Countable Ordinals

Brian Huffman

March 17, 2025

Abstract

This development defines a well-ordered type of countable ordinals. It includes notions of continuous and normal functions, recursively defined functions over ordinals, least fixed-points, and derivatives. Much of ordinal arithmetic is formalized, including exponentials and logarithms. The development concludes with formalizations of Cantor Normal Form and Veblen hierarchies over normal functions.

Contents

1 Definition of Ordinals **2**
- 1.1 Preliminary datatype for ordinals 2
- 1.2 Ordinal type 4
- 1.3 Induction over ordinals 7

2 Ordinal Induction **9**
- 2.1 Zero and successor ordinals 9
  - 2.1.1 Derived properties of 0 and oSuc 10
- 2.2 Strict monotonicity 11
- 2.3 Limit ordinals 12
  - 2.3.1 Making strict monotonic sequences 13
- 2.4 Induction principle for ordinals 14

3 Continuity **15**
- 3.1 Continuous functions 15
- 3.2 Normal functions 16

4 Recursive Definitions **18**
- 4.1 Partial orders 19
- 4.2 Recursive definitions for *ordinal* $\Rightarrow$ *ordinal* 20

5 Ordinal Arithmetic **21**
5.1 Addition . . . 21
5.2 Subtraction . . . 23
5.3 Multiplication . . . 24
5.4 Exponentiation . . . 26

6 Inverse Functions **28**
6.1 Division . . . 30
6.2 Derived properties of division . . . 32
6.3 Logarithms . . . 32

7 Fixed-points **34**
7.1 Derivatives of ordinal functions . . . 36

8 Omega **37**
8.1 Embedding naturals in the ordinals . . . 37
8.2 Omega, the least limit ordinal . . . 38
8.3 Arithmetic properties of ω . . . 39
8.4 Additive principal ordinals . . . 40
8.5 Cantor normal form . . . 43
8.6 Epsilon 0 . . . 46

9 Veblen Hierarchies **46**
9.1 Closed, unbounded sets . . . 46
9.2 Ordering functions . . . 47
9.3 Critical ordinals . . . 49
9.4 Veblen hierarchy over a normal function . . . 51
9.5 Veblen hierarchy over $\lambda x.\ 1 + x$ . . . 53

1 Definition of Ordinals

theory *OrdinalDef*
  imports *Main*
begin

1.1 Preliminary datatype for ordinals

datatype $ord0 = ord0\text{-}Zero \mid ord0\text{-}Lim\ nat \Rightarrow ord0$

subterm ordering on ord0

definition
  $ord0\text{-}prec :: (ord0 \times ord0)\ set$ **where**
  $ord0\text{-}prec = (\bigcup f\ i.\ \{(f\ i,\ ord0\text{-}Lim\ f)\})$

lemma $wf\text{-}ord0\text{-}prec$: $wf\ ord0\text{-}prec$
proof $-$
  have $\forall x.\ (\forall y.\ (y,\ x) \in ord0\text{-}prec \longrightarrow P\ y) \longrightarrow P\ x \Longrightarrow P\ a$ **for** $P\ a$
    unfolding $ord0\text{-}prec\text{-}def$ **by** $(induction\ a)\ blast+$
  then show *?thesis*
    by $(metis\ wfUNIVI)$
qed

lemmas $ord0\text{-}prec\text{-}induct = wf\text{-}induct[OF\ wf\text{-}trancl[OF\ wf\text{-}ord0\text{-}prec]]$

less-than-or-equal ordering on ord0

inductive-set $ord0\text{-}leq :: (ord0 \times ord0)\ set$ **where**
  $[\![\forall a.\ (a,x) \in ord0\text{-}prec^{+} \longrightarrow (\exists b.\ (b,y) \in ord0\text{-}prec^{+} \wedge (a,b) \in ord0\text{-}leq)]\!]$
  $\Longrightarrow (x,y) \in ord0\text{-}leq$

lemma $ord0\text{-}leqI$:
  $[\![\forall a.\ (a,x) \in ord0\text{-}prec^{+} \longrightarrow (a,y) \in ord0\text{-}leq\ O\ ord0\text{-}prec^{+}]\!]$
 $\Longrightarrow (x,y) \in ord0\text{-}leq$
  by $(meson\ ord0\text{-}leq.intros\ relcomp.cases)$

lemma $ord0\text{-}leqD$:
  $[\![(x,y) \in ord0\text{-}leq;\ (a,x) \in ord0\text{-}prec^{+}]\!] \Longrightarrow (a,y) \in ord0\text{-}leq\ O\ ord0\text{-}prec^{+}$
  by $(ind\text{-}cases\ (x,y) \in ord0\text{-}leq,\ auto)$

lemma $ord0\text{-}leq\text{-}refl$: $(x,\ x) \in ord0\text{-}leq$
  by $(rule\ ord0\text{-}prec\text{-}induct,\ rule\ ord0\text{-}leqI,\ auto)$

lemma $ord0\text{-}leq\text{-}trans$:
  $(x,y) \in ord0\text{-}leq \Longrightarrow (y,z) \in ord0\text{-}leq \Longrightarrow (x,z) \in ord0\text{-}leq$
proof $(induction\ x\ arbitrary:\ y\ z\ rule:\ ord0\text{-}prec\text{-}induct)$
  case $(1\ x)$
  then show *?case*
    by $(meson\ ord0\text{-}leq.cases\ ord0\text{-}leq.intros)$
qed

lemma *wf-ord0-leq*: *wf* (*ord0-leq O ord0-prec*$^+$)
  unfolding *wf-def*
proof *clarify*
  fix *P x*
  assume $*$: $\forall x.\ (\forall y.\ (y, x) \in$ *ord0-leq O ord0-prec*$^+ \longrightarrow P\ y) \longrightarrow P\ x$
  have $\forall z.\ (z, x) \in$ *ord0-leq* $\longrightarrow P\ z$
    by (*rule ord0-prec-induct*) (*meson* $*$ *ord0-leq.cases ord0-leq-trans relcomp.cases*)
  then show *P x*
    by (*simp add*: *ord0-leq-refl*)
qed

ordering on ord0

instantiation *ord0* :: *ord*
begin

definition
  ord0-less-def: $x < y \longleftrightarrow (x,y) \in$ *ord0-leq O ord0-prec*$^+$

definition
  ord0-le-def: $x \le y \longleftrightarrow (x,y) \in$ *ord0-leq*

instance ..

end

lemma *ord0-order-refl*[*simp*]: $(x{::}ord0) \le x$
  by (*simp add*: *ord0-le-def ord0-leq-refl*)

lemma *ord0-order-trans*: $[\![(x{::}ord0) \le y;\ y \le z]\!] \Longrightarrow x \le z$
  using *ord0-le-def ord0-leq-trans* **by** *blast*

lemma *ord0-wf*: *wf* $\{(x,y{::}ord0).\ x < y\}$
  using *ord0-less-def wf-ord0-leq* **by** *auto*

lemmas *ord0-less-induct* = *wf-induct*[*OF ord0-wf*]

lemma *ord0-leI*: $[\![\forall a{::}ord0.\ a < x \longrightarrow a < y]\!] \Longrightarrow x \le y$
  by (*meson ord0-le-def ord0-leqD ord0-leqI ord0-leq-refl ord0-less-def*)

lemma *ord0-less-le-trans*: $[\![(x{::}ord0) < y;\ y \le z]\!] \Longrightarrow x < z$
  by (*meson ord0-le-def ord0-leq.cases ord0-leq-trans ord0-less-def relcomp.intros relcompEpair*)

lemma *ord0-le-less-trans*:
  $[\![(x{::}ord0) \le y;\ y < z]\!] \Longrightarrow x < z$
  by (*meson ord0-le-def ord0-leq-trans ord0-less-def relcomp.cases relcomp.intros*)

lemma *rev-ord0-le-less-trans*:

$[\![(y::ord0) < z; x \leq y]\!] \Longrightarrow x < z$
by (*rule ord0-le-less-trans*)

lemma *ord0-less-trans*: $[\![(x::ord0) < y; y < z]\!] \Longrightarrow x < z$
unfolding *ord0-less-def*
by (*meson ord0-leq.cases relcomp.cases relcompI*[*OF ord0-leq-trans trancl-trans*])

lemma *ord0-less-imp-le*: $(x::ord0) < y \Longrightarrow x \leq y$
using *ord0-leI ord0-less-trans* **by** *blast*

lemma *ord0-linear-lemma*:
fixes m :: *ord0* **and** n :: *ord0*
shows $m < n \lor n < m \lor (m \leq n \land n \leq m)$
proof −
have $m < n \lor n < m \lor m \leq n \land n \leq m$ **for** m
proof (*induction n arbitrary*: *m rule*: *ord0-less-induct*)
case (*1 n*)
have $\forall y. (y, n) \in \{(x, y). x < y\} \longrightarrow (\forall x. x < y \lor y < x \lor x \leq y \land y \leq x)$
$\Longrightarrow$
$m < n \lor n < m \lor m \leq n \land n \leq m$
proof (*induction m rule*: *ord0-less-induct*)
case (*1 x*)
then show *?case*
by (*smt* (*verit*, *best*) *mem-Collect-eq old.prod.case ord0-leI ord0-le-less-trans*
ord0-less-imp-le)
qed
then show *?case*
using *1* **by** *blast*
qed
then show *?thesis*
by *simp*
qed

lemma *ord0-linear*: $(x::ord0) \leq y \lor y \leq x$
using *ord0-less-imp-le ord0-linear-lemma* **by** *blast*

lemma *ord0-order-less-le*: $(x::ord0) < y \longleftrightarrow (x \leq y \land \neg\, y \leq x)$ (**is** *?L=?R*)
proof
show *?L* $\Longrightarrow$ *?R*
by (*metis ord0-less-def ord0-less-imp-le ord0-less-le-trans wf-not-refl wf-ord0-leq*)
show *?R* $\Longrightarrow$ *?L*
using *ord0-less-imp-le ord0-linear-lemma* **by** *blast*
qed

1.2 Ordinal type

definition
ord0rel :: (*ord0* × *ord0*) *set* **where**
ord0rel = $\{(x,y). x \leq y \land y \leq x\}$

typedef *ordinal* = (*UNIV*::*ord0 set*) // *ord0rel*
  by (*unfold quotient-def*, *auto*)

theorem *Abs-ordinal-cases2* [*case-names Abs-ordinal*, *cases type*: *ordinal*]:
  $(\bigwedge z.\ x = \textit{Abs-ordinal}\ (\textit{ord0rel}\ ``\ \{z\}) \Longrightarrow P) \Longrightarrow P$
  by (*cases x*, *auto simp add*: *quotient-def*)

instantiation *ordinal* :: *ord*
begin

definition
  ordinal-less-def: $x < y \longleftrightarrow (\forall\, a{\in}\textit{Rep-ordinal}\ x.\ \forall\, b{\in}\textit{Rep-ordinal}\ y.\ a < b)$

definition
  ordinal-le-def: $x \leq y \longleftrightarrow (\forall\, a{\in}\textit{Rep-ordinal}\ x.\ \forall\, b{\in}\textit{Rep-ordinal}\ y.\ a \leq b)$

instance ..

end

lemma *Rep-Abs-ord0rel* [*simp*]:
  $\textit{Rep-ordinal}\ (\textit{Abs-ordinal}\ (\textit{ord0rel}\ ``\ \{x\})) = (\textit{ord0rel}\ ``\ \{x\})$
  by (*simp add*: *Abs-ordinal-inverse quotientI*)

lemma *mem-ord0rel-Image* [*simp*, *intro*!]: $x \in \textit{ord0rel}\ ``\ \{x\}$
  by (*simp add*: *ord0rel-def*)

lemma *equiv-ord0rel*: *equiv UNIV ord0rel*
  unfolding *equiv-def refl-on-def sym-def trans-def ord0rel-def*
  by (*auto elim*: *ord0-order-trans*)

lemma *Abs-ordinal-eq*[*simp*]:
  $(\textit{Abs-ordinal}\ (\textit{ord0rel}\ ``\ \{x\}) = \textit{Abs-ordinal}\ (\textit{ord0rel}\ ``\ \{y\})) = (x \leq y \wedge y \leq x)$
 apply (*simp add*: *Abs-ordinal-inject quotientI eq-equiv-class-iff* [*OF equiv-ord0rel*])
 apply (*simp add*: *ord0rel-def*)
 done

lemma *Abs-ordinal-le*[*simp*]:
   $\textit{Abs-ordinal}\ (\textit{ord0rel}\ ``\ \{x\}) \leq \textit{Abs-ordinal}\ (\textit{ord0rel}\ ``\ \{y\}) \longleftrightarrow (x \leq y)$ (**is** *?L*=*?R*)
proof
 show $?L \Longrightarrow ?R$
  using *Rep-Abs-ord0rel ordinal-le-def* **by** *blast*
next
 assume *?R*
 then have $\bigwedge a\ b.\ [\![(x,\ a) \in \textit{ord0rel};\ (y,\ b) \in \textit{ord0rel}]\!] \Longrightarrow a \leq b$
  unfolding *ord0rel-def* **by** (*blast intro*: *ord0-order-trans*)

then show *?L*
by (*auto simp add*: *ordinal-le-def*)
qed

lemma *Abs-ordinal-less*[*simp*]:
Abs-ordinal (*ord0rel* `` $\{x\}$) $<$ *Abs-ordinal* (*ord0rel* `` $\{y\}$) $\longleftrightarrow$ ($x < y$) (**is** *?L=?R*)
proof
show *?L* $\Longrightarrow$ *?R*
using *Rep-Abs-ord0rel ordinal-less-def* **by** *blast*
next
assume *?R*
then have $\bigwedge a\ b.\ [\![(x,\ a) \in ord0rel;\ (y,\ b) \in ord0rel]\!] \Longrightarrow a < b$
unfolding *ord0rel-def*
by (*blast intro*: *ord0-le-less-trans ord0-less-le-trans*)
then show *?L*
by (*auto simp add*: *ordinal-less-def*)
qed

instance *ordinal* :: *linorder*
proof
show (x::*ordinal*) $\leq x$ **for** x
by (*cases* x, *simp*)
show ((x::*ordinal*) $< y$) $=$ ($x \leq y \wedge \neg\ y \leq x$) **for** x y
by (*cases* x, *cases* y, *auto simp add*: *ord0-order-less-le*)
show (x::*ordinal*) $\leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$ **for** x y z
by (*cases* x, *cases* y, *cases* z, *auto elim*: *ord0-order-trans*)
show (x::*ordinal*) $\leq y \Longrightarrow y \leq x \Longrightarrow x = y$ **for** x y
by (*cases* x, *cases* y, *simp*)
show (x::*ordinal*) $\leq y \vee y \leq x$ **for** x y
by (*cases* x, *cases* y, *simp add*: *ord0-linear*)
qed

instance *ordinal* :: *wellorder*
proof
show $P\ a$ **if** ($\bigwedge x$::*ordinal*. ($\bigwedge y.\ y < x \Longrightarrow P\ y$) $\Longrightarrow P\ x$) **for** P a
proof (*rule Abs-ordinal-cases2*)
fix z
assume a: $a =$ *Abs-ordinal* (*ord0rel* `` $\{z\}$)
have P (*Abs-ordinal* (*ord0rel* `` $\{z\}$))
using *that*
apply (*rule ord0-less-induct*)
by (*metis Abs-ordinal-cases2 Abs-ordinal-less CollectI case-prodI*)
with a **show** $P\ a$ **by** *simp*
qed
qed

lemma *ordinal-linear*: (x::*ordinal*) $\leq y \vee y \leq x$
by *auto*

lemma *ordinal-wf*: *wf* $\{(x,y::ordinal).\ x < y\}$
by (*simp add*: *wf*)

