

Countable Ordinals

Brian Huffman

December 12, 2009

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	3
1.1	Preliminary datatype for ordinals	3
1.2	Ordinal type	6
1.3	Induction over ordinals	8
2	Ordinal Induction	10
2.1	Zero and successor ordinals	10
2.1.1	Derived properties of 0 and oSuc	11
2.2	Strict monotonicity	12
2.3	Limit ordinals	13
2.3.1	Making strict monotonic sequences	16
2.4	Induction principle for ordinals	17
3	Continuity	18
3.1	Continuous functions	18
3.2	Normal functions	20
4	Recursive Definitions	22
4.1	Partial orders	23
4.2	Recursive definitions for <i>ordinal</i> \Rightarrow <i>ordinal</i>	25

5	Ordinal Arithmetic	26
5.1	Addition	27
5.2	Subtraction	28
5.3	Multiplication	30
5.4	Exponentiation	32
6	Inverse Functions	34
6.1	Division	37
6.2	Derived properties of division	39
6.3	Logarithms	40
7	Fixed-points	42
7.1	Derivatives of ordinal functions	44
8	Omega	46
8.1	Embedding naturals in the ordinals	46
8.2	Omega, the least limit ordinal	47
8.3	Arithmetic properties of ω	48
8.4	Additive principal ordinals	49
8.5	Cantor normal form	52
8.6	Epsilon 0	55
9	Veblen Hierarchies	55
9.1	Closed, unbounded sets	56
9.2	Ordering functions	57
9.3	Critical ordinals	59
9.4	Veblen hierarchy over a normal function	60
9.5	Veblen hierarchy over $\lambda x. 1 + x$	63

1 Definition of Ordinals

```
theory OrdinalDef
imports Main
begin
```

1.1 Preliminary datatype for ordinals

```
datatype ord0 = ord0-Zero | ord0-Lim nat  $\Rightarrow$  ord0
```

subterm ordering on ord0

definition

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

lemma wf-ord0-prec: wf ord0-prec

```
apply (unfold ord0-prec-def)
apply (rule wfUNIVI, induct-tac x)
apply (drule spec, erule mp, simp)
apply (drule spec, erule mp, auto)
done
```

lemmas ord0-prec-induct = wf-induct[OF wf-trancl[OF wf-ord0-prec]]

less-than-or-equal ordering on ord0

inductive-set ord0-leq :: (ord0 \times ord0) set **where**

```
 $\llbracket \forall a. (a,x) \in \text{ord0-prec}^+ \longrightarrow (\exists b. (b,y) \in \text{ord0-prec}^+ \wedge (a,b) \in \text{ord0-leq}) \rrbracket$ 
 $\implies (x,y) \in \text{ord0-leq}$ 
```

lemma ord0-leqI:

```
 $\llbracket \forall a. (a,x) \in \text{ord0-prec}^+ \longrightarrow (a,y) \in \text{ord0-leq} \text{ } O \text{ } \text{ord0-prec}^+ \rrbracket$ 
 $\implies (x,y) \in \text{ord0-leq}$ 
```

by (rule ord0-leq.intros, auto)

lemma ord0-leqD:

```
 $\llbracket (x,y) \in \text{ord0-leq}; (a,x) \in \text{ord0-prec}^+ \rrbracket \implies (a,y) \in \text{ord0-leq} \text{ } O \text{ } \text{ord0-prec}^+$ 
by (ind-cases (x,y)  $\in$  ord0-leq, auto)
```

lemma ord0-leq-refl: $(x, x) \in \text{ord0-leq}$

by (rule ord0-prec-induct, rule ord0-leqI, auto)

lemma ord0-leq-trans[rule-format]:

```
 $\forall y. (x,y) \in \text{ord0-leq} \longrightarrow$ 
 $(\forall z. (y,z) \in \text{ord0-leq} \longrightarrow (x,z) \in \text{ord0-leq})$ 
apply (rule ord0-prec-induct, clarify)
apply (rule ord0-leqI, clarify)
apply (drule spec, drule mp, assumption)
apply (drule ord0-leqD, assumption, clarify)
apply (drule spec, drule mp, assumption)
```

```

apply (drule ord0-leqD, assumption, clarify)
apply (drule spec, drule mp, assumption)
apply auto
done

