
    for f f' :: 'a  $\Rightarrow$  'b and u
      by (metis inv-f-f rangeI subsetD VLambda-eq-D2 [OF that, of V-of1 u] *)
    then show ?thesis
      by (auto simp: inj-on-def)
  qed
  then show OFCLASS('a  $\Rightarrow$  'b, embeddable-class)
    by intro-classes (metis embeddable-class.ex-inj small order-refl replacement small-iff)
  qed

instance fun :: (small,small) small
proof -
  have inj ( $\lambda f. VLambda A (\lambda x. V-of2 (f (inv V-of1 x)))$ ) (is inj ? $\varphi$ )
    range ( $\lambda f. VLambda A (\lambda x. V-of2 (f (inv V-of1 x)))$ )  $\leq$  elts (VPi A ( $\lambda x. B$ ))
    if *: inj V-of1 inj V-of2 range V-of1  $\leq$  elts A and range V-of2  $\leq$  elts B
    for V-of1 :: 'a  $\Rightarrow$  V and V-of2 :: 'b  $\Rightarrow$  V and A B
  proof -
    have f u = f' u
      if VLambda A ( $\lambda u. V-of2 (f (inv V-of1 u))$ ) = VLambda A ( $\lambda x. V-of2 (f' (inv V-of1 x))$ )
    (inv V-of1 x)))

```

```

    for  $f f' :: 'a \Rightarrow 'b$  and  $u$ 
    by (metis inv-f-f rangeI subsetD VLambda-eq-D2 [OF that, of V-of1 u] *)
  then show  $\text{inj } ?\varphi$ 
    by (auto simp: inj-on-def)
  show  $\text{range } ?\varphi \leq \text{elts } (VPi A (\lambda x. B))$ 
    using that by (simp add: VPi-I subset-eq)
qed
with small [where  $'a='a$ ] small [where  $'a='b$ ]
show OFCLASS( $'a \Rightarrow 'b$ , small-class)
  by intro-classes (smt (verit, best) down-raw order-refl imageE small-def)
qed

```

```

instance set :: (small) small
proof -
  have 1:  $\text{inj } (\lambda x. ZFC\text{-in-HOL.set } (V\text{-of } 'x))$ 
    if  $\text{inj } V\text{-of}$  for  $V\text{-of} :: 'a \Rightarrow V$ 
    by (simp add: inj-on-def inj-image-eq-iff [OF that])
  have 2:  $\text{range } (\lambda x. ZFC\text{-in-HOL.set } (V\text{-of } 'x)) \leq \text{elts } (VPow A)$ 
    if  $\text{range } V\text{-of} \leq \text{elts } A$  for  $V\text{-of} :: 'a \Rightarrow V$  and  $A$ 
    using that by (auto simp: inj-on-def image-subset-iff)
  from small [where  $'a='a$ ]
  show OFCLASS( $'a$  set, small-class)
    by intro-classes (metis 1 2 down-raw imageE small-def order-refl)
qed

```

```

instance real :: small
proof -
  have small (range (Rep-real))
    by simp
  then show OFCLASS(real, small-class)
    by intro-classes (metis Rep-real-inverse image-inv-f-f inj-on-def replacement)
qed

```

```

instance complex :: small
proof -
  have  $\bigwedge c. c \in \text{range } (\lambda(x,y). \text{Complex } x y)$ 
    by (metis case-prod-conv complex.exhaust-sel rangeI)
  then show OFCLASS(complex, small-class)
    by intro-classes (meson TC-small replacement smaller-than-small subset-eq)
qed

```

end

7 ZF sets corresponding to \mathbb{R} and \mathbb{C} and the cardinality of the continuum