1.3 Induction over ordinals

zero and strict limits

definition
oZero :: *ordinal* **where**
oZero = *Abs-ordinal* (*ord0rel* ‘‘ {*ord0-Zero*})

definition
oStrictLimit :: (*nat* ⇒ *ordinal*) ⇒ *ordinal* **where**
oStrictLimit f = *Abs-ordinal*
(*ord0rel* ‘‘ {*ord0-Lim* (λ*n*. *SOME x*. *x* ∈ *Rep-ordinal* (*f n*))})

induction over ordinals

lemma *ord0relD*: $(x,y) \in ord0rel \Longrightarrow x \le y \wedge y \le x$
by (*simp add*: *ord0rel-def*)

lemma *ord0-precD*: $(x,y) \in ord0\text{-}prec \Longrightarrow \exists f\ n.\ x = f\ n \wedge y = ord0\text{-}Lim\ f$
by (*simp add*: *ord0-prec-def*)

lemma *less-ord0-LimI*: *f n* < *ord0-Lim f*
using *ord0-leq-refl ord0-less-def ord0-prec-def* **by** *fastforce*

lemma *less-ord0-LimD*:
assumes *x* < *ord0-Lim f* **shows** ∃ *n*. *x* ≤ *f n*
proof −
obtain *y* **where** *x*≤*y* *y* < *ord0-Lim f*
using *assms ord0-linear* **by** *auto*
then consider (*y*, *ord0-Lim f*) ∈ *ord0-prec* | *z* **where** *y* ≤ *z* (*z*, *ord0-Lim f*) ∈ *ord0-prec*
apply (*clarsimp simp add*: *ord0-less-def ord0-le-def*)
by (*metis ord0-less-def ord0-less-imp-le relcomp.relcompI that*(*2*) *tranclE*)
then show *?thesis*
by (*metis* ‹*x* ≤ *y*› *ord0.inject ord0-order-trans ord0-precD*)
qed

lemma *some-ord0rel*: (*x*, *SOME y*. (*x*,*y*) ∈ *ord0rel*) ∈ *ord0rel*
by (*rule-tac x*=*x* **in** *someI*, *simp add*: *ord0rel-def*)

lemma *ord0-Lim-le*: ∀ *n*. *f n* ≤ *g n* ⟹ *ord0-Lim f* ≤ *ord0-Lim g*
by (*metis less-ord0-LimD less-ord0-LimI ord0-le-less-trans ord0-linear ord0-order-less-le*)

lemma *ord0-Lim-ord0rel*:
∀ *n*. (*f n*, *g n*) ∈ *ord0rel* ⟹ (*ord0-Lim f*, *ord0-Lim g*) ∈ *ord0rel*
by (*simp add*: *ord0rel-def ord0-Lim-le*)

```
lemma Abs-ordinal-oStrictLimit:
  Abs-ordinal (ord0rel `` {ord0-Lim f})
  = oStrictLimit (λn. Abs-ordinal (ord0rel `` {f n}))
  apply (simp add: oStrictLimit-def)
  using ord0-Lim-le ord0relD some-ord0rel by presburger

lemma oStrictLimit-induct:
  assumes base: P oZero
  assumes step: ⋀f. ∀ n. P (f n) ⟹ P (oStrictLimit f)
  shows P a
proof −
  obtain z where z: a = Abs-ordinal (ord0rel `` {z})
    using Abs-ordinal-cases2 by auto
  have P (Abs-ordinal (ord0rel `` {z}))
  proof (induction z)
    case ord0-Zero
    with base oZero-def show ?case by auto
  next
    case (ord0-Lim x)
    then show ?case
      by (simp add: Abs-ordinal-oStrictLimit local.step)
  qed
  then show ?thesis
    by (simp add: z)
qed
```

order properties of 0 and strict limits

```
lemma oZero-least: oZero ≤ x
proof −
  have x = Abs-ordinal (ord0rel `` {z}) ⟹ ord0-Zero ≤ z for z
  proof (induction z arbitrary: x)
    case (ord0-Lim u)
    then show ?case
      by (meson less-ord0-LimI ord0-le-less-trans ord0-less-imp-le rangeI)
  qed auto
  then show ?thesis
    by (metis Abs-ordinal-cases2 Abs-ordinal-le oZero-def)
qed

lemma oStrictLimit-ub: f n < oStrictLimit f
  apply (cases f n, simp add: oStrictLimit-def)
  apply (rule-tac y=SOME x. x ∈ Rep-ordinal (f n) in ord0-le-less-trans)
 apply (metis (no-types) Image-singleton-iff Rep-Abs-ord0rel empty-iff mem-ord0rel-Image
ord0relD some-in-eq)
  by (meson less-ord0-LimI)

lemma oStrictLimit-lub:
  assumes ∀ n. f n < x shows oStrictLimit f ≤ x
proof −
```

have $\exists\, n.\ x \leq f\ n$ **if** x: $x < oStrictLimit\ f$
proof $-$
obtain z **where** z: $x = Abs\text{-}ordinal\ (ord0rel\ ``\ \{z\})$
$z < ord0\text{-}Lim\ (\lambda n.\ SOME\ x.\ x \in Rep\text{-}ordinal\ (f\ n))$
using *less-ord0-LimI x* **unfolding** *oStrictLimit-def*
by (*metis Abs-ordinal-cases2 Abs-ordinal-less*)
then obtain n **where** $z \leq (SOME\ x.\ x \in Rep\text{-}ordinal\ (f\ n))$
using *less-ord0-LimD* **by** *blast*
then have $Abs\text{-}ordinal\ (ord0rel\ ``\ \{z\}) \leq f\ n$
apply (*rule-tac x=f n* **in** *Abs-ordinal-cases2*)
using *ord0-order-trans ord0relD some-ord0rel* **by** *auto*
then show *?thesis*
using ‹$x = Abs\text{-}ordinal\ (ord0rel\ ``\ \{z\})$› **by** *auto*
qed
then show *?thesis*
using *assms linorder-not-le* **by** *blast*
qed

lemma *less-oStrictLimitD*: $x < oStrictLimit\ f \Longrightarrow \exists\, n.\ x \leq f\ n$
by (*metis leD leI oStrictLimit-lub*)

end

2 Ordinal Induction

theory *OrdinalInduct*
imports *OrdinalDef*
begin

2.1 Zero and successor ordinals

definition
$oSuc :: ordinal \Rightarrow ordinal$ **where**
$oSuc\ x = oStrictLimit\ (\lambda n.\ x)$

lemma *less-oSuc*[*iff*]: $x < oSuc\ x$
by (*metis oStrictLimit-ub oSuc-def*)

lemma *oSuc-leI*: $x < y \Longrightarrow oSuc\ x \leq y$
by (*simp add*: *oStrictLimit-lub oSuc-def*)

instantiation *ordinal* :: {*zero*, *one*}
begin

definition
ordinal-zero-def: $(0::ordinal) = oZero$

definition
ordinal-one-def [*simp*]: $(1::ordinal) = oSuc\ 0$

instance ..

end

2.1.1 Derived properties of 0 and oSuc

lemma *less-oSuc-eq-le*: $(x < oSuc\ y) = (x \leq y)$
by (*metis dual-order.strict-trans1 less-oSuc linorder-not-le oSuc-leI*)

lemma *ordinal-0-le* [*iff*]: $0 \leq (x{::}ordinal)$
by (*simp add*: *oZero-least ordinal-zero-def*)

lemma *ordinal-not-less-0* [*iff*]: $\neg\ (x{::}ordinal) < 0$
by (*simp add*: *linorder-not-less*)

lemma *ordinal-le-0* [*iff*]: $(x \leq 0) = (x = (0{::}ordinal))$
by (*simp add*: *order-le-less*)

lemma *ordinal-neq-0* [*iff*]: $(x \neq 0) = (0 < (x{::}ordinal))$
by (*simp add*: *order-less-le*)

lemma *ordinal-not-0-less* [*iff*]: $(\neg\ 0 < x) = (x = (0{::}ordinal))$
by (*simp add*: *linorder-not-less*)

lemma *oSuc-le-eq-less*: $(oSuc\ x \leq y) = (x < y)$
by (*meson leD leI less-oSuc-eq-le*)

lemma *zero-less-oSuc* [*iff*]: $0 < oSuc\ x$
by (*rule order-le-less-trans*, *rule ordinal-0-le*, *rule less-oSuc*)

lemma *oSuc-not-0* [*iff*]: $oSuc\ x \neq 0$
by *simp*

lemma *less-oSuc0* [*iff*]: $(x < oSuc\ 0) = (x = 0)$
by (*simp add*: *less-oSuc-eq-le*)

lemma *oSuc-less-oSuc* [*iff*]: $(oSuc\ x < oSuc\ y) = (x < y)$
by (*simp add*: *less-oSuc-eq-le oSuc-le-eq-less*)

lemma *oSuc-eq-oSuc* [*iff*]: $(oSuc\ x = oSuc\ y) = (x = y)$
by (*metis less-oSuc less-oSuc-eq-le order-antisym*)

lemma *oSuc-le-oSuc* [*iff*]: $(oSuc\ x \leq oSuc\ y) = (x \leq y)$
by (*simp add*: *order-le-less*)

lemma *le-oSucE*:
$[\![x \leq oSuc\ y;\ x \leq y \Longrightarrow R;\ x = oSuc\ y \Longrightarrow R]\!] \Longrightarrow R$
by (*auto simp add*: *order-le-less less-oSuc-eq-le*)

lemma *less-oSucE*:
  $[\![x < oSuc\ y;\ x < y \Longrightarrow P;\ x = y \Longrightarrow P]\!] \Longrightarrow P$
  by (*auto simp add*: *less-oSuc-eq-le order-le-less*)

2.2 Strict monotonicity

locale *strict-mono* =
  fixes f
  assumes *strict-mono*: $A < B \Longrightarrow f\ A < f\ B$

lemmas *strict-monoI* = *strict-mono.intro*
  and *strict-monoD* = *strict-mono.strict-mono*

lemma *strict-mono-natI*:
  fixes $f :: nat \Rightarrow 'a::order$
  shows $(\bigwedge n.\ f\ n < f\ (Suc\ n)) \Longrightarrow strict\text{-}mono\ f$
  using *OrdinalInduct.strict-monoI lift-Suc-mono-less* **by** *blast*

lemma *mono-natI*:
  fixes $f :: nat \Rightarrow 'a::order$
  shows $(\bigwedge n.\ f\ n \le f\ (Suc\ n)) \Longrightarrow mono\ f$
  by (*simp add*: *mono-iff-le-Suc*)

lemma *strict-mono-mono*:
  fixes $f :: 'a::order \Rightarrow 'b::order$
  shows $strict\text{-}mono\ f \Longrightarrow mono\ f$
  by (*auto intro*!: *monoI simp add*: *order-le-less strict-monoD*)

lemma *strict-mono-monoD*:
  fixes $f :: 'a::order \Rightarrow 'b::order$
  shows $[\![strict\text{-}mono\ f;\ A \le B]\!] \Longrightarrow f\ A \le f\ B$
  by (*rule monoD*[*OF strict-mono-mono*])

lemma *strict-mono-cancel-eq*:
  fixes $f :: 'a::linorder \Rightarrow 'b::linorder$
  shows $strict\text{-}mono\ f \Longrightarrow (f\ x = f\ y) = (x = y)$
  by (*metis OrdinalInduct.strict-monoD not-less-iff-gr-or-eq*)

lemma *strict-mono-cancel-less*:
  fixes $f :: 'a::linorder \Rightarrow 'b::linorder$
  shows $strict\text{-}mono\ f \Longrightarrow (f\ x < f\ y) = (x < y)$
  using *OrdinalInduct.strict-monoD linorder-neq-iff* **by** *fastforce*

lemma *strict-mono-cancel-le*:
  fixes $f :: 'a::linorder \Rightarrow 'b::linorder$
  shows $strict\text{-}mono\ f \Longrightarrow (f\ x \le f\ y) = (x \le y)$
  by (*meson linorder-not-less strict-mono-cancel-less*)

2.3 Limit ordinals

definition
oLimit :: (*nat* ⇒ *ordinal*) ⇒ *ordinal* **where**
oLimit f = (*LEAST k*. ∀ *n*. *f n* ≤ *k*)

lemma *oLimit-leI*: ∀ *n*. *f n* ≤ *x* ⟹ *oLimit f* ≤ *x*
by (*simp add*: *oLimit-def wellorder-Least-lemma*(*2*))

lemma *le-oLimit* [*iff*]: *f n* ≤ *oLimit f*
by (*smt* (*verit*, *best*) *LeastI-ex leD oLimit-def oStrictLimit-ub ordinal-linear*)

lemma *le-oLimitI*: *x* ≤ *f n* ⟹ *x* ≤ *oLimit f*
by (*erule order-trans*, *rule le-oLimit*)

lemma *less-oLimitI*: *x* < *f n* ⟹ *x* < *oLimit f*
by (*erule order-less-le-trans*, *rule le-oLimit*)

lemma *less-oLimitD*: *x* < *oLimit f* ⟹ ∃ *n*. *x* < *f n*
by (*meson linorder-not-le oLimit-leI*)

lemma *less-oLimitE*: ⟦*x* < *oLimit f*; ⋀*n*. *x* < *f n* ⟹ *P*⟧ ⟹ *P*
by (*auto dest*: *less-oLimitD*)

lemma *le-oLimitE*:
⟦*x* ≤ *oLimit f*; ⋀*n*. *x* ≤ *f n* ⟹ *R*; *x* = *oLimit f* ⟹ *R*⟧ ⟹ *R*
by (*auto simp add*: *order-le-less dest*: *less-oLimitD*)

lemma *oLimit-const* [*simp*]: *oLimit* (λ*n*. *x*) = *x*
by (*meson dual-order.refl le-oLimit oLimit-leI order-antisym*)

lemma *strict-mono-less-oLimit*: *strict-mono f* ⟹ *f n* < *oLimit f*
by (*meson OrdinalInduct.strict-monoD lessI less-oLimitI*)

lemma *oLimit-eqI*:
⟦⋀*n*. ∃ *m*. *f n* ≤ *g m*; ⋀*n*. ∃ *m*. *g n* ≤ *f m*⟧ ⟹ *oLimit f* = *oLimit g*
by (*meson le-oLimitI nle-le oLimit-leI*)

lemma *oLimit-Suc*:
f 0 < *oLimit f* ⟹ *oLimit* (λ*n*. *f* (*Suc n*)) = *oLimit f*
by (*smt* (*verit*, *ccfv-SIG*) *linorder-not-le nle-le oLimit-eqI oLimit-leI old.nat.exhaust*)

lemma *oLimit-shift*:
∀ *n*. *f n* < *oLimit f* ⟹ *oLimit* (λ*n*. *f* (*n* + *k*)) = *oLimit f*
apply (*induct-tac k*, *simp*)
by (*metis* (*no-types*, *lifting*) *add-Suc-shift leD le-oLimit less-oLimitD not-less-iff-gr-or-eq oLimit-Suc*)

lemma *oLimit-shift-mono*:
mono f ⟹ *oLimit* (λ*n*. *f* (*n* + *k*)) = *oLimit f*

by (*meson le-add1 monoD oLimit-eqI*)

limit ordinal predicate

definition
limit-ordinal :: *ordinal* $\Rightarrow$ *bool* **where**
limit-ordinal $x \longleftrightarrow (x \neq 0) \wedge (\forall y.\ x \neq oSuc\ y)$

lemma *limit-ordinal-not-0* [*simp*]: $\neg$ *limit-ordinal 0*
by (*simp add*: *limit-ordinal-def*)

lemma *zero-less-limit-ordinal* [*simp*]: *limit-ordinal* $x \Longrightarrow 0 < x$
by (*simp add*: *limit-ordinal-def*)

lemma *limit-ordinal-not-oSuc* [*simp*]: $\neg$ *limit-ordinal* (*oSuc p*)
by (*simp add*: *limit-ordinal-def*)

lemma *oSuc-less-limit-ordinal*:
limit-ordinal $x \Longrightarrow (oSuc\ w < x) = (w < x)$
by (*metis limit-ordinal-not-oSuc oSuc-le-eq-less order-le-less*)

lemma *limit-ordinal-oLimitI*:
$\forall n.\ f\ n < oLimit\ f \Longrightarrow$ *limit-ordinal* (*oLimit f*)
by (*metis less-oLimitD less-oSuc less-oSucE limit-ordinal-def order-less-imp-triv ordinal-neq-0*)

lemma *strict-mono-limit-ordinal*:
strict-mono f $\Longrightarrow$ *limit-ordinal* (*oLimit f*)
by (*simp add*: *limit-ordinal-oLimitI strict-mono-less-oLimit*)

lemma *limit-ordinalI*:
$[\![0 < z; \forall x{<}z.\ oSuc\ x < z]\!] \Longrightarrow$ *limit-ordinal z*
using *limit-ordinal-def* **by** *blast*

2.3.1 Making strict monotonic sequences

primrec *make-mono* :: (*nat* $\Rightarrow$ *ordinal*) $\Rightarrow$ *nat* $\Rightarrow$ *nat*
where
make-mono f 0 $= 0$
| *make-mono f* (*Suc n*) = (*LEAST x*. *f* (*make-mono f n*) < *f x*)

lemma *f-make-mono-less*:
$\forall n.\ f\ n < oLimit\ f \Longrightarrow f\ (make\text{-}mono\ f\ n) < f\ (make\text{-}mono\ f\ (Suc\ n))$
by (*metis less-oLimitD make-mono.simps*(*2*) *wellorder-Least-lemma*(*1*))

lemma *strict-mono-f-make-mono*:
$\forall n.\ f\ n < oLimit\ f \Longrightarrow strict\text{-}mono\ (\lambda n.\ f\ (make\text{-}mono\ f\ n))$
by (*rule strict-mono-natI*, *erule f-make-mono-less*)

lemma *le-f-make-mono*:

$[\![\forall n.\ f\ n < oLimit\ f;\ m \le make\text{-}mono\ f\ n]\!] \Longrightarrow f\ m \le f\ (make\text{-}mono\ f\ n)$
apply (*auto simp add*: *order-le-less*)
apply (*case-tac n*, *simp-all*)
by (*metis LeastI less-oLimitD linorder-le-less-linear not-less-Least order-le-less-trans*)

lemma *make-mono-less*:
$\forall n.\ f\ n < oLimit\ f \Longrightarrow make\text{-}mono\ f\ n < make\text{-}mono\ f\ (Suc\ n)$
by (*meson f-make-mono-less le-f-make-mono linorder-not-less*)

declare *make-mono.simps* [*simp del*]

lemma *oLimit-make-mono-eq*:
assumes $\forall n.\ f\ n < oLimit\ f$ **shows** $oLimit\ (\lambda n.\ f\ (make\text{-}mono\ f\ n)) = oLimit\ f$
proof −
have $k \le make\text{-}mono\ f\ k$ **for** *k*
by (*induction k*) (*auto simp*: *Suc-leI assms make-mono-less order-le-less-trans*)
then show *?thesis*
by (*meson assms le-f-make-mono oLimit-eqI*)
qed

2.4 Induction principle for ordinals

lemma *oLimit-le-oStrictLimit*: $oLimit\ f \le oStrictLimit\ f$
by (*simp add*: *oLimit-leI oStrictLimit-ub order-less-imp-le*)

lemma *oLimit-induct* [*case-names zero suc lim*]:
assumes *zero*: *P 0*
and *suc*: $\bigwedge x.\ P\ x \Longrightarrow P\ (oSuc\ x)$
and *lim*: $\bigwedge f.\ [\![strict\text{-}mono\ f;\ \forall n.\ P\ (f\ n)]\!] \Longrightarrow P\ (oLimit\ f)$
shows *P a*
apply (*rule oStrictLimit-induct*)
apply (*rule zero*[*unfolded ordinal-zero-def*])
apply (*cut-tac f=f* **in** *oLimit-le-oStrictLimit*)
apply (*simp add*: *order-le-less*, *erule disjE*)
apply (*metis dual-order.order-iff-strict leD le-oLimit less-oStrictLimitD oSuc-le-eq-less suc*)
by (*metis lim oLimit-make-mono-eq oStrictLimit-ub strict-mono-f-make-mono*)

lemma *ordinal-cases* [*case-names zero suc lim*]:
assumes *zero*: $a = 0 \Longrightarrow P$
and *suc*: $\bigwedge x.\ a = oSuc\ x \Longrightarrow P$
and *lim*: $\bigwedge f.\ [\![strict\text{-}mono\ f;\ a = oLimit\ f]\!] \Longrightarrow P$
shows *P*
apply (*subgoal-tac* $\forall x.\ a = x \longrightarrow P$, *force*)
apply (*rule allI*)
apply (*rule-tac a=x* **in** *oLimit-induct*)
apply (*rule impI*, *erule zero*)
apply (*rule impI*, *erule suc*)
apply (*rule impI*, *erule lim*, *assumption*)

done

end

3 Continuity

theory *OrdinalCont*
  imports *OrdinalInduct*
begin

3.1 Continuous functions

locale *continuous* =
  fixes F :: *ordinal* $\Rightarrow$ *ordinal*
  assumes *cont*: F (*oLimit* f) = *oLimit* ($\lambda n.$ F (f n))

lemmas *continuousD* = *continuous.cont*

lemma (**in** *continuous*) *monoD*: **assumes** $x \leq y$ **shows** $F\ x \leq F\ y$
proof −
  have *oLimit* (*case-nat* u ($\lambda n.$ v)) = *max* u v **for** u v
    apply (*simp add*: *max-def*)
    by (*metis* (*no-types*, *lifting*) *le-oLimit less-oLimitE linorder-not-le oLimit-Suc nat.case order-le-less*)
  then show $F\ x \leq F\ y$
    by (*metis* ‹$x \leq y$› *cont le-oLimit max.absorb2 nat.case*(*1*))
qed

lemma (**in** *continuous*) *mono*: *mono* F
  by (*simp add*: *local.monoD monoI*)

lemma *continuousI*:
  assumes *lim*: $\bigwedge f.$ *strict-mono* $f \Longrightarrow F$ (*oLimit* f) = *oLimit* ($\lambda n.$ F (f n))
  assumes *suc*: $\bigwedge x.$ $F\ x \leq F$ (*oSuc* x)
  shows *continuous* F
proof −
  have *mono*: $x \leq y \Longrightarrow F\ x \leq F\ y$ **for** x y
  proof (*induction* y *arbitrary*: x *rule*: *oLimit-induct*)
    case *zero*
    then show *?case* **by** *auto*
  next
    case (*suc* x)
    with *assms* **show** *?case*
      by (*metis antisym-conv1 le-oSucE nless-le order.trans*)
  next
    case (*lim* f)
    with *assms* **show** *?case* **thm** *assms*(*1*)
      by (*metis le-oLimitI nle-le oLimit-leI*)
  qed

have F (*oLimit f*) = *oLimit* ($\lambda n.$ F (f n)) **for** f
proof (*cases* $\forall n.$ *f n* < *oLimit f*)
case *True*
then have §: *oLimit* ($\lambda n.$ f (*make-mono f n*)) = *oLimit f*
by (*simp add*: *oLimit-make-mono-eq*)
have $\bigwedge n.$ $\exists m.$ F (*f n*) $\leq$ F (f (*make-mono f m*))
by (*metis True mono less-oLimitD linorder-not-less oLimit-make-mono-eq ordinal-linear*)
then show *?thesis*
by (*metis True* § *oLimit-eqI lim strict-mono-f-make-mono*)
next
case *False*
then show *?thesis*
by (*metis le-oLimit less-oLimitE linorder-not-le mono nle-le*)
qed
with *mono* **show** *?thesis*
by (*simp add*: *continuous.intro*)
qed

3.2 Normal functions

locale *normal* = *continuous* +
assumes *strict*: *strict-mono F*

lemma (**in** *normal*) *mono*: *mono F*
by (*rule mono*)

lemma (**in** *normal*) *continuous*: *continuous F*
by (*rule continuous.intro*, *rule cont*)

lemma (**in** *normal*) *monoD*: $x \leq y \Longrightarrow F\ x \leq F\ y$
by (*rule monoD*)

lemma (**in** *normal*) *strict-monoD*: $x < y \Longrightarrow F\ x < F\ y$
by (*erule strict-monoD*[*OF strict*])

lemma (**in** *normal*) *cancel-eq*: $(F\ x = F\ y) = (x = y)$
by (*rule strict-mono-cancel-eq*[*OF strict*])

lemma (**in** *normal*) *cancel-less*: $(F\ x < F\ y) = (x < y)$
by (*rule strict-mono-cancel-less*[*OF strict*])

lemma (**in** *normal*) *cancel-le*: $(F\ x \leq F\ y) = (x \leq y)$
by (*rule strict-mono-cancel-le*[*OF strict*])

lemma (**in** *normal*) *oLimit*: F (*oLimit f*) = *oLimit* ($\lambda n.$ F (f n))
by (*rule cont*)

lemma (**in** *normal*) *increasing*: $x \leq F\ x$

proof (*induction x rule*: *oLimit-induct*)
  case *zero*
  then show *?case*
    by *simp*
next
  case (*suc x*)
  then show *?case*
   by (*simp add*: *normal.strict-monoD normal-axioms oSuc-leI order.strict-trans1*)
next
  case (*lim f*)
  then show *?case*
    by (*metis cont le-oLimitI oLimit-leI*)
qed

lemma *normalI*:
  assumes *lim*: $\bigwedge f.$ *strict-mono f* $\Longrightarrow F\ (oLimit\ f) = oLimit\ (\lambda n.\ F\ (f\ n))$
  assumes *suc*: $\bigwedge x.\ F\ x < F\ (oSuc\ x)$
  shows *normal F*
proof −
  have *mono*: $x \leq y \Longrightarrow F\ x \leq F\ y$ **for** *x y*
    using *continuousI assms*
    by (*metis continuous.monoD linorder-not-less ordinal-linear*)
  then have *OrdinalInduct.strict-mono F*
    by (*metis OrdinalInduct.strict-monoI leD oSuc-leI order-less-le suc*)
  then show *?thesis*
    by (*meson continuousI leD lim nle-le normal.intro normal-axioms.intro suc*)
qed

lemma *normal-range-le*:
  assumes *nml*: *normal F normal G* **and** *range G* $\subseteq$ *range F*
  shows $F\ x \leq G\ x$
proof (*induction x rule*: *oLimit-induct*)
  case *zero*
  with *assms* **show** *?case*
    by (*metis image-iff normal.monoD ordinal-0-le range-subsetD*)
next
  case (*suc x*)
  then have $G\ (oSuc\ x) \in$ *range F*
    using *assms*(*3*) **by** *blast*
  then show *?case*
     by (*smt* (*verit*, *ccfv-SIG*) *nml dual-order.trans leD le-oSucE less-oSuc normal.cancel-le ordinal-linear rangeE suc*)
next
  case (*lim f*)
  then show *?case*
    by (*metis nml le-oLimitI normal.oLimit oLimit-leI*)
qed

lemma *normal-range-eq*:

$[\![$*normal F*; *normal G*; *range F* = *range G*$]\!] \Longrightarrow$ *F* = *G*
by (*force simp add*: *normal-range-le intro*: *order-antisym*)

end

4 Recursive Definitions

theory *OrdinalRec*
imports *OrdinalCont*
begin

definition
oPrec :: *ordinal* ⇒ *ordinal* **where**
oPrec x = (*THE p. x* = *oSuc p*)

lemma *oPrec-oSuc* [*simp*]: *oPrec* (*oSuc x*) = *x*
by (*simp add*: *oPrec-def*)

lemma *oPrec-less*: $\exists$ *p. x* = *oSuc p* $\Longrightarrow$ *oPrec x* < *x*
by *clarsimp*

definition
ordinal-rec0 ::
[′*a*, *ordinal* ⇒ ′*a* ⇒ ′*a*, (*nat* ⇒ ′*a*) ⇒ ′*a*, *ordinal*] ⇒ ′*a* **where**
ordinal-rec0 z s l ≡ *wfrec* {(*x*,*y*). *x* < *y*} (λ*F x*.
if x = *0 then z else*
if (∃ *p. x* = *oSuc p*) *then s* (*oPrec x*) (*F* (*oPrec x*)) *else*
(*THE y*. ∀ *f*. (∀ *n. f n* < *oLimit f*) ∧ *oLimit f* = *x*
⟶ *l* (λ*n. F* (*f n*)) = *y*))

lemma *ordinal-rec0-0* [*simp*]: *ordinal-rec0 z s l 0* = *z*
by (*simp add*: *cut-apply def-wfrec*[*OF ordinal-rec0-def wf*])

lemma *ordinal-rec0-oSuc*: *ordinal-rec0 z s l* (*oSuc x*) = *s x* (*ordinal-rec0 z s l x*)
by (*simp add*: *cut-apply def-wfrec*[*OF ordinal-rec0-def wf*])

lemma *limit-ordinal-not-0*: *limit-ordinal x* $\Longrightarrow$ *x* ≠ *0* **and** *limit-ordinal-not-oSuc*:
limit-ordinal x $\Longrightarrow$ *x* ≠ *oSuc p*
by *auto*

lemma *ordinal-rec0-limit-ordinal*:
limit-ordinal x $\Longrightarrow$ *ordinal-rec0 z s l x* =
(*THE y*. ∀ *f*. (∀ *n. f n* < *oLimit f*) ∧ *oLimit f* = *x* ⟶
l (λ*n. ordinal-rec0 z s l* (*f n*)) = *y*)
apply (*rule trans*[*OF def-wfrec*[*OF ordinal-rec0-def wf*]])
apply (*simp add*: *limit-ordinal-not-oSuc limit-ordinal-not-0*)
apply (*rule-tac f*=*The* **in** *arg-cong*, *rule ext*)
apply (*rule-tac f*=*All* **in** *arg-cong*, *rule ext*)

apply *safe*
apply (*simp add*: *cut-apply*)
apply (*simp add*: *cut-apply*)
done

4.1 Partial orders

locale *porder* =
fixes *le* :: $'a \Rightarrow 'a \Rightarrow bool$ (**infixl** ‹<<› *55*)
assumes *po-refl*: $\bigwedge x.\ x << x$
and *po-trans*: $\bigwedge x\ y\ z.\ [\![x << y;\ y << z]\!] \Longrightarrow x << z$
and *po-antisym*: $\bigwedge x\ y.\ [\![x << y;\ y << x]\!] \Longrightarrow x = y$

lemma *porder-order*: *porder* $((\leq) :: 'a{::}order \Rightarrow 'a \Rightarrow bool)$
using *porder-def* **by** *fastforce*

lemma (**in** *porder*) *flip*: *porder* $(\lambda x\ y.\ y << x)$
by (*metis* (*no-types*, *lifting*) *po-antisym po-refl po-trans porder-def*)

locale *omega-complete* = *porder* +
fixes *lub* :: $(nat \Rightarrow 'a) \Rightarrow 'a$
assumes *is-ub-lub*: $\bigwedge f\ n.\ f\ n << lub\ f$
assumes *is-lub-lub*: $\bigwedge f\ x.\ \forall n.\ f\ n << x \Longrightarrow lub\ f << x$

lemma (**in** *omega-complete*) *lub-cong-lemma*:
$[\![\forall n.\ f\ n < oLimit\ f;\ \forall m.\ g\ m < oLimit\ g;\ oLimit\ f \leq oLimit\ g;$
$\forall y{<}oLimit\ g.\ \forall x{\leq}y.\ F\ x << F\ y]\!]$
$\Longrightarrow lub\ (\lambda n.\ F\ (f\ n)) << lub\ (\lambda n.\ F\ (g\ n))$
apply (*rule is-lub-lub*[*rule-format*])
by (*metis dual-order.trans is-ub-lub leD linorder-le-cases oLimit-leI po-trans*)

lemma (**in** *omega-complete*) *lub-cong*:
$[\![\forall n.\ f\ n < oLimit\ f;\ \forall m.\ g\ m < oLimit\ g;\ oLimit\ f = oLimit\ g;$
$\forall y{<}oLimit\ g.\ \forall x{\leq}y.\ F\ x << F\ y]\!]$
$\Longrightarrow lub\ (\lambda n.\ F\ (f\ n)) = lub\ (\lambda n.\ F\ (g\ n))$
by (*simp add*: *lub-cong-lemma po-antisym*)

lemma (**in** *omega-complete*) *ordinal-rec0-mono*:
assumes *s*: $\forall p\ x.\ x << s\ p\ x$ **and** $x \leq y$
shows *ordinal-rec0 z s lub x* << *ordinal-rec0 z s lub y*
proof −
have $\forall y{\leq}w.\ \forall x{\leq}y.$ *ordinal-rec0 z s lub x* << *ordinal-rec0 z s lub y* **for** *w*
proof (*induction w rule*: *oLimit-induct*)
case *zero*
then show *?case*
by (*simp add*: *po-refl*)
next
case (*suc x*)

then show *?case*
by (*metis le-oSucE oSuc-le-oSuc ordinal-rec0-oSuc po-refl po-trans s*)
next
case (*lim f*)
then have $\forall g.\ (\forall n.\ g\ n < oLimit\ g) \wedge oLimit\ g = oLimit\ f \longrightarrow$
$lub\ (\lambda n.\ ordinal\text{-}rec0\ z\ s\ lub\ (g\ n)) =$
$lub\ (\lambda n.\ ordinal\text{-}rec0\ z\ s\ lub\ (f\ n))$
by (*metis linorder-not-le lub-cong oLimit-leI order-le-less strict-mono-less-oLimit*)
with *lim* **have** $ordinal\text{-}rec0\ z\ s\ lub\ (oLimit\ f) =$
$lub\ (\lambda n.\ ordinal\text{-}rec0\ z\ s\ lub\ (f\ n))$
apply (*simp add*: *ordinal-rec0-limit-ordinal strict-mono-limit-ordinal*)
by (*smt* (*verit*, *del-insts*) *the-equality strict-mono-less-oLimit*)
then show *?case*
by (*smt* (*verit*, *ccfv-SIG*) *is-ub-lub le-oLimitE lim.IH order-le-less po-refl po-trans*)
qed
with *assms* **show** *?thesis*
by *blast*
qed

lemma (**in** *omega-complete*) *ordinal-rec0-oLimit*:
assumes s: $\forall p\ x.\ x << s\ p\ x$
shows $ordinal\text{-}rec0\ z\ s\ lub\ (oLimit\ f) =$
$lub\ (\lambda n.\ ordinal\text{-}rec0\ z\ s\ lub\ (f\ n))$
proof (*cases* $\forall n.\ f\ n < oLimit\ f$)
case *True*
then show *?thesis*
apply (*simp add*: *ordinal-rec0-limit-ordinal limit-ordinal-oLimitI*)
apply (*rule the-equality*)
apply (*metis lub-cong omega-complete.ordinal-rec0-mono omega-complete-axioms s*)
by *simp*
next
case *False*
then show *?thesis*
apply (*simp add*: *linorder-not-less*, *clarify*)
by (*smt* (*verit*, *best*) *is-lub-lub is-ub-lub le-oLimit ordinal-rec0-mono po-antisym s*)
qed