```

theory General-Cardinals
  imports ZFC-Typeclasses HOL-Analysis.Continuum-Not-Denumerable

```

begin

7.1 Making the embedding from the type class explicit

definition $V\text{-of} :: 'a::\text{embeddable} \Rightarrow V$
where $V\text{-of} \equiv \text{SOME } f. \text{inj } f$

lemma $\text{inj-}V\text{-of}: \text{inj } V\text{-of}$
unfolding $V\text{-of-def}$ **by** (*metis embeddable-class.ex-inj some-eq-imp*)

declare $\text{inv-}f\text{-f}$ [*OF inj-V-of, simp*]

lemma $\text{inv-}V\text{-of-image-eq}$ [*simp*]: $\text{inv } V\text{-of } ' (V\text{-of } ' X) = X$
by (*auto simp: image-comp*)

lemma $\text{infinite-}V\text{-of}: \text{infinite } (UNIV::'a \text{ set}) \Longrightarrow \text{infinite } (\text{range } (V\text{-of}::'a::\text{embeddable} \Rightarrow V))$
using $\text{finite-imageD inj-}V\text{-of}$ **by** *blast*

lemma $\text{countable-}V\text{-of}: \text{countable } (\text{range } (V\text{-of}::'a::\text{countable} \Rightarrow V))$
by *blast*

lemma $\text{elts-set-}V\text{-of}: \text{small } X \Longrightarrow \text{elts } (ZFC\text{-in-HOL.set } (V\text{-of } ' X)) \approx X$
by (*metis inj-V-of inj-eq inj-on-def inj-on-image-epoll-self replacement set-of-elts small-iff*)

lemma $V\text{-of-image-times}: V\text{-of } ' (X \times Y) \approx (V\text{-of } ' X) \times (V\text{-of } ' Y)$

proof –

have $V\text{-of } ' (X \times Y) \approx X \times Y$

by (*meson inj-V-of inj-eq inj-onI inj-on-image-epoll-self*)

also have $\dots \approx (V\text{-of } ' X) \times (V\text{-of } ' Y)$

by (*metis eqpoll-sym inj-V-of inj-eq inj-onI inj-on-image-epoll-self times-epoll-cong*)

finally show *?thesis* .

qed

7.2 The cardinality of the continuum

definition $\text{Real-set} \equiv ZFC\text{-in-HOL.set } (\text{range } (V\text{-of}::\text{real} \Rightarrow V))$

definition $\text{Complex-set} \equiv ZFC\text{-in-HOL.set } (\text{range } (V\text{-of}::\text{complex} \Rightarrow V))$

definition $C\text{-continuum} \equiv \text{vcard } \text{Real-set}$

lemma $V\text{-of-Real-set}: \text{bij-betw } V\text{-of } (UNIV::\text{real set}) (\text{elts } \text{Real-set})$
by (*simp add: Real-set-def bij-betw-def inj-V-of*)

lemma $\text{uncountable-Real-set}: \text{uncountable } (\text{elts } \text{Real-set})$
using $V\text{-of-Real-set countable-iff-bij uncountable-UNIV-real}$ **by** *blast*

lemma $\text{Card } C\text{-continuum}$
by (*simp add: C-continuum-def Card-def*)

lemma *C-continuum-ge*: $C\text{-continuum} \geq \aleph_1$
by (*metis C-continuum-def Ord- ω_1 Ord-cardinal Ord-linear2 countable-iff-vcard-less1*
uncountable-Real-set)

lemma *V-of-Complex-set: bij-betw V-of (UNIV::complex set) (elts Complex-set)*
by (*simp add: Complex-set-def bij-betw-def inj-V-of*)

lemma *uncountable-Complex-set: uncountable (elts Complex-set)*
using *V-of-Complex-set countableI-bij2 uncountable-UNIV-complex* **by** *blast*

lemma *Complex-vcard: vcard Complex-set = C-continuum*

proof –

define *emb1* **where** $emb1 \equiv V\text{-of } o \text{ complex-of-real } o \text{ inv } V\text{-of}$

have *elts Real-set \approx elts Complex-set*

proof (*rule lepoll-antisym*)

show *elts Real-set \lesssim elts Complex-set*

unfolding *lepoll-def*

proof (*intro conjI exI*)

show *inj-on emb1 (elts Real-set)*

unfolding *emb1-def Real-set-def*

by (*simp add: inj-V-of inj-compose inj-of-real inj-on-imageI*)

show *emb1 ‘ elts Real-set \subseteq elts Complex-set*

by (*simp add: emb1-def Real-set-def Complex-set-def image-subset-iff*)

qed

define *emb2* **where** $emb2 \equiv (\lambda z. (V\text{-of } (Re\ z), V\text{-of } (Im\ z))) \text{ } o \text{ inv } V\text{-of}$

have *elts Complex-set \lesssim elts Real-set \times elts Real-set*

unfolding *lepoll-def*

proof (*intro conjI exI*)

show *inj-on emb2 (elts Complex-set)*

unfolding *emb2-def Complex-set-def inj-on-def*

by (*simp add: complex.expand inj-V-of inj-eq*)

show *emb2 ‘ elts Complex-set \subseteq elts Real-set \times elts Real-set*

by (*simp add: emb2-def Real-set-def Complex-set-def image-subset-iff*)

qed

also have $\dots \approx$ *elts Real-set*

by (*simp add: infinite-times-epoll-self uncountable-Real-set uncountable-infinite*)

finally show *elts Complex-set \lesssim elts Real-set .*

qed

then show *?thesis*

by (*simp add: C-continuum-def cardinal-cong*)

qed

lemma *gcard-Union-le-cmult*:

assumes *small U* **and** $\kappa: \bigwedge x. x \in U \implies gcard\ x \leq \kappa$ **and** *sm*: $\bigwedge x. x \in U \implies$
small x

shows $gcard (\bigcup U) \leq gcard\ U \otimes \kappa$

proof –

have $\exists f. f \in x \rightarrow elts\ \kappa \wedge inj\text{-on } f\ x$ **if** $x \in U$ **for** x