4.2 Recursive definitions for *ordinal* ⇒ *ordinal*

definition
ordinal-rec ::
[*ordinal*, *ordinal* ⇒ *ordinal* ⇒ *ordinal*, *ordinal*] ⇒ *ordinal* **where**
$ordinal\text{-}rec\ z\ s = ordinal\text{-}rec0\ z\ s\ oLimit$

lemma *omega-complete-oLimit*: *omega-complete* (≤) *oLimit*
by (*simp add*: *oLimit-leI omega-complete-axioms-def omega-complete-def porder-order*)

lemma *ordinal-rec-0* [*simp*]: *ordinal-rec z s 0* = *z*
  by (*simp add*: *ordinal-rec-def*)

lemma *ordinal-rec-oSuc* [*simp*]:
  ordinal-rec z s (*oSuc x*) = *s x* (*ordinal-rec z s x*)
  by (*unfold ordinal-rec-def*, *rule ordinal-rec0-oSuc*)

lemma *ordinal-rec-oLimit*:
  assumes *s*: $\forall$ *p x*. *x* $\leq$ *s p x*
  shows *ordinal-rec z s* (*oLimit f*) = *oLimit* (λ*n*. *ordinal-rec z s* (*f n*))
  by (*metis omega-complete.ordinal-rec0-oLimit omega-complete-oLimit ordinal-rec-def s*)

lemma *continuous-ordinal-rec*:
  assumes *s*: $\forall$ *p x*. *x* $\leq$ *s p x*
  shows *continuous* (*ordinal-rec z s*)
  by (*simp add*: *continuous.intro ordinal-rec-oLimit s*)

lemma *mono-ordinal-rec*:
  assumes *s*: $\forall$ *p x*. *x* $\leq$ *s p x*
  shows *mono* (*ordinal-rec z s*)
  by (*simp add*: *continuous.mono continuous-ordinal-rec s*)

lemma *normal-ordinal-rec*:
  assumes *s*: $\forall$ *p x*. *x* < *s p x*
  shows *normal* (*ordinal-rec z s*)
  by (*simp add*: *assms continuous.cont continuous-ordinal-rec less-imp-le normalI*)

end

5 Ordinal Arithmetic

theory *OrdinalArith*
imports *OrdinalRec*
begin

5.1 Addition

instantiation *ordinal* :: *plus*
begin

definition
  (+) = (λ*x*. *ordinal-rec x* (λ*p*. *oSuc*))

instance ..

end

lemma *normal-plus*: *normal* ((+) *x*)
by (*simp add*: *plus-ordinal-def normal-ordinal-rec*)

lemma *ordinal-plus-0* [*simp*]: $x + 0 = (x::ordinal)$
by (*simp add*: *plus-ordinal-def*)

lemma *ordinal-plus-oSuc* [*simp*]: $x + oSuc\ y = oSuc\ (x + y)$
by (*simp add*: *plus-ordinal-def*)

lemma *ordinal-plus-oLimit* [*simp*]: $x + oLimit\ f = oLimit\ (\lambda n.\ x + f\ n)$
by (*simp add*: *normal.oLimit normal-plus*)

lemma *ordinal-0-plus* [*simp*]: $0 + x = (x::ordinal)$
by (*rule-tac a=x* **in** *oLimit-induct*, *simp-all*)

lemma *ordinal-plus-assoc*: $(x + y) + z = x + (y + z::ordinal)$
by (*rule-tac a=z* **in** *oLimit-induct*, *simp-all*)

lemma *ordinal-plus-monoL* [*rule-format*]:
$\forall x\ x'.\ x \le x' \longrightarrow x + y \le x' + (y::ordinal)$
apply (*rule-tac a=y* **in** *oLimit-induct*, *simp-all*)
apply *clarify*
apply (*rule oLimit-leI*, *clarify*)
apply (*rule-tac n=n* **in** *le-oLimitI*)
apply *simp*
done

lemma *ordinal-plus-monoR*: $y \le y' \Longrightarrow x + y \le x + (y'::ordinal)$
by (*rule normal.monoD*[*OF normal-plus*])

lemma *ordinal-plus-mono*:
$[\![x \le x';\ y \le y']\!] \Longrightarrow x + y \le x' + (y'::ordinal)$
by (*rule order-trans*[*OF ordinal-plus-monoL ordinal-plus-monoR*])

lemma *ordinal-plus-strict-monoR*: $y < y' \Longrightarrow x + y < x + (y'::ordinal)$
by (*rule normal.strict-monoD*[*OF normal-plus*])

lemma *ordinal-le-plusL* [*simp*]: $y \le x + (y::ordinal)$
by (*cut-tac ordinal-plus-monoL*[*OF ordinal-0-le*], *simp*)

lemma *ordinal-le-plusR* [*simp*]: $x \le x + (y::ordinal)$
by (*cut-tac ordinal-plus-monoR*[*OF ordinal-0-le*], *simp*)

lemma *ordinal-less-plusR*: $0 < y \Longrightarrow x < x + (y::ordinal)$
by (*drule-tac ordinal-plus-strict-monoR*, *simp*)

lemma *ordinal-plus-left-cancel* [*simp*]:
$(w + x = w + y) = (x = (y::ordinal))$
by (*rule normal.cancel-eq*[*OF normal-plus*])

lemma *ordinal-plus-left-cancel-le* [*simp*]:
$(w + x \leq w + y) = (x \leq (y{::}ordinal))$
by (*rule normal.cancel-le*[*OF normal-plus*])

lemma *ordinal-plus-left-cancel-less* [*simp*]:
$(w + x < w + y) = (x < (y{::}ordinal))$
by (*rule normal.cancel-less*[*OF normal-plus*])

lemma *ordinal-plus-not-0*: $(0 < x + y) = (0 < x \lor 0 < (y{::}ordinal))$
by (*metis ordinal-le-0 ordinal-le-plusL ordinal-neq-0 ordinal-plus-0*)

lemma *not-inject*: $(\neg P) = (\neg Q) \Longrightarrow P = Q$
by *auto*

lemma *ordinal-plus-eq-0*:
$((x{::}ordinal) + y = 0) = (x = 0 \land y = 0)$
by (*rule not-inject*, *simp add*: *ordinal-plus-not-0*)

5.2 Subtraction

instantiation *ordinal* :: *minus*
begin

definition
minus-ordinal-def:
$x - y = ordinal\text{-}rec\ 0\ (\lambda p\ w.\ if\ y \leq p\ then\ oSuc\ w\ else\ w)\ x$

instance ..

end

lemma *continuous-minus*: *continuous* $(\lambda x.\ x - y)$
unfolding *minus-ordinal-def*
by (*simp add*: *continuous-ordinal-rec order-less-imp-le*)

lemma *ordinal-0-minus* [*simp*]: $0 - x = (0{::}ordinal)$
by (*simp add*: *minus-ordinal-def*)

lemma *ordinal-oSuc-minus* [*simp*]: $y \leq x \Longrightarrow oSuc\ x - y = oSuc\ (x - y)$
by (*simp add*: *minus-ordinal-def*)

lemma *ordinal-oLimit-minus* [*simp*]: $oLimit\ f - y = oLimit\ (\lambda n.\ f\ n - y)$
by (*rule continuousD*[*OF continuous-minus*])

lemma *ordinal-minus-0* [*simp*]: $x - 0 = (x{::}ordinal)$
by (*rule-tac a=x* **in** *oLimit-induct*, *simp-all*)

lemma *ordinal-oSuc-minus2*: $x < y \Longrightarrow oSuc\ x - y = x - y$

by (*simp add*: *minus-ordinal-def linorder-not-le*[*symmetric*])

lemma *ordinal-minus-eq-0* [*rule-format*, *simp*]:
$x \le y \longrightarrow x - y = (0::ordinal)$
apply (*rule-tac a=x* **in** *oLimit-induct*)
apply *simp*
apply (*simp add*: *ordinal-oSuc-minus2 order-less-imp-le oSuc-le-eq-less*)
apply (*simp add*: *order-trans*[*OF le-oLimit*])
done

lemma *ordinal-plus-minus1* [*simp*]: $(x + y) - x = (y::ordinal)$
by (*rule-tac a=y* **in** *oLimit-induct*, *simp-all*)

lemma *ordinal-plus-minus2* [*simp*]: $x \le y \Longrightarrow x + (y - x) = (y::ordinal)$
apply (*subgoal-tac* $\forall z.\ y < x + z \longrightarrow x + (y - x) = y$)
apply (*drule-tac x=oSuc y* **in** *spec*, *erule mp*)
apply (*rule order-less-le-trans*[*OF less-oSuc*], *simp*)
apply (*rule allI*, *rule-tac a=z* **in** *oLimit-induct*)
apply (*simp add*: *linorder-not-less*[*symmetric*])
apply (*clarsimp simp add*: *less-oSuc-eq-le*)
apply (*clarsimp*, *drule less-oLimitD*, *clarsimp*)
done

lemma *ordinal-minusI*: $x = y + z \Longrightarrow x - y = (z::ordinal)$
by *simp*

lemma *ordinal-minus-less-eq* [*simp*]:
$(y::ordinal) \le x \Longrightarrow (x - y < z) = (x < y + z)$
by (*metis ordinal-plus-left-cancel-less ordinal-plus-minus2*)

lemma *ordinal-minus-le-eq* [*simp*]: $(x - y \le z) = (x \le y + (z::ordinal))$
proof (*rule linorder-le-cases*)
assume $x \le y$ **then show** *?thesis*
using *order-trans* **by** *force*
next
assume $y \le x$ **then show** *?thesis*
by (*metis ordinal-plus-left-cancel-le ordinal-plus-minus2*)
qed

lemma *ordinal-minus-monoL*: $x \le y \Longrightarrow x - z \le y - (z::ordinal)$
by (*erule continuous.monoD*[*OF continuous-minus*])

lemma *ordinal-minus-monoR*: $x \le y \Longrightarrow z - y \le z - (x::ordinal)$
by (*metis linorder-le-cases order-trans ordinal-minus-le-eq ordinal-plus-monoL*)

5.3 Multiplication

instantiation *ordinal* :: *times*
begin

definition
  times-ordinal-def: $(*) = (\lambda x.\ \textit{ordinal-rec}\ 0\ (\lambda p\ w.\ w + x))$

instance ..

end

lemma *continuous-times*: *continuous* $((*)\ x)$
  by (*simp add*: *times-ordinal-def continuous-ordinal-rec*)

lemma *normal-times*: $0 < x \Longrightarrow \textit{normal}\ ((*)\ x)$
  unfolding *times-ordinal-def*
  by (*simp add*: *normal-ordinal-rec ordinal-less-plusR*)

lemma *ordinal-times-0* [*simp*]: $x * 0 = (0::\textit{ordinal})$
  by (*simp add*: *times-ordinal-def*)

lemma *ordinal-times-oSuc* [*simp*]: $x * \textit{oSuc}\ y = (x * y) + x$
  by (*simp add*: *times-ordinal-def*)

lemma *ordinal-times-oLimit* [*simp*]: $x * \textit{oLimit}\ f = \textit{oLimit}\ (\lambda n.\ x * f\ n)$
  by (*simp add*: *times-ordinal-def ordinal-rec-oLimit*)

lemma *ordinal-0-times* [*simp*]: $0 * x = (0::\textit{ordinal})$
  by (*rule-tac a=x* **in** *oLimit-induct*, *simp-all*)

lemma *ordinal-1-times* [*simp*]: $\textit{oSuc}\ 0 * x = (x::\textit{ordinal})$
  by (*rule-tac a=x* **in** *oLimit-induct*, *simp-all*)

lemma *ordinal-times-1* [*simp*]: $x * \textit{oSuc}\ 0 = (x::\textit{ordinal})$
  by *simp*

lemma *ordinal-times-distrib*:
  $x * (y + z) = (x * y) + (x * z::\textit{ordinal})$
  by (*rule-tac a=z* **in** *oLimit-induct*, *simp-all add*: *ordinal-plus-assoc*)

lemma *ordinal-times-assoc*:
  $(x * y::\textit{ordinal}) * z = x * (y * z)$
  by (*rule-tac a=z* **in** *oLimit-induct*, *simp-all add*: *ordinal-times-distrib*)

lemma *ordinal-times-monoL* [*rule-format*]:
  $\forall x\ x'.\ x \le x' \longrightarrow x * y \le x' * (y::\textit{ordinal})$
  apply (*rule-tac a=y* **in** *oLimit-induct*)
    apply *simp*
   apply *clarify*
   apply (*simp add*: *ordinal-plus-mono*)
  apply *clarsimp*
  apply (*rule oLimit-leI*, *clarify*)

apply (*rule-tac n=n* **in** *le-oLimitI*)
apply *simp*
done

lemma *ordinal-times-monoR*: $y \leq y' \Longrightarrow x * y \leq x * (y'::ordinal)$
by (*rule continuous.monoD*[*OF continuous-times*])

lemma *ordinal-times-mono*:
$[\![x \leq x'; y \leq y']\!] \Longrightarrow x * y \leq x' * (y'::ordinal)$
by (*rule order-trans*[*OF ordinal-times-monoL ordinal-times-monoR*])

lemma *ordinal-times-strict-monoR*:
$[\![y < y'; 0 < x]\!] \Longrightarrow x * y < x * (y'::ordinal)$
by (*rule normal.strict-monoD*[*OF normal-times*])

lemma *ordinal-le-timesL* [*simp*]: $0 < x \Longrightarrow y \leq x * (y::ordinal)$
by (*drule ordinal-times-monoL*[*OF oSuc-leI*], *simp*)

lemma *ordinal-le-timesR* [*simp*]: $0 < y \Longrightarrow x \leq x * (y::ordinal)$
by (*drule ordinal-times-monoR*[*OF oSuc-leI*], *simp*)

lemma *ordinal-less-timesR*: $[\![0 < x; oSuc\ 0 < y]\!] \Longrightarrow x < x * (y::ordinal)$
by (*drule ordinal-times-strict-monoR*, *assumption*, *simp*)

lemma *ordinal-times-left-cancel* [*simp*]:
$0 < w \Longrightarrow (w * x = w * y) = (x = (y::ordinal))$
by (*rule normal.cancel-eq*[*OF normal-times*])

lemma *ordinal-times-left-cancel-le* [*simp*]:
$0 < w \Longrightarrow (w * x \leq w * y) = (x \leq (y::ordinal))$
by (*rule normal.cancel-le*[*OF normal-times*])

lemma *ordinal-times-left-cancel-less* [*simp*]:
$0 < w \Longrightarrow (w * x < w * y) = (x < (y::ordinal))$
by (*rule normal.cancel-less*[*OF normal-times*])

lemma *ordinal-times-eq-0*:
$((x::ordinal) * y = 0) = (x = 0 \vee y = 0)$
by (*metis ordinal-0-times ordinal-neq-0 ordinal-times-0 ordinal-times-strict-monoR*)

lemma *ordinal-times-not-0* [*simp*]:
$((0::ordinal) < x * y) = (0 < x \wedge 0 < y)$
by (*metis ordinal-neq-0 ordinal-times-eq-0*)

5.4 Exponentiation

definition
exp-ordinal :: [*ordinal*, *ordinal*] ⇒ *ordinal* (**infixr** ‹∗∗› *75*) **where**
$(**) = (\lambda x.\ if\ 0 < x\ then\ ordinal\text{-}rec\ 1\ (\lambda p\ w.\ w * x)$

else $(\lambda y.\ if\ y = 0\ then\ 1\ else\ 0))$

lemma *continuous-exp*: $0 < x \Longrightarrow continuous\ ((**)\ x)$
by (*simp add*: *exp-ordinal-def continuous-ordinal-rec*)

lemma *ordinal-exp-0* [*simp*]: $x ** 0 = (1::ordinal)$
by (*simp add*: *exp-ordinal-def*)

lemma *ordinal-exp-oSuc* [*simp*]: $x ** oSuc\ y = (x ** y) * x$
by (*simp add*: *exp-ordinal-def*)

lemma *ordinal-exp-oLimit* [*simp*]:
$0 < x \Longrightarrow x ** oLimit\ f = oLimit\ (\lambda n.\ x ** f\ n)$
by (*rule continuousD*[*OF continuous-exp*])

lemma *ordinal-0-exp* [*simp*]: $0 ** x = (if\ x = 0\ then\ 1\ else\ 0)$
by (*simp add*: *exp-ordinal-def*)

lemma *ordinal-1-exp* [*simp*]: $oSuc\ 0 ** x = oSuc\ 0$
by (*rule-tac a*=*x* **in** *oLimit-induct*, *simp-all*)

lemma *ordinal-exp-1* [*simp*]: $x ** oSuc\ 0 = x$
by *simp*

lemma *ordinal-exp-distrib*:
$x ** (y + z) = (x ** y) * (x ** (z::ordinal))$
apply (*case-tac* $x = 0$, *simp-all add*: *ordinal-plus-not-0*)
apply (*rule-tac a*=*z* **in** *oLimit-induct*, *simp-all add*: *ordinal-times-assoc*)
done

lemma *ordinal-exp-not-0* [*simp*]: $(0 < x ** y) = (0 < x \lor y = 0)$
apply *auto*
apply (*erule contrapos-pp*, *simp*)
apply (*rule-tac a*=*y* **in** *oLimit-induct*, *simp-all*)
apply (*rule less-oLimitI*, *erule spec*)
done

lemma *ordinal-exp-eq-0* [*simp*]: $(x ** y = 0) = (x = 0 \land 0 < y)$
by (*rule not-inject*, *simp*)

lemma *ordinal-exp-assoc*:
$(x ** y) ** z = x ** (y * z)$
apply (*case-tac* $x = 0$, *simp-all*)
apply (*rule-tac a*=*z* **in** *oLimit-induct*, *simp-all add*: *ordinal-exp-distrib*)
done

lemma *ordinal-exp-monoL* [*rule-format*]:
$\forall x\ x'.\ x \le x' \longrightarrow x ** y \le x' ** (y::ordinal)$
apply (*rule-tac a*=*y* **in** *oLimit-induct*)

```
   apply simp
  apply (simp add: ordinal-times-mono)
 apply clarsimp
 apply (case-tac x = 0, simp)
 apply (case-tac x' = 0, simp-all)
 apply (rule oLimit-leI, clarify)
 apply (rule-tac n=n in le-oLimitI)
 apply simp
 done

lemma normal-exp: oSuc 0 < x ⟹ normal ((**) x)
 using order-less-trans[OF less-oSuc]
 by (simp add: normalI ordinal-less-timesR)

lemma ordinal-exp-monoR:
 ⟦0 < x; y ≤ y'⟧ ⟹ x ** y ≤ x ** (y'::ordinal)
 by (rule continuous.monoD[OF continuous-exp])

lemma ordinal-exp-mono:
 ⟦0 < x'; x ≤ x'; y ≤ y'⟧ ⟹ x ** y ≤ x' ** (y'::ordinal)
 by (rule order-trans[OF ordinal-exp-monoL ordinal-exp-monoR])

lemma ordinal-exp-strict-monoR:
 ⟦oSuc 0 < x; y < y'⟧ ⟹ x ** y < x ** (y'::ordinal)
 by (rule normal.strict-monoD[OF normal-exp])

lemma ordinal-le-expR [simp]: 0 < y ⟹ x ≤ x ** (y::ordinal)
 by (metis leI nless-le oSuc-le-eq-less ordinal-exp-1 ordinal-exp-mono ordinal-le-0)

lemma ordinal-exp-left-cancel [simp]:
 oSuc 0 < w ⟹ (w ** x = w ** y) = (x = y)
 by (rule normal.cancel-eq[OF normal-exp])

lemma ordinal-exp-left-cancel-le [simp]:
 oSuc 0 < w ⟹ (w ** x ≤ w ** y) = (x ≤ y)
 by (rule normal.cancel-le[OF normal-exp])

lemma ordinal-exp-left-cancel-less [simp]:
 oSuc 0 < w ⟹ (w ** x < w ** y) = (x < y)
 by (rule normal.cancel-less[OF normal-exp])

end
```

6 Inverse Functions

```
theory OrdinalInverse
imports OrdinalArith
begin
```

lemma (**in** *normal*) *oInv-ex*:
  assumes $F\ 0 \le a$ **shows** $\exists\, q.\ F\ q \le a \wedge a < F\ (oSuc\ q)$
proof −
  have $a < F\ z \Longrightarrow (\exists\, q{<}z.\ F\ q \le a \wedge a < F\ (oSuc\ q))$ **for** z
  proof (*induction z rule*: *oLimit-induct*)
    case *zero*
    then show *?case*
      using *assms* **by** *auto*
  next
    case (*suc x*)
    then show *?case*
      by (*metis less-oSuc linorder-not-le order-less-trans*)
  next
    case (*lim f*)
    then show *?case*
      by (*metis less-oLimitD less-oLimitI oLimit*)
  qed
  then show *?thesis*
    by (*metis increasing oSuc-le-eq-less*)
qed

lemma *oInv-uniq*:
  assumes $mono\ (F{::}ordinal \Rightarrow ordinal)\ F\ x \le a\ a < F\ (oSuc\ x)\ F\ y \le a\ a < F\ (oSuc\ y)$
  shows $x = y$
proof (*cases* $x{<}y$)
  case *True*
  with *assms* **show** *?thesis*
    by (*meson dual-order.trans leD monoD oSuc-leI*)
next
  case *False*
  with *assms* **show** *?thesis*
    by (*meson dual-order.strict-trans2 less-oSucE mono-strict-invE*)
qed

definition
  $oInv :: (ordinal \Rightarrow ordinal) \Rightarrow ordinal \Rightarrow ordinal$ **where**
  $oInv\ F\ a = (if\ F\ 0 \le a\ then\ (THE\ x.\ F\ x \le a \wedge a < F\ (oSuc\ x))\ else\ 0)$

lemma (**in** *normal*) *oInv-bounds*: $F\ 0 \le a \Longrightarrow F\ (oInv\ F\ a) \le a \wedge a < F\ (oSuc\ (oInv\ F\ a))$
  by (*simp add*: *oInv-def*) (*metis* (*no-types*, *lifting*) *theI′ mono oInv-ex oInv-uniq*)

lemma (**in** *normal*) *oInv-bound1*:
  $F\ 0 \le a \Longrightarrow F\ (oInv\ F\ a) \le a$
  by (*rule oInv-bounds*[*THEN conjunct1*])

lemma (**in** *normal*) *oInv-bound2*: $a < F\ (oSuc\ (oInv\ F\ a))$
  by (*metis cancel-less linorder-not-le oInv-bounds order.strict-trans ordinal-not-0-less*)

lemma (**in** *normal*) *oInv-equality*: $[\![F\ x \le a;\ a < F\ (oSuc\ x)]\!] \Longrightarrow oInv\ F\ a = x$
by (*meson mono normal.cancel-le normal-axioms oInv-bound1 oInv-bound2 oInv-uniq ordinal-0-le order-trans*)

lemma (**in** *normal*) *oInv-inverse*: $oInv\ F\ (F\ x) = x$
by (*rule oInv-equality*, *simp-all add*: *cancel-less*)

lemma (**in** *normal*) *oInv-equality'*: $a = F\ x \Longrightarrow oInv\ F\ a = x$
by (*simp add*: *oInv-inverse*)

lemma (**in** *normal*) *oInv-eq-0*: $a \le F\ 0 \Longrightarrow oInv\ F\ a = 0$
by (*metis nle-le oInv-def oInv-equality'*)

lemma (**in** *normal*) *oInv-less*: $[\![F\ 0 \le a;\ a < F\ z]\!] \Longrightarrow oInv\ F\ a < z$
using *cancel-less oInv-bound1* **by** *fastforce*

lemma (**in** *normal*) *le-oInv*: $F\ z \le a \Longrightarrow z \le oInv\ F\ a$
by (*metis cancel-le dual-order.trans le-oSucE linorder-not-less oInv-bound2 order-le-less*)

lemma (**in** *normal*) *less-oInvD*: $x < oInv\ F\ a \Longrightarrow F\ (oSuc\ x) \le a$
by (*metis* (*no-types*) *linorder-not-le nle-le oInv-eq-0 oInv-less oSuc-leI ordinal-0-le*)

lemma (**in** *normal*) *oInv-le*: $a < F\ (oSuc\ x) \Longrightarrow oInv\ F\ a \le x$
by (*metis leD less-oInvD nle-le order-le-less*)

lemma (**in** *normal*) *mono-oInv*: *mono* (*oInv F*)
proof
  fix $x\ y$ :: *ordinal*
  assume $x \le y$
  show $oInv\ F\ x \le oInv\ F\ y$
  proof (*rule linorder-le-cases* [*of x F 0*])
    assume $x \le F\ 0$ **then show** *?thesis* **by** (*simp add*: *oInv-eq-0*)
  next
    assume $F\ 0 \le x$ **show** *?thesis*
      by (*rule le-oInv*, *simp only*: ‹$x \le y$› ‹$F\ 0 \le x$› *order-trans* [*OF oInv-bound1*])
  qed
qed

lemma (**in** *normal*) *oInv-decreasing*: $F\ 0 \le x \Longrightarrow oInv\ F\ x \le x$
by (*meson dual-order.trans increasing oInv-bound1*)

6.1 Division

instantiation *ordinal* :: *modulo*
begin

definition

div-ordinal-def:
$x\ div\ y = (if\ 0 < y\ then\ oInv\ ((*)\ y)\ x\ else\ 0)$

definition
mod-ordinal-def:
$x\ mod\ y = ((x::ordinal) - y * (x\ div\ y))$

instance ..

end

lemma *ordinal-divI*: $[\![x = y * q + r;\ r < y]\!] \Longrightarrow x\ div\ y = (q::ordinal)$
using *div-ordinal-def normal.oInv-equality normal-times* **by** *auto*

lemma *ordinal-times-div-le*: $y * (x\ div\ y) \leq (x::ordinal)$
by (*simp add*: *div-ordinal-def normal.oInv-bound1 normal-times*)

lemma *ordinal-less-times-div-plus*: $0 < y \Longrightarrow x < y * (x\ div\ y) + (y::ordinal)$
by (*metis div-ordinal-def normal.oInv-bound2 normal-times ordinal-times-oSuc*)

lemma *ordinal-modI*: $[\![x = y * q + r;\ r < y]\!] \Longrightarrow x\ mod\ y = (r::ordinal)$
by (*simp add*: *mod-ordinal-def ordinal-divI*)

lemma *ordinal-mod-less*: $0 < y \Longrightarrow x\ mod\ y < (y::ordinal)$
by (*simp add*: *mod-ordinal-def ordinal-less-times-div-plus ordinal-times-div-le*)

lemma *ordinal-div-plus-mod*: $y * (x\ div\ y) + (x\ mod\ y) = (x::ordinal)$
by (*simp add*: *mod-ordinal-def ordinal-times-div-le*)

lemma *ordinal-div-less*: $x < y * z \Longrightarrow x\ div\ y < (z::ordinal)$
using *div-ordinal-def normal.oInv-less normal-times* **by** *auto*

lemma *ordinal-le-div*: $[\![0 < y;\ y * z \leq x]\!] \Longrightarrow (z::ordinal) \leq x\ div\ y$
by (*simp add*: *div-ordinal-def normal.le-oInv normal-times*)

lemma *ordinal-mono-div*: *mono* $(\lambda x.\ x\ div\ y::ordinal)$
by (*smt* (*verit*) *Orderings.order-eq-iff div-ordinal-def monoD monoI normal.mono-oInv normal-times*)

lemma *ordinal-div-monoL*: $x \leq x' \Longrightarrow x\ div\ y \leq x'\ div\ (y::ordinal)$
by (*erule monoD*[*OF ordinal-mono-div*])

lemma *ordinal-div-decreasing*: $(x::ordinal)\ div\ y \leq x$
by (*simp add*: *div-ordinal-def normal.oInv-decreasing normal-times*)

lemma *ordinal-div-0*: $x\ div\ 0 = (0::ordinal)$
by (*simp add*: *div-ordinal-def*)

lemma *ordinal-mod-0*: $x\ mod\ 0 = (x::ordinal)$

by (*simp add*: *mod-ordinal-def*)

6.2 Derived properties of division

lemma *ordinal-div-1* [*simp*]: *x div oSuc 0 = x*
using *ordinal-divI* **by** *force*

lemma *ordinal-mod-1* [*simp*]: *x mod oSuc 0 = 0*
by (*simp add*: *mod-ordinal-def*)

lemma *ordinal-div-self* [*simp*]: *0 < x* $\Longrightarrow$ *x div x =* (*1*::*ordinal*)
by (*metis ordinal-divI ordinal-one-def ordinal-plus-0 ordinal-times-1*)

lemma *ordinal-mod-self* [*simp*]: *x mod x =* (*0*::*ordinal*)
by (*metis ordinal-modI ordinal-mod-0 ordinal-neq-0 ordinal-plus-0 ordinal-times-1*)

lemma *ordinal-div-greater* [*simp*]: *x < y* $\Longrightarrow$ *x div y =* (*0*::*ordinal*)
by (*simp add*: *ordinal-divI*)

lemma *ordinal-mod-greater* [*simp*]: *x < y* $\Longrightarrow$ *x mod y =* (*x*::*ordinal*)
by (*simp add*: *mod-ordinal-def*)

lemma *ordinal-0-div* [*simp*]: *0 div x =* (*0*::*ordinal*)
by (*metis div-ordinal-def ordinal-div-greater*)

lemma *ordinal-0-mod* [*simp*]: *0 mod x =* (*0*::*ordinal*)
by (*simp add*: *mod-ordinal-def*)

lemma *ordinal-1-dvd* [*simp*]: *oSuc 0 dvd x*
by (*simp add*: *dvdI*)

lemma *ordinal-dvd-mod*: *y dvd x =* (*x mod y =* (*0*::*ordinal*))
by (*metis dvd-def ordinal-0-times ordinal-div-plus-mod ordinal-modI ordinal-mod-0 ordinal-neq-0 ordinal-plus-0*)

lemma *ordinal-dvd-times-div*: *y dvd x* $\Longrightarrow$ *y* $*$ (*x div y*) *=* (*x*::*ordinal*)
by (*metis ordinal-div-plus-mod ordinal-dvd-mod ordinal-plus-0*)

lemma *ordinal-dvd-oLimit*:
assumes $\forall$ *n. x dvd f n* **shows** *x dvd oLimit f*
proof
show *oLimit f = x* $*$ *oLimit* (λ*n. f n div x*)
using *assms* **by** (*simp add*: *ordinal-dvd-times-div*)
qed

6.3 Logarithms

definition
oLog :: *ordinal* $\Rightarrow$ *ordinal* $\Rightarrow$ *ordinal* **where**
oLog b = (λ*x. if 1 < b then oInv* (($**$) *b*) *x else 0*)

lemma *ordinal-oLogI*:
  assumes $b ** y \le x$ $x < b ** y * b$ **shows** $oLog\ b\ x = y$
proof (*cases* $1 < b$)
  case *True*
  then show *?thesis*
    by (*simp add*: *assms normal.oInv-equality normal-exp oLog-def*)
qed (*use assms linorder-neq-iff* **in** *fastforce*)

lemma *ordinal-exp-oLog-le*: $[\![0 < x;\ oSuc\ 0 < b]\!] \Longrightarrow b ** (oLog\ b\ x) \le x$
  by (*simp add*: *normal.oInv-bound1 normal-exp oLog-def oSuc-leI*)

lemma *ordinal-less-exp-oLog*: $oSuc\ 0 < b \Longrightarrow x < b ** (oLog\ b\ x) * b$
 by (*metis normal.oInv-bound2 normal-exp oLog-def ordinal-exp-oSuc ordinal-one-def*)

lemma *ordinal-oLog-less*: $[\![0 < x;\ oSuc\ 0 < b;\ x < b ** y]\!] \Longrightarrow oLog\ b\ x < y$
  by (*simp add*: *normal.oInv-less normal-exp oLog-def oSuc-leI*)

lemma *ordinal-le-oLog*:
  $[\![oSuc\ 0 < b;\ b ** y \le x]\!] \Longrightarrow y \le oLog\ b\ x$
  by (*simp add*: *oLog-def normal.le-oInv*[*OF normal-exp*])

lemma *ordinal-oLogI2*:
  assumes $oSuc\ 0 < b$ $x = b ** y * q + r$ $0 < q$ $q < b$ $r < b ** y$
  shows $oLog\ b\ x = y$
proof (*rule ordinal-oLogI*)
  show $b ** y \le x$
    using *assms* **by** (*metis dual-order.trans ordinal-le-plusR ordinal-le-timesR*)
  show $x < b ** y * b$
    using *assms*
    by (*metis leD leI order-less-trans ordinal-divI ordinal-exp-not-0 ordinal-le-div*)
qed

lemma *ordinal-div-exp-oLog-less*: $oSuc\ 0 < b \Longrightarrow x\ div\ (b ** oLog\ b\ x) < b$
  by (*simp add*: *ordinal-div-less ordinal-less-exp-oLog*)