```

using  $\kappa$  [OF that] gcard-le-lepoll by (metis image-subset-iff-funcset lepoll-def sm that)
then obtain  $\varphi$  where  $\varphi: \bigwedge x. x \in U \implies (\varphi x) \in x \rightarrow \text{elts } \kappa \wedge \text{inj-on } (\varphi x) x$ 
by metis
define  $u$  where  $u \equiv \lambda y. @x. x \in U \wedge y \in x$ 
have  $u: u y \in U \wedge y \in (u y)$  if  $y \in \bigcup (U)$  for  $y$ 
unfolding u-def using that by (fast intro!: someI2)
define  $\psi$  where  $\psi \equiv \lambda y. (u y, \varphi (u y) y)$ 
have  $U: \text{elts } (\text{gcard } U) \approx U$ 
using assms by (simp add: gcard-epoll)
have  $\bigcup U \lesssim U \times \text{elts } \kappa$ 
unfolding lepoll-def
proof (intro exI conjI)
show inj-on  $\psi$  ( $\bigcup U$ )
using  $\varphi$   $u$  by (smt (verit)  $\psi$ -def inj-on-def prod.inject)
show  $\psi ' \bigcup U \subseteq U \times \text{elts } \kappa$ 
using  $\varphi$   $u$  by (auto simp:  $\psi$ -def)
qed
also have  $\dots \approx \text{elts } (\text{gcard } U \otimes \kappa)$ 
using U elts-cmult eqpoll-sym eqpoll-trans times-epoll-cong by blast
finally have  $(\bigcup U) \lesssim \text{elts } (\text{gcard } U \otimes \kappa)$  .
then show ?thesis
by (metis cardinal-idem cmult-def gcard-eq-vcard lepoll-imp-gcard-le small-elts)
qed

```

lemma *gcard-Times* [*simp*]: $\text{gcard } (X \times Y) = \text{gcard } X \otimes \text{gcard } Y$

proof (*cases small X \wedge small Y*)

case *True*

have $\text{elts } (\text{gcard } (X \times Y)) \approx X \times Y$

by (*simp add: True gcard-epoll*)

also have $\dots \approx \text{elts } (\text{gcard } X) \times \text{elts } (\text{gcard } Y)$

by (*simp add: True eqpoll-sym gcard-epoll times-epoll-cong*)

also have $\dots \approx \text{elts } (\text{gcard } X \otimes \text{gcard } Y)$

by (*simp add: elts-cmult eqpoll-sym*)

finally show *?thesis*

using *Card-cardinal-eq cmult-def gcardinal-cong* **by** *force*

next

case *False*

have $\text{gcard } (X \times Y) = 0$

by (*metis False Times-empty gcard-big-0 gcard-empty-0 small-Times-iff*)

then show *?thesis*

by (*metis False cmult-0 cmult-commute gcard-big-0*)

qed

7.3 Countable and uncountable sets

lemma *countable-iff-g-le-Aleph0*:

assumes *small X*

shows *countable X \longleftrightarrow gcard X \leq \aleph_0*

proof –
have $\text{countable } X \longleftrightarrow X \lesssim \text{elts } \omega$
by (*simp add: ω -def countable-iff-lepoll inj-ord-of-nat*)
also have $\dots \longleftrightarrow \text{gcard } X \leq \aleph_0$
using *Card- ω Card-def assms gcard-le-lepoll lepoll-imp-gcard-le* **by** *fastforce*
finally show *?thesis* .
qed

lemma *countable-imp-g-le-Aleph0*: $\text{countable } X \implies \text{gcard } X \leq \aleph_0$
by (*meson countable countable-iff-g-le-Aleph0*)