lemma *ordinal-oLog-base-0*: $oLog\ 0\ x = 0$
  by (*simp add*: *oLog-def*)

lemma *ordinal-oLog-base-1*: $oLog\ (oSuc\ 0)\ x = 0$
  by (*simp add*: *oLog-def*)

lemma *ordinal-oLog-0*: $oLog\ b\ 0 = 0$
  by (*simp add*: *oLog-def normal.oInv-eq-0*[*OF normal-exp*])

lemma *ordinal-oLog-exp*: $oSuc\ 0 < b \Longrightarrow oLog\ b\ (b ** x) = x$
  by (*simp add*: *oLog-def normal.oInv-inverse*[*OF normal-exp*])

lemma *ordinal-oLog-self*: $oSuc\ 0 < b \Longrightarrow oLog\ b\ b = oSuc\ 0$

by (*metis ordinal-exp-1 ordinal-oLog-exp*)

lemma *ordinal-mono-oLog*: *mono* (*oLog b*)
by (*simp add*: *monoD monoI normal.mono-oInv normal-exp oLog-def*)

lemma *ordinal-oLog-monoR*: $x \leq y \Longrightarrow oLog\ b\ x \leq oLog\ b\ y$
by (*erule monoD*[*OF ordinal-mono-oLog*])

lemma *ordinal-oLog-decreasing*: $oLog\ b\ x \leq x$
by (*metis normal.increasing normal-exp oLog-def ordinal-0-le ordinal-oLog-exp ordinal-oLog-monoR ordinal-one-def*)

end

7 Fixed-points

theory *OrdinalFix*
imports *OrdinalInverse*
begin

primrec *iter* :: $nat \Rightarrow ('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a)$
where
iter 0 $F\ x = x$
| *iter* (*Suc n*) $F\ x = F$ (*iter n F x*)

definition
oFix :: (*ordinal* ⇒ *ordinal*) ⇒ *ordinal* ⇒ *ordinal* **where**
oFix F a = *oLimit* ($\lambda n.$ *iter n F a*)

lemma *oFix-fixed*:
assumes *continuous F* $a \leq F\ a$
shows *F* (*oFix F a*) = *oFix F a*
proof −
have $a \leq$ *oLimit* ($\lambda n.$ *F* (*iter n F a*))
by (*metis OrdinalFix.iter.simps*(*1*) ‹$a \leq F\ a$› *le-oLimitI*)
then have *iter k F a* $\leq$ *oLimit* ($\lambda n.$ *F* (*iter n F a*)) **for** *k*
by (*induction k*) *auto*
then have *oLimit* ($\lambda n.$ *F* (*iter n F a*)) = *oLimit* ($\lambda n.$ *iter n F a*)
by (*metis* (*no-types*, *lifting*) *OrdinalFix.iter.simps*(*2*) *le-oLimit nle-le oLimit-leI*)
then show *?thesis*
by (*simp add*: *assms*(*1*) *continuousD oFix-def*)
qed

lemma *oFix-least*:
assumes *mono F* $F\ x = x$ $a \leq x$ **shows** *oFix F a* $\leq x$
proof −
have *iter n F a* $\leq x$ **for** *n*
proof (*induction n*)
case (*Suc n*)

with *assms monotoneD* **show** *?case* **by** *fastforce*
qed (*use assms* **in** *auto*)
then show *?thesis*
by (*simp add*: *oFix-def oLimit-leI*)
qed

lemma *mono-oFix*:
assumes *mono F* **shows** *mono* (*oFix F*)
proof −
have *iter n F x* $\leq$ *iter n F y* **if** $x \leq y$ **for** *n x y*
using *that assms*
by (*induction n*) (*auto simp*: *monoD*)
then show *?thesis*
by (*metis le-oLimitI monoI oFix-def oLimit-leI*)
qed

lemma *less-oFixD*: $[\![x < oFix\ F\ a;\ mono\ F;\ F\ x = x]\!] \Longrightarrow x < a$
by (*meson linorder-not-le oFix-least*)

lemma *less-oFixI*: $a < F\ a \Longrightarrow a < oFix\ F\ a$
by (*metis OrdinalFix.iter.simps leD le-oLimit oFix-def order-neq-le-trans*)

lemma *le-oFix*: $a \leq oFix\ F\ a$
by (*metis OrdinalFix.iter.simps*(*1*) *le-oLimit oFix-def*)

lemma *le-oFix1*: $F\ a \leq oFix\ F\ a$
by (*metis OrdinalFix.iter.simps le-oLimit oFix-def*)

lemma *less-oFix-0D*:
assumes $x < oFix\ F\ 0$ *mono F* **shows** $x < F\ x$
proof −
have $x < iter\ n\ F\ 0 \Longrightarrow x < F\ x$ **for** *n*
proof (*induction n*)
case *0* **then show** *?case* **by** *auto*
next
case (*Suc n*)
with ‹*mono F*› **show** *?case*
using *monotoneD order.strict-trans2* **by** *fastforce*
qed
then show *?thesis*
using *assms*(*1*) *less-oLimitD oFix-def* **by** *fastforce*
qed

lemma *zero-less-oFix-eq*: $(0 < oFix\ F\ 0) = (0 < F\ 0)$
proof −
have $F\ 0 \leq 0 \Longrightarrow iter\ n\ F\ 0 \leq 0$ **for** *n*
by (*induction n*) *auto*
then show *?thesis*
using *less-oFixI oFix-def* **by** *fastforce*

qed

lemma *oFix-eq-self*:
assumes $F\ a = a$ **shows** $oFix\ F\ a = a$
proof $-$
have $iter\ n\ F\ a = a$ **for** n
by (*induction n*) (*auto simp*: *assms*)
then show *?thesis*
by (*simp add*: *oFix-def*)
qed

7.1 Derivatives of ordinal functions

The derivative of F enumerates all the fixed-points of F

definition
$oDeriv :: (ordinal \Rightarrow ordinal) \Rightarrow ordinal \Rightarrow ordinal$ **where**
$oDeriv\ F = ordinal\text{-}rec\ (oFix\ F\ 0)\ (\lambda p\ x.\ oFix\ F\ (oSuc\ x))$

lemma *oDeriv-0* [*simp*]:
$oDeriv\ F\ 0 = oFix\ F\ 0$
by (*simp add*: *oDeriv-def*)

lemma *oDeriv-oSuc* [*simp*]:
$oDeriv\ F\ (oSuc\ x) = oFix\ F\ (oSuc\ (oDeriv\ F\ x))$
by (*simp add*: *oDeriv-def*)

lemma *oDeriv-oLimit* [*simp*]:
$oDeriv\ F\ (oLimit\ f) = oLimit\ (\lambda n.\ oDeriv\ F\ (f\ n))$
by (*metis dual-order.trans le-oFix less-oSuc oDeriv-def order-le-less ordinal-rec-oLimit*)

lemma *oDeriv-fixed*:
assumes *normal F* **shows** $F\ (oDeriv\ F\ n) = oDeriv\ F\ n$
proof (*induction n rule*: *oLimit-induct*)
case *zero*
then show *?case*
by (*simp add*: *assms normal.continuous oFix-fixed*)
next
case (*suc x*)
then show *?case*
by (*simp add*: *assms normal.continuous normal.increasing oFix-fixed*)
next
case (*lim f*)
then show *?case*
by (*simp add*: *assms continuousD normal.continuous*)
qed

lemma *oDeriv-fixedD*: $[\![oDeriv\ F\ x = x;\ normal\ F]\!] \Longrightarrow F\ x = x$
by (*metis oDeriv-fixed*)

lemma *normal-oDeriv*: *normal* (*oDeriv F*)
  by (*metis le-oFix normal-ordinal-rec oDeriv-def oSuc-le-eq-less*)

lemma *oDeriv-increasing*:
  assumes *continuous F* **shows** $F\ n \le oDeriv\ F\ n$
proof (*induction n rule*: *oLimit-induct*)
  case *zero*
  then show *?case*
    by (*simp add*: *le-oFix1*)
next
  case (*suc x*)
  with *continuous.monoD* [*OF assms*] **show** *?case*
   by (*metis dual-order.trans le-oFix1 normal.increasing normal-oDeriv oDeriv-oSuc oSuc-le-oSuc*)
next
  case (*lim f*)
  then show *?case*
    by (*metis assms continuousD le-oLimitI oDeriv-oLimit oLimit-leI*)
qed

lemma *oDeriv-total*:
  assumes *normal F* $F\ x = x$ **shows** $\exists n.\ x = oDeriv\ F\ n$
proof −
  have $\exists n.\ oDeriv\ F\ n \le x \land x < oDeriv\ F\ (oSuc\ n)$
   by (*metis assms normal.mono normal.oInv-ex normal-oDeriv oDeriv-0 oFix-least ordinal-0-le*)
  then show *?thesis*
   by (*metis assms leD normal.mono oDeriv-oSuc oFix-least oSuc-leI order-neq-le-trans*)
qed

lemma *range-oDeriv*: *normal F* $\Longrightarrow$ *range* (*oDeriv F*) $= \{x.\ F\ x = x\}$
  by (*auto intro*: *oDeriv-fixed dest*: *oDeriv-total*)

end

8 Omega

theory *OrdinalOmega*
imports *OrdinalFix*
begin

8.1 Embedding naturals in the ordinals

primrec *ordinal-of-nat* :: *nat* $\Rightarrow$ *ordinal*
where
  ordinal-of-nat 0 = *0*
| *ordinal-of-nat* (*Suc n*) = *oSuc* (*ordinal-of-nat n*)

lemma *strict-mono-ordinal-of-nat*: *strict-mono ordinal-of-nat*

by (*simp add*: *strict-mono-natI*)

lemma *not-limit-ordinal-nat*: $\neg$ *limit-ordinal* (*ordinal-of-nat* n)
by (*induct* n) *simp-all*

lemma *ordinal-of-nat-eq* [*simp*]:
(*ordinal-of-nat* x = *ordinal-of-nat* y) = (x = y)
by (*rule strict-mono-cancel-eq*[*OF strict-mono-ordinal-of-nat*])

lemma *ordinal-of-nat-less* [*simp*]:
(*ordinal-of-nat* x < *ordinal-of-nat* y) = (x < y)
by (*rule strict-mono-cancel-less*[*OF strict-mono-ordinal-of-nat*])

lemma *ordinal-of-nat-le* [*simp*]:
(*ordinal-of-nat* $x \leq$ *ordinal-of-nat* y) = ($x \leq y$)
by (*rule strict-mono-cancel-le*[*OF strict-mono-ordinal-of-nat*])

lemma *ordinal-of-nat-plus* [*simp*]:
ordinal-of-nat x + *ordinal-of-nat* y = *ordinal-of-nat* ($x + y$)
by (*induct* y) *simp-all*

lemma *ordinal-of-nat-times* [*simp*]:
ordinal-of-nat x $*$ *ordinal-of-nat* y = *ordinal-of-nat* ($x * y$)
by (*induct* y) (*simp-all add*: *add.commute*)

lemma *ordinal-of-nat-exp* [*simp*]:
ordinal-of-nat x $**$ *ordinal-of-nat* y = *ordinal-of-nat* (x ^ y)
by (*induct* y, *cases* x) (*simp-all add*: *mult.commute*)

lemma *oSuc-plus-ordinal-of-nat*:
oSuc x + *ordinal-of-nat* n = *oSuc* (x + *ordinal-of-nat* n)
by (*induct* n) *simp-all*

lemma *less-ordinal-of-nat*:
(x < *ordinal-of-nat* n) = ($\exists\, m$. x = *ordinal-of-nat* $m \wedge m < n$)
by (*induction* n) (*use less-oSuc-eq-le* **in** *force*)+

lemma *le-ordinal-of-nat*:
($x \leq$ *ordinal-of-nat* n) = ($\exists\, m$. x = *ordinal-of-nat* $m \wedge m \leq n$)
by (*auto simp add*: *order-le-less less-ordinal-of-nat*)

8.2 Omega, the least limit ordinal

definition
omega :: *ordinal* (‹ω›) **where**
omega = *oLimit ordinal-of-nat*

lemma *less-omegaD*: $x < \omega \Longrightarrow \exists\, n$. x = *ordinal-of-nat* n
by (*metis less-oLimitD less-ordinal-of-nat omega-def*)

lemma *omega-leI*: $\forall\, n.\ \textit{ordinal-of-nat}\ n \le x \Longrightarrow \omega \le x$
  by (*simp add*: *oLimit-leI omega-def*)

lemma *nat-le-omega* [*simp*]: $\textit{ordinal-of-nat}\ n \le \omega$
  by (*simp add*: *oLimit-leI omega-def*)

lemma *nat-less-omega* [*simp*]: $\textit{ordinal-of-nat}\ n < \omega$
  by (*simp add*: *omega-def strict-mono-less-oLimit strict-mono-ordinal-of-nat*)

lemma *zero-less-omega* [*simp*]: $0 < \omega$
  using *nat-less-omega ordinal-neq-0* **by** *fastforce*

lemma *limit-ordinal-omega*: $\textit{limit-ordinal}\ \omega$
  by (*metis limit-ordinal-oLimitI nat-less-omega omega-def*)

lemma *Least-limit-ordinal*: $(\textit{LEAST}\ x.\ \textit{limit-ordinal}\ x) = \omega$
proof (*rule Least-equality*)
  show $\bigwedge y.\ \textit{limit-ordinal}\ y \Longrightarrow \omega \le y$
    by (*metis leI less-omegaD not-limit-ordinal-nat*)
qed (*rule limit-ordinal-omega*)

lemma $\textit{range}\ f = \textit{range}\ \textit{ordinal-of-nat} \Longrightarrow \textit{oLimit}\ f = \omega$
  by (*metis le-oLimit oLimit-leI omega-def order-antisym rangeE rangeI*)

8.3 Arithmetic properties of ω

lemma *oSuc-less-omega* [*simp*]: $(\textit{oSuc}\ x < \omega) = (x < \omega)$
  by (*rule oSuc-less-limit-ordinal*[*OF limit-ordinal-omega*])

lemma *oSuc-plus-omega* [*simp*]: $\textit{oSuc}\ x + \omega = x + \omega$
proof −
  have $\bigwedge n.\ \exists\, m.\ \textit{oSuc}\ x + \textit{ordinal-of-nat}\ n \le x + \textit{ordinal-of-nat}\ m$
    using *oSuc-le-eq-less oSuc-plus-ordinal-of-nat* **by** *auto*
  moreover
  have $\bigwedge n.\ \exists\, m.\ x + \textit{ordinal-of-nat}\ n \le \textit{oSuc}\ x + \textit{ordinal-of-nat}\ m$
    using *dual-order.order-iff-strict oSuc-plus-ordinal-of-nat* **by** *auto*
  ultimately show *?thesis*
    by (*simp add*: *oLimit-eqI omega-def*)
qed

lemma *ordinal-of-nat-plus-omega* [*simp*]:
  $\textit{ordinal-of-nat}\ n + \omega = \omega$
  by (*induct n*) *simp-all*

lemma *ordinal-of-nat-times-omega* [*simp*]:
  assumes $k > 0$ **shows** $\textit{ordinal-of-nat}\ k * \omega = \omega$
proof −
  have $\exists\, m.\ \textit{ordinal-of-nat}\ n \le \textit{ordinal-of-nat}\ (k * m)$

by (*metis assms le-add1 mult-eq-if not-less-zero ordinal-of-nat-le*)
then have *oLimit* (λn. *ordinal-of-nat* ($k * n$)) = *oLimit ordinal-of-nat*
by (*metis assms oLimit-eqI gr0-conv-Suc le-add1 mult-Suc ordinal-of-nat-le*)
then show *?thesis*
by (*simp add*: *omega-def*)
qed

lemma *ordinal-plus-times-omega*: $x + x * \omega = x * \omega$
by (*metis oSuc-plus-omega ordinal-0-plus ordinal-times-1 ordinal-times-distrib*)

lemma *ordinal-plus-absorb*: $x * \omega \le y \Longrightarrow x + y = y$
by (*metis ordinal-plus-assoc ordinal-plus-minus2 ordinal-plus-times-omega*)

lemma *ordinal-less-plusL*:
assumes $y < x * \omega$ **shows** $y < x + y$
proof (*cases* $x = 0$)
case *True*
with *assms* **show** *?thesis* **by** *auto*
next
case *False*
then obtain n **where** n: *ordinal-of-nat* $n = y$ *div* x
using *assms less-omegaD ordinal-div-less* **by** *metis*
then have $y < x * (1 +$ *ordinal-of-nat* $n)$
using n **unfolding** *ordinal-one-def oSuc-plus-ordinal-of-nat*
by (*metis False ordinal-0-plus ordinal-less-times-div-plus ordinal-neq-0 ordinal-times-oSuc*)
also have ... $\le x + y$
using n **by** (*simp add*: *ordinal-times-distrib ordinal-times-div-le*)
finally show *?thesis* **.**
qed

lemma *ordinal-plus-absorb-iff*: $(x + y = y) = (x * \omega \le y)$
by (*metis linorder-linear order-le-less order-less-irrefl ordinal-less-plusL ordinal-plus-absorb*)

lemma *ordinal-less-plusL-iff*: $(y < x + y) = (y < x * \omega)$
by (*metis leI linorder-neq-iff ordinal-less-plusL ordinal-plus-absorb*)

8.4 Additive principal ordinals

locale *additive-principal* =
fixes a :: *ordinal*
assumes *not-0*: $0 < a$
assumes *sum-eq*: $\bigwedge b.\ b < a \Longrightarrow b + a = a$

lemma (**in** *additive-principal*) *sum-less*:
$[\![x < a;\ y < a]\!] \Longrightarrow x + y < a$
by (*metis ordinal-plus-strict-monoR sum-eq*)

lemma (**in** *additive-principal*) *times-nat-less*:
  $x < a \Longrightarrow x * \textit{ordinal-of-nat } n < a$
  by (*induct n*) (*auto simp*: *not-0 sum-less*)

lemma *not-additive-principal-0*: ¬ *additive-principal 0*
  by (*simp add*: *additive-principal-def*)

lemma *additive-principal-oSuc*:
  additive-principal (*oSuc a*) = (*a* = *0*)
  unfolding *additive-principal-def*
  by (*metis less-oSuc0 ordinal-plus-0 ordinal-plus-left-cancel-less ordinal-plus-oSuc*)

lemma *additive-principal-intro2* [*rule-format*]:
  assumes *not-0*: $0 < a$ **and** *lessa*: $(\forall x<a.\ \forall y<a.\ x + y < a)$
  shows *additive-principal a*
proof −
  have $\forall b<a.\ b + a = a$
    using *lessa*
  proof (*induction a rule*: *oLimit-induct*)
    case *zero*
    then show *?case* **by** *auto*
  next
    case (*suc x*)
    then show *?case*
     by (*metis le-oSucE less-oSuc linorder-not-le ordinal-le-plusL ordinal-plus-oSuc*)
  next
    case (*lim f*)
    then show *?case*
      by (*metis leD order-le-less ordinal-le-plusL ordinal-plus-minus2*)
  qed
  then show *?thesis*
    by (*simp add*: *additive-principal-def not-0*)
qed

lemma *additive-principal-1*: *additive-principal* (*oSuc 0*)
  by (*simp add*: *additive-principal-def*)

lemma *additive-principal-omega*: *additive-principal* ω
 using *additive-principal.intro less-omegaD ordinal-of-nat-plus-omega zero-less-omega*
by *blast*

lemma *additive-principal-times-omega*:
  assumes $0 < x$ **shows** *additive-principal* $(x * \omega)$
proof (*rule additive-principal.intro*)
  fix *b*
  assume $b < x * \omega$
  then obtain *k* **where** *k*: $b < x * \textit{ordinal-of-nat } k$
    by (*metis less-oLimitD omega-def ordinal-times-oLimit*)
  then have $b + x * \textit{ordinal-of-nat } n \le x * \textit{ordinal-of-nat } (k + n)$ **for** *n*

by (*metis order-less-imp-le ordinal-of-nat-plus ordinal-plus-monoL ordinal-times-distrib*)
then show $b + x * \omega = x * \omega$
by (*metis oLimit-eqI omega-def ordinal-le-plusL ordinal-plus-oLimit ordinal-times-oLimit*)
qed (*use assms* **in** *auto*)

lemma *additive-principal-oLimit*:
assumes $\forall n.$ *additive-principal* $(f\ n)$
shows *additive-principal* (*oLimit* f)
proof (*rule additive-principal.intro*)
show $0 <$ *oLimit* f
by (*metis assms less-oLimitI not-additive-principal-0 ordinal-neq-0*)
next
fix b
assume $b <$ *oLimit* f
then obtain k **where** $b < f\ k$
using *less-oLimitD* **by** *auto*
then have $\exists m.\ b + f\ n \le f\ m$ **for** n
by (*metis additive-principal.sum-eq assms order.trans leI order-less-imp-le ordinal-plus-left-cancel-le*)
then show $b +$ *oLimit* $f =$ *oLimit* f
by (*metis* ‹$b < f\ k$› *additive-principal-def assms le-oLimit ordinal-plus-assoc ordinal-plus-minus2*)
qed

lemma *additive-principal-omega-exp*: *additive-principal* $(\omega ** x)$
by (*induction x rule*: *oLimit-induct*)
(*auto simp*: *additive-principal-1 additive-principal-times-omega additive-principal-oLimit*)

lemma (**in** *additive-principal*) *omega-exp*: $\exists x.\ a = \omega ** x$
proof −
have $\exists x.\ \omega ** x \le a \wedge a < \omega ** (oSuc\ x)$
by (*metis not-0 oSuc-less-omega ordinal-exp-oLog-le ordinal-exp-oSuc ordinal-less-exp-oLog zero-less-omega*)
then show *?thesis*
by (*metis leD order-le-imp-less-or-eq ordinal-exp-oSuc ordinal-plus-absorb-iff sum-eq*)
qed

lemma *additive-principal-iff*:
additive-principal $a = (\exists x.\ a = \omega ** x)$
using *additive-principal.omega-exp additive-principal-omega-exp* **by** *blast*

lemma *absorb-omega-exp*:
$x < \omega ** a \Longrightarrow x + \omega ** a = \omega ** a$
by (*rule additive-principal.sum-eq*[*OF additive-principal-omega-exp*])

lemma *absorb-omega-exp2*: $a < b \Longrightarrow \omega ** a + \omega ** b = \omega ** b$
by (*rule absorb-omega-exp*, *simp add*: *ordinal-exp-strict-monoR*)

8.5 Cantor normal form

lemma *cnf-lemma*: $x > 0 \Longrightarrow x - \omega ** oLog\ \omega\ x < x$
by (*simp add*: *ordinal-exp-oLog-le ordinal-less-exp-oLog ordinal-less-plusL*)

primrec *from-cnf* **where**
from-cnf [] = *0*
| *from-cnf* (*x* # *xs*) = ω ** *x* + *from-cnf xs*

function *to-cnf* **where**
[*simp del*]: *to-cnf x* = (*if x* = *0 then* [] *else*
oLog ω *x* # *to-cnf* (*x* − ω ** *oLog* ω *x*))
by *pat-completeness auto*

termination by (*relation* {(*x*, *y*). *x* < *y*})
(*simp-all add*: *wf cnf-lemma*)

lemma *to-cnf-0* [*simp*]: *to-cnf 0* = []
by (*simp add*: *to-cnf.simps*)

lemma *to-cnf-not-0*:
0 < *x* $\Longrightarrow$ *to-cnf x* = *oLog* ω *x* # *to-cnf* (*x* − ω ** *oLog* ω *x*)
by (*simp add*: *to-cnf.simps*[*of x*])

lemma *to-cnf-eq-Cons*: *to-cnf x* = *a* # *list* $\Longrightarrow$ *a* = *oLog* ω *x*
by (*case-tac x* = *0*, *simp*, *simp add*: *to-cnf-not-0*)

lemma *to-cnf-inverse*: *from-cnf* (*to-cnf x*) = *x*
using *wf*
proof (*induction rule*: *wf-induct-rule*)
case (*less x*)
then have *IH*: ∀ *y*<*x*. *from-cnf* (*to-cnf y*) = *y*
by *simp*
show *?case*
proof (*cases x* = *0*)
case *False*
with *cnf-lemma* **show** *?thesis*
by (*simp add*: *ordinal-exp-oLog-le to-cnf-not-0 IH*)
qed *auto*
qed

primrec *normalize-cnf* **where**
normalize-cnf-Nil: *normalize-cnf* [] = []
| *normalize-cnf-Cons*: *normalize-cnf* (*x* # *xs*) =
(*case xs of* [] ⇒ [*x*] | *y* # *ys* ⇒
(*if x* < *y then* [] *else* [*x*]) @ *normalize-cnf xs*)

lemma *from-cnf-normalize-cnf*: *from-cnf* (*normalize-cnf xs*) = *from-cnf xs*
proof (*induction xs*)
case *Nil*

then show *?case* **by** *auto*
next
case (*Cons a xs*)
have $\bigwedge x\ y.\ a < x \Longrightarrow \omega ** x + \textit{from-cnf}\ y = \omega ** a + (\omega ** x + \textit{from-cnf}\ y)$
by (*metis absorb-omega-exp2 from-cnf.simps*(*2*) *ordinal-plus-assoc*)
with *Cons* **show** *?case*
by *simp* (*auto simp del*: *normalize-cnf-Cons split*: *list.split*)
qed

lemma *normalize-cnf-to-cnf*: *normalize-cnf* (*to-cnf x*) = *to-cnf x*
using *wf*
proof (*induction rule*: *wf-induct-rule*)
case (*less x*)
then have *IH*: $\forall y<x.$ *normalize-cnf* (*to-cnf y*) = *to-cnf y*
by *simp*
show *?case*
proof (*cases x* = *0*)
case *False*
then have §: *normalize-cnf* (*to-cnf* ($x - \omega ** \textit{oLog}\ \omega\ x$)) = *to-cnf* ($x - \omega **$
oLog ω *x*)
using *IH cnf-lemma* **by** *blast*
with *False* **show** *?thesis*
apply (*simp add*: *to-cnf-not-0*)
apply (*case-tac to-cnf* ($x - \omega ** \textit{oLog}\ \omega\ x$), *simp-all*)
by (*metis cnf-lemma linorder-not-le order-le-less ordinal-oLog-monoR to-cnf-eq-Cons*)
qed *auto*
qed

alternate form of CNF

lemma *cnf2-lemma*:
$0 < x \Longrightarrow x \bmod \omega ** \textit{oLog}\ \omega\ x < x$
by (*meson oSuc-less-omega order-less-le-trans ordinal-exp-not-0 ordinal-exp-oLog-le ordinal-mod-less zero-less-omega*)

primrec *from-cnf2* **where**
from-cnf2 [] = *0*
| *from-cnf2* (*x* # *xs*) = ω ** *fst x* * *ordinal-of-nat* (*snd x*) + *from-cnf2 xs*

function *to-cnf2* **where**
[*simp del*]: *to-cnf2 x* = (*if x* = *0 then* [] *else*
(*oLog* ω *x*, *inv ordinal-of-nat* (*x div* (ω ** *oLog* ω *x*)))
\# *to-cnf2* (*x mod* (ω ** *oLog* ω *x*)))
by *pat-completeness auto*

termination by (*relation* {(*x*,*y*). *x* < *y*})
(*simp-all add*: *wf cnf2-lemma*)

```
lemma to-cnf2-0 [simp]: to-cnf2 0 = []
  by (simp add: to-cnf2.simps)

lemma to-cnf2-not-0:
  0 < x ⟹ to-cnf2 x = (oLog ω x, inv ordinal-of-nat (x div (ω ** oLog ω x)))
                    # to-cnf2 (x mod (ω ** oLog ω x))
  by (simp add: to-cnf2.simps[of x])

lemma to-cnf2-eq-Cons: to-cnf2 x = (a,b) # list ⟹ a = oLog ω x
  by (metis Pair-inject list.discI list.inject to-cnf2.elims)

lemma ordinal-of-nat-of-ordinal:
  x < ω ⟹ ordinal-of-nat (inv ordinal-of-nat x) = x
  by (simp add: f-inv-into-f image-def less-omegaD)

lemma to-cnf2-inverse: from-cnf2 (to-cnf2 x) = x
  using wf
proof (induction x rule: wf-induct-rule)
  case (less x)
  show ?case
  proof (cases x > 0)
    case True
    then show ?thesis
      by (simp add: cnf2-lemma less ordinal-div-exp-oLog-less ordinal-div-plus-mod
ordinal-of-nat-of-ordinal to-cnf2-not-0)
  qed auto
qed

primrec is-normalized2 where
  is-normalized2-Nil: is-normalized2 [] = True
| is-normalized2-Cons: is-normalized2 (x # xs) =
     (case xs of [] ⇒ True | y # ys ⇒ fst y < fst x ∧ is-normalized2 xs)

lemma is-normalized2-to-cnf2: is-normalized2 (to-cnf2 x)
  using wf
proof (induction x rule: wf-induct-rule)
  case (less x)
  show ?case
  proof (cases x > 0)
    case True
    then have *: is-normalized2 (to-cnf2 (x mod ω ** oLog ω x))
      using cnf2-lemma less by blast
    show ?thesis
    proof (cases x mod ω ** oLog ω x = 0)
      case True
      with to-cnf2-not-0 ‹x > 0› show ?thesis by simp
    next
      case False
      with ‹x > 0› * show ?thesis
```

by (*simp add*: *ordinal-mod-less ordinal-oLog-less to-cnf2-not-0*)
qed
qed *auto*
qed

8.6 Epsilon 0

definition *epsilon0* :: *ordinal* (‹ε_0›) **where**
epsilon0 = *oFix* ((**) ω) *0*

lemma *less-omega-exp*: $x < \varepsilon_0 \Longrightarrow x < \omega ** x$
by (*simp add*: *epsilon0-def less-oFix-0D normal.mono normal-exp*)

lemma *omega-exp-epsilon0*: $\omega ** \varepsilon_0 = \varepsilon_0$
by (*simp add*: *continuous-exp epsilon0-def oFix-fixed*)

lemma *oLog-omega-less*: $[\![0 < x;\ x < \varepsilon_0]\!] \Longrightarrow oLog\ \omega\ x < x$
by (*simp add*: *less-omega-exp ordinal-oLog-less*)

end

9 Veblen Hierarchies

theory *OrdinalVeblen*
imports *OrdinalOmega*
begin

9.1 Closed, unbounded sets

locale *normal-set* =
fixes *A* :: *ordinal set*
assumes *closed*: $\bigwedge g.\ \forall n.\ g\ n \in A \Longrightarrow oLimit\ g \in A$
and *unbounded*: $\bigwedge x.\ \exists y{\in}A.\ x < y$

lemma (**in** *normal-set*) *less-next*: $x < (LEAST\ z.\ z \in A \wedge x < z)$
by (*metis* (*no-types*, *lifting*) *LeastI unbounded*)

lemma (**in** *normal-set*) *mem-next*: $(LEAST\ z.\ z \in A \wedge x < z) \in A$
by (*metis* (*no-types*, *lifting*) *LeastI unbounded*)

lemma (**in** *normal*) *normal-set-range*: *normal-set* (*range F*)
proof (*rule normal-set.intro*)
fix *g* :: $nat \Rightarrow ordinal$
assume $\forall n.\ g\ n \in range\ F$
then have $\bigwedge n.\ g\ n = F\ (LEAST\ z.\ g\ n = F\ z)$
by (*meson LeastI rangeE*)
then have $oLimit\ g = F\ (oLimit\ (\lambda n.\ LEAST\ z.\ g\ n = F\ z))$
by (*simp add*: *continuousD continuous-axioms*)
then show $oLimit\ g \in range\ F$

by *simp*
next
show $\bigwedge x.\ \exists\, y{\in}range\ F.\ x < y$
using *oInv-bound2* **by** *blast*
qed

lemma *oLimit-mem-INTER*:
assumes *norm*: $\forall\, n.\ normal\text{-}set\ (A\ n)$
and *A*: $\forall\, n.\ A\ (Suc\ n) \subseteq A\ n$ $\forall\, n.\ f\ n \in A\ n$ **and** *mono f*
shows *oLimit f* $\in (\bigcap n.\ A\ n)$
proof
fix *k*
have $f\ (n + k) \in A\ k$ **for** *n*
using *A le-add2 lift-Suc-antimono-le* **by** *blast*
then have *oLimit* $(\lambda n.\ f\ (n + k)) \in A\ k$
by (*simp add*: *norm normal-set.closed*)
then show *oLimit f* $\in A\ k$
by (*simp add*: ‹*mono f*› *oLimit-shift-mono*)
qed

lemma *normal-set-INTER*:
assumes *norm*: $\forall\, n.\ normal\text{-}set\ (A\ n)$ **and** *A*: $\forall\, n.\ A\ (Suc\ n) \subseteq A\ n$
shows *normal-set* $(\bigcap n.\ A\ n)$
proof (*rule normal-set.intro*)
fix $g :: nat \Rightarrow ordinal$
assume $\forall\, n.\ g\ n \in \bigcap\ (range\ A)$
then show *oLimit* $g \in \bigcap\ (range\ A)$
using *norm normal-set.closed* **by** *force*
next
fix *x*
define *F* **where** $F \equiv \lambda n.\ LEAST\ y.\ y \in A\ n \wedge x < y$
have $x <$ *oLimit F*
by (*simp add*: *F-def less-oLimitI norm normal-set.less-next*)
moreover
have §: $F\ n \in A\ n$ **for** *n*
by (*simp add*: *F-def norm normal-set.mem-next*)
then have $F\ n \le F\ (Suc\ n)$ **for** *n*
unfolding *F-def*
by (*metis* (*no-types*, *lifting*) *A LeastI Least-le norm normal-set-def subsetD*)
then have *oLimit* $F \in \bigcap\ (range\ A)$
by (*meson* § *A mono-natI norm oLimit-mem-INTER*)
ultimately show $\exists\, y{\in}\bigcap\ (range\ A).\ x < y$
by *blast*
qed