lemma *finite-iff-g-le-Aleph0*: $\text{small } X \implies \text{finite } X \longleftrightarrow \text{gcard } X < \aleph_0$
by (*metis Aleph-0 eqpoll-finite-iff finite-iff-less-Aleph0 gcard-eq-vcard gcard-eqpoll gcardinal-cong*)

lemma *finite-imp-g-le-Aleph0*: $\text{finite } X \implies \text{gcard } X < \aleph_0$
by (*meson finite-iff-g-le-Aleph0 finite-imp-small*)

lemma *countable-infinite-gcard*: $\text{countable } X \wedge \text{infinite } X \longleftrightarrow \text{gcard } X = \aleph_0$
proof –
have $\text{gcard } X = \omega$
if $\text{countable } X$ **and** $\text{infinite } X$
using *that countable countable-imp-g-le-Aleph0 finite-iff-g-le-Aleph0 less-V-def*
by *auto*
moreover have $\text{countable } X$ **if** $\text{gcard } X = \omega$
by (*metis Aleph-0 countable-iff-g-le-Aleph0 dual-order.refl gcard-big-0 omega-nonzero that*)
moreover have False **if** $\text{gcard } X = \omega$ **and** $\text{finite } X$
by (*metis Aleph-0 dual-order.irrefl finite-iff-g-le-Aleph0 finite-imp-small that*)
ultimately show *?thesis*
by *auto*
qed

lemma *uncountable-gcard*: $\text{small } X \implies \text{uncountable } X \longleftrightarrow \text{gcard } X > \aleph_0$
by (*simp add: Card-is-Ord Ord-not-le countable-iff-g-le-Aleph0*)

lemma *uncountable-gcard-ge*: $\text{small } X \implies \text{uncountable } X \longleftrightarrow \text{gcard } X \geq \aleph_1$
by (*simp add: uncountable-gcard csucc-le-Card-iff one-V-def*)

lemma *subset-smaller-gcard*:
assumes $\kappa: \kappa \leq \text{gcard } X$ *Card* κ
obtains Y **where** $Y \subseteq X$ $\text{gcard } Y = \kappa$
proof (*cases small X*)
case *True*
then have $\text{elts } \kappa \lesssim X$
by (*meson assms(1) eqpoll-imp-lepoll gcard-eqpoll lepoll-trans less-eq-V-def subset-imp-lepoll*)
then obtain Y **where** $Y \subseteq X$ $\text{elts } \kappa \approx Y$
by (*metis bij-betw-def eqpoll-def lepoll-def*)

```

then show ?thesis
  using Card-def ‹Card  $\kappa$ › gcardinal-cong that by force
next
  case False
  with assms show ?thesis
    by (metis antisym gcard-big-0 le-0 order-refl that)
qed

lemma Real-gcard: gcard (UNIV::real set) = C-continuum
  by (metis C-continuum-def V-of-Real-set bij-betw-def gcard-eq-vcard gcard-image)

lemma Complex-gcard: gcard (UNIV::complex set) = C-continuum
  by (metis Complex-vcard V-of-Complex-set bij-betw-def gcard-eq-vcard gcard-image)

end

```

8 Acknowledgements

The author was supported by the ERC Advanced Grant ALEXANDRIA (Project 742178) funded by the European Research Council.

References

- [1] L. Kirby. Addition and multiplication of sets. *Mathematical Logic Quarterly*, 53(1):52–65, 2007.
- [2] S. Obua. Partizan games in Isabelle/HOLZF. In K. Barkaoui, A. Cavalcanti, and A. Cerone, editors, *Theoretical Aspects of Computing — ICTAC 2006*, pages 272–286. Springer, 2006.