9.2 Ordering functions

There is a one-to-one correspondence between closed, unbounded sets of ordinals and normal functions on ordinals.

definition
ordering :: (*ordinal set*) ⇒ (*ordinal* ⇒ *ordinal*) **where**
ordering A = *ordinal-rec* (*LEAST z. z* ∈ *A*) (λ*p x. LEAST z. z* ∈ *A* ∧ *x* < *z*)

lemma *ordering-0*:
ordering A 0 = (*LEAST z. z* ∈ *A*)
by (*simp add*: *ordering-def*)

lemma *ordering-oSuc*:
ordering A (*oSuc x*) = (*LEAST z. z* ∈ *A* ∧ *ordering A x* < *z*)
by (*simp add*: *ordering-def*)

lemma (**in** *normal-set*) *normal-ordering*: *normal* (*ordering A*)
by (*simp add*: *OrdinalVeblen.ordering-def normal-ordinal-rec normal-set.less-next normal-set-axioms*)

lemma (**in** *normal-set*) *ordering-oLimit*: *ordering A* (*oLimit f*) = *oLimit* (λ*n. ordering A* (*f n*))
by (*simp add*: *normal.oLimit normal-ordering*)

lemma (**in** *normal*) *ordering-range*: *ordering* (*range F*) = *F*
proof
fix *x*
show *ordering* (*range F*) *x* = *F x*
proof (*induction x rule*: *oLimit-induct*)
case *zero*
have (*LEAST z. z* ∈ *range F*) = *F 0*
by (*metis Least-equality Least-mono UNIV-I mono ordinal-0-le*)
then show *?case*
by (*simp add*: *ordering-0*)
next
case (*suc x*)
have *ordering* (*range F*) (*oSuc x*) = (*LEAST z. z* ∈ *range F* ∧ *F x* < *z*)
by (*simp add*: *ordering-oSuc suc*)
also have ... = *F* (*oSuc x*)
using *cancel-less less-oInvD oInv-inverse*
by (*bestsimp intro*!: *Least-equality local.strict-monoD*)
finally show *?case* **.**
next
case (*lim f*)
then show *?case*
using *oLimit* **by** (*simp add*: *normal-set-range normal-set.ordering-oLimit*)
qed
qed

lemma (**in** *normal-set*) *ordering-mem*: *ordering A x* ∈ *A*
proof (*induction x rule*: *oLimit-induct*)
case *zero*
then show *?case*

by (*metis LeastI ordering-0 unbounded*)
next
case (*suc x*)
then show *?case*
by (*simp add*: *mem-next ordering-oSuc*)
next
case (*lim f*)
then show *?case*
by (*simp add*: *closed normal.oLimit normal-ordering*)
qed

lemma (**in** *normal-set*) *range-ordering*: *range* (*ordering A*) = *A*
proof −
have $\forall y.\ y \in A \longrightarrow y < ordering\ A\ x \longrightarrow y \in range\ (ordering\ A)$ **for** *x*
proof (*induction x rule*: *oLimit-induct*)
case *zero*
then show *?case*
using *not-less-Least ordering-0* **by** *auto*
next
case (*suc x*)
then show *?case*
using *not-less-Least ordering-oSuc* **by** *fastforce*
next
case (*lim f*)
then show *?case*
by (*metis less-oLimitD ordering-oLimit*)
qed
then show *?thesis*
using *normal.oInv-bound2 normal-ordering ordering-mem* **by** *fastforce*
qed

lemma *ordering-INTER-0*:
assumes *norm*: $\forall n.\ normal\text{-}set\ (A\ n)$ **and** *A*: $\forall n.\ A\ (Suc\ n) \subseteq A\ n$
shows $ordering\ (\bigcap n.\ A\ n)\ 0 = oLimit\ (\lambda n.\ ordering\ (A\ n)\ 0)$
proof −
have $oLimit\ (\lambda n.\ OrdinalVeblen.ordering\ (A\ n)\ 0) \in \bigcap\ (range\ A)$
using *assms*
by (*metis* (*mono-tags*, *lifting*) *Least-le mono-natI normal-set.ordering-mem oLimit-mem-INTER ordering-0 subsetD*)
moreover have $\bigwedge y.\ y \in \bigcap\ (range\ A) \Longrightarrow oLimit\ (\lambda n.\ ordering\ (A\ n)\ 0) \le y$
by (*simp add*: *Least-le oLimit-def ordering-0*)
ultimately show *?thesis*
by (*metis LeastI Least-le nle-le ordering-0*)
qed

9.3 Critical ordinals

definition
critical-set :: *ordinal set* ⇒ *ordinal* ⇒ *ordinal set* **where**

critical-set A =
  ordinal-rec0 A (λ*p x*. *x* ∩ *range* (*oDeriv* (*ordering x*))) (λ*f*. ⋂ *n*. *f n*)

lemma *critical-set-0* [*simp*]: *critical-set A 0* = *A*
by (*simp add*: *critical-set-def*)

lemma *critical-set-oSuc-lemma*:
critical-set A (*oSuc n*) = *critical-set A n* ∩ *range* (*oDeriv* (*ordering* (*critical-set A n*)))
by (*simp add*: *critical-set-def ordinal-rec0-oSuc*)

lemma *omega-complete-INTER*: *omega-complete* (λ*x y*. *y* ⊆ *x*) (λ*f*. ⋂ (*range f*))
by (*simp add*: *INF-greatest Inf-lower omega-complete-axioms-def omega-complete-def porder.flip porder-order*)

lemma *critical-set-oLimit*: *critical-set A* (*oLimit f*) = (⋂ *n*. *critical-set A* (*f n*))
unfolding *critical-set-def*
by (*best intro*!: *omega-complete.ordinal-rec0-oLimit omega-complete-INTER*)

lemma *critical-set-mono*: *x* ≤ *y* ⟹ *critical-set A y* ⊆ *critical-set A x*
unfolding *critical-set-def*
by (*intro omega-complete.ordinal-rec0-mono* [*OF omega-complete-INTER*]) *force*

lemma (**in** *normal-set*) *range-oDeriv-subset*: *range* (*oDeriv* (*ordering A*)) ⊆ *A*
by (*metis image-subsetI normal-ordering oDeriv-fixed rangeI range-ordering*)

lemma *normal-set-critical-set*: *normal-set A* ⟹ *normal-set* (*critical-set A x*)
proof (*induction x rule*: *oLimit-induct*)
case *zero*
then show *?case*
by *simp*
next
case (*suc x*)
then show *?case*
by (*simp add*: *Int-absorb1 critical-set-oSuc-lemma normal.normal-set-range normal-oDeriv normal-set.range-oDeriv-subset*)
next
case (*lim f*)
then show *?case*
unfolding *critical-set-oLimit*
by (*meson critical-set-mono lessI normal-set-INTER order-le-less strict-mono.strict-mono*)
qed

lemma *critical-set-oSuc*:
normal-set A ⟹ *critical-set A* (*oSuc x*) = *range* (*oDeriv* (*ordering* (*critical-set A x*)))
by (*metis critical-set-oSuc-lemma inf.absorb-iff2 normal-set.range-oDeriv-subset normal-set-critical-set*)

9.4 Veblen hierarchy over a normal function

definition
oVeblen :: (*ordinal* ⇒ *ordinal*) ⇒ *ordinal* ⇒ *ordinal* ⇒ *ordinal* **where**
oVeblen F = (λ*x*. *ordering* (*critical-set* (*range F*) *x*))

lemma (**in** *normal*) *oVeblen-0*: *oVeblen F 0* = *F*
by (*simp add*: *normal.ordering-range normal-axioms oVeblen-def*)

lemma (**in** *normal*) *oVeblen-oSuc*: *oVeblen F* (*oSuc x*) = *oDeriv* (*oVeblen F x*)
using *critical-set-oSuc normal.normal-set-range normal.ordering-range normal-axioms normal-oDeriv oVeblen-def* **by** *presburger*

lemma (**in** *normal*) *oVeblen-oLimit*:
oVeblen F (*oLimit f*) = *ordering* (⋂*n*. *range* (*oVeblen F* (*f n*)))
unfolding *oVeblen-def*
using *critical-set-oLimit normal-set.range-ordering normal-set-critical-set normal-set-range* **by** *presburger*

lemma (**in** *normal*) *normal-oVeblen*: *normal* (*oVeblen F x*)
unfolding *oVeblen-def*
by (*simp add*: *normal-set.normal-ordering normal-set-critical-set normal-set-range*)

lemma (**in** *normal*) *continuous-oVeblen-0*: *continuous* (λ*x*. *oVeblen F x 0*)
proof (*rule continuousI*)
fix *f*:: *nat* ⇒ *ordinal*
assume *f*: *OrdinalInduct.strict-mono f*
have *normal-set* (*critical-set* (*range F*) (*f n*)) **for** *n*
using *normal-set-critical-set normal-set-range* **by** *blast*
moreover
have *critical-set* (*range F*) (*f* (*Suc n*)) ⊆ *critical-set* (*range F*) (*f n*) **for** *n*
by (*simp add*: *f critical-set-mono strict-mono-monoD*)
ultimately show *oVeblen F* (*oLimit f*) *0* = *oLimit* (λ*n*. *oVeblen F* (*f n*) *0*)
using *ordering-INTER-0* **by** (*simp add*: *oVeblen-def critical-set-oLimit*)
next
show ⋀*x*. *oVeblen F x 0* ≤ *oVeblen F* (*oSuc x*) *0*
by (*simp add*: *le-oFix1 oVeblen-oSuc*)
qed

lemma (**in** *normal*) *oVeblen-oLimit-0*:
oVeblen F (*oLimit f*) *0* = *oLimit* (λ*n*. *oVeblen F* (*f n*) *0*)
by (*rule continuousD*[*OF continuous-oVeblen-0*])

lemma (**in** *normal*) *normal-oVeblen-0*:
assumes *0* < *F 0* **shows** *normal* (λ*x*. *oVeblen F x 0*)
proof −
{ **fix** *x*
have *0* < *oVeblen F x 0*
by (*metis leD ordinal-0-le ordinal-neq-0 continuous.monoD continuous-oVeblen-0 oVeblen-0 assms*)

then have *oVeblen F x 0* < *oVeblen F x* (*oDeriv* (*oVeblen F x*) *0*)
by (*simp add*: *normal.strict-monoD normal-oVeblen zero-less-oFix-eq*)
then have *oVeblen F x 0* < *oVeblen F* (*oSuc x*) *0*
by (*metis normal-oVeblen oDeriv-fixed oVeblen-oSuc*)
}
then show *?thesis*
using *continuous-def continuous-oVeblen-0 normalI* **by** *blast*
qed

lemma (**in** *normal*) *range-oVeblen*:
range (*oVeblen F x*) = *critical-set* (*range F*) *x*
unfolding *oVeblen-def*
using *normal-set.range-ordering normal-set-critical-set normal-set-range* **by** *blast*

lemma (**in** *normal*) *range-oVeblen-subset*:
$x \leq y \Longrightarrow$ *range* (*oVeblen F y*) ⊆ *range* (*oVeblen F x*)
using *critical-set-mono range-oVeblen* **by** *presburger*

lemma (**in** *normal*) *oVeblen-fixed*:
assumes *x*<*y*
shows *oVeblen F x* (*oVeblen F y a*) = *oVeblen F y a*
using *assms*
proof (*induction y arbitrary*: *x a rule*: *oLimit-induct*)
case *zero*
then show *?case*
by *auto*
next
case (*suc u*)
then show *?case*
by (*metis antisym-conv3 leD normal-oVeblen oDeriv-fixed oSuc-le-eq-less oVeblen-oSuc*)
next
case (*lim f x a*)
then obtain *n* **where** *x* < *f n*
using *less-oLimitD* **by** *blast*
have *oVeblen F* (*oLimit f*) *a* ∈ *range* (*oVeblen F* (*f n*))
by (*simp add*: *range-oVeblen-subset range-subsetD*)
then show *?case*
using *lim.IH* ‹*x* < *f n*› **by** *force*
qed

lemma (**in** *normal*) *critical-set-fixed*:
assumes *0* < *z*
shows *range* (*oVeblen F z*) = {*x*. ∀ *y*<*z*. *oVeblen F y x* = *x*} (**is** *?L* = *?R*)
proof
show *?L* ⊆ *?R*
using *oVeblen-fixed* **by** *auto*
have {*x*. ∀ *y*<*z*. *oVeblen F y x* = *x*} ⊆ *range* (*oVeblen F z*)
using *assms*

```
  proof (induction z rule: oLimit-induct)
    case zero
    then show ?case by auto
  next
    case (suc x)
    then show ?case
      by (force simp: normal-oVeblen oVeblen-oSuc range-oDeriv)
  next
    case (lim f)
    show ?case
    proof clarsimp
      fix x
      assume ∀y<oLimit f. oVeblen F y x = x
      then have x ∈ critical-set (range F) (f n) for n
        by (metis lim.hyps rangeI range-oVeblen strict-mono-less-oLimit)
      then show x ∈ range (oVeblen F (oLimit f))
        by (simp add: range-oVeblen critical-set-oLimit)
    qed
  qed
  then show ?R ⊆ ?L
    by blast
qed
```

9.5 Veblen hierarchy over $\lambda x.\ 1 + x$

```
lemma oDeriv-id: oDeriv id = id
proof
  fix x show oDeriv id x = id x
    by (induction x rule: oLimit-induct) (auto simp add: oFix-eq-self)
qed

lemma oFix-plus: oFix (λx. a + x) 0 = a ∗ ω
proof −
  have iter n ((+) a) 0 = a ∗ ordinal-of-nat n for n
  proof (induction n)
    case 0
    then show ?case by auto
  next
    case (Suc n)
    have a + a ∗ ordinal-of-nat n = a ∗ ordinal-of-nat n + a for n
      by (induction n) (simp-all flip: ordinal-plus-assoc)
    with Suc show ?case by simp
  qed
  then show ?thesis
    by (simp add: oFix-def omega-def)
qed

lemma oDeriv-plus: oDeriv ((+) a) = ((+) (a ∗ ω))
proof
```

show *oDeriv* ((+) *a*) *x* = *a* ∗ ω + *x* **for** *x*
proof (*induction x rule*: *oLimit-induct*)
case (*suc x*)
then show *?case*
by (*simp add*: *oFix-eq-self ordinal-plus-absorb*)
qed (*auto simp*: *oFix-plus*)
qed

lemma *oVeblen-1-plus*: *oVeblen* ((+) *1*) *x* = ((+) (ω ∗∗ *x*))
using *wf*
proof (*induction x rule*: *wf-induct-rule*)
case (*less x*)
have *oVeblen* ((+) (*oSuc 0*)) *x* = (+) (ω ∗∗ *x*)
proof (*cases x rule*: *ordinal-cases*)
case *zero*
then show *?thesis*
by (*simp add*: *normal.oVeblen-0 normal-plus*)
next
case (*suc y*)
with *less* **show** *?thesis*
by (*simp add*: *normal.oVeblen-oSuc*[*OF normal-plus*] *oDeriv-plus*)
next
case (*lim f*)
show *?thesis*
proof (*rule normal-range-eq*)
show *normal* (*oVeblen* ((+) (*oSuc 0*)) *x*)
using *normal.normal-oVeblen normal-plus* **by** *blast*
show *normal* ((+) (ω ∗∗ *x*))
using *normal-plus* **by** *blast*
have ∀ *y*<*oLimit f*. ω ∗∗ *y* + *u* = *u* ⟹ *u* ∈ *range* ((+) (*oLimit* (λ*n*. ω ∗∗ *f* *n*))) **for** *u*
by (*metis rangeI lim oLimit-leI ordinal-le-plusR strict-mono-less-oLimit ordinal-plus-minus2*)
moreover
have ω ∗∗ *y* + (*oLimit* (λ*n*. ω ∗∗ *f n*) + *u*) = *oLimit* (λ*n*. ω ∗∗ *f n*) + *u*
if *y* < *oLimit f* **for** *u y*
by (*metis absorb-omega-exp2 ordinal-exp-oLimit ordinal-plus-assoc that zero-less-omega*)
ultimately show *range* (*oVeblen* ((+) (*oSuc 0*)) *x*) = *range* ((+) (ω ∗∗ *x*))
using *less lim*
by (*force simp add*: *strict-mono-limit-ordinal normal.critical-set-fixed*[*OF normal-plus*])
qed
qed
then show *?case*
by *simp*
qed

end