```

```

lemma wf-ord0-leq: wf (ord0-leq O ord0-prec+)
apply (unfold wf-def, clarify)
apply (subgoal-tac  $\forall z. (z,x) \in \text{ord0-leq} \longrightarrow P z$ )
apply (drule spec, erule mp, rule ord0-leq-refl)
apply (rule ord0-prec-induct, clarify)
apply (drule spec, erule mp, clarify)
apply (drule ord0-leqD, assumption, clarify)
apply (drule spec, drule mp, assumption)
apply (drule spec, erule mp)
apply (erule ord0-leq-trans, assumption)
done

```

ordering on ord0

```

instantiation ord0 :: ord
begin

```

definition

ord0-less-def: $x < y \iff (x,y) \in \text{ord0-leq} O \text{ord0-prec}^+$

definition

ord0-le-def: $x \leq y \iff (x,y) \in \text{ord0-leq}$

instance ..

end

```

lemma ord0-order-refl[simp]: (x::ord0) ≤ x
by (unfold ord0-le-def, rule ord0-leq-refl)

```

```

lemma ord0-order-trans:  $\llbracket (x::\text{ord0}) \leq y; y \leq z \rrbracket \implies x \leq z$ 
by (unfold ord0-le-def, rule ord0-leq-trans)

```

```

lemma ord0-wf: wf  $\{(x,y::\text{ord0}). x < y\}$ 
apply (subgoal-tac  $\{(x,y). x < y\} = \text{ord0-leq} O \text{ord0-prec}^+$ )
apply (simp add: wf-ord0-leq)
apply (auto simp add: ord0-less-def)
done

```

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

lemma ord0-leI:

```

 $\llbracket \forall a::\text{ord0}. a < x \longrightarrow a < y \rrbracket \implies x \leq y$ 
apply (unfold ord0-less-def ord0-le-def)
apply (rule ord0-leqI[rule-format])

```

```

apply (drule spec, erule mp)
apply (erule rel-compI[OF ord0-leq-refl])
done

```

```

lemma ord0-less-le-trans:
 $\llbracket (x::ord0) < y; y \leq z \rrbracket \implies x < z$ 
apply (unfold ord0-le-def ord0-less-def, clarify)
apply (drule ord0-leqD, assumption, clarify)
by (rule rel-compI[OF ord0-leq-trans])

```

```

lemma ord0-le-less-trans:
 $\llbracket (x::ord0) \leq y; y < z \rrbracket \implies x < z$ 
apply (unfold ord0-le-def ord0-less-def, clarify)
by (rule rel-compI[OF ord0-leq-trans])

```

```

lemma rev-ord0-le-less-trans:
 $\llbracket (y::ord0) < z; x \leq y \rrbracket \implies x < z$ 
by (rule ord0-le-less-trans)

```

```

lemma ord0-less-trans:
 $\llbracket (x::ord0) < y; y < z \rrbracket \implies x < z$ 
apply (unfold ord0-less-def, clarify)
apply (drule ord0-leqD, assumption, clarify)
by (rule rel-compI[OF ord0-leq-trans trancl-trans])

```

```

lemma ord0-less-imp-le:  $(x::ord0) < y \implies x \leq y$ 
by (rule ord0-leI[rule-format], rule ord0-less-trans)

```

```

lemma ord0-linear-lemma:
fixes m :: ord0 and n :: ord0
shows  $m < n \vee n < m \vee (m \leq n \wedge n \leq m)$ 
apply (rule-tac x=m in spec)
apply (rule-tac a=n in ord0-less-induct, rename-tac n)
apply (rule allI, rename-tac m)
apply (rule-tac a=m in ord0-less-induct, rename-tac m)
apply (case-tac  $\forall a. a < n \longrightarrow a < m$ )
apply (rule disjI2)
apply (case-tac  $\forall a. a < m \longrightarrow a < n$ )
apply (rule disjI2)
apply (rule conjI, erule ord0-leI, erule ord0-leI)
apply (rule disjI1, clarsimp)
apply (drule spec, drule mp, assumption)
apply (erule rev-ord0-le-less-trans)
apply (force simp add: ord0-less-imp-le)
apply (rule disjI1, clarsimp)
apply (drule spec, drule mp, assumption)
apply (rule-tac x=m in spec, simp)
apply (erule rev-ord0-le-less-trans)
apply (force simp add: ord0-less-imp-le)

```

done

lemma *ord0-linear*: $(x::ord0) \leq y \vee y \leq x$
apply (*cut-tac ord0-linear-lemma*[of $x y$])
apply (*auto dest: ord0-less-imp-le*)
done

lemma *ord0-order-less-le*: $(x::ord0) < y = (x \leq y \wedge \neg y \leq x)$
apply (*rule iffI*)
apply (*clarsimp simp add: ord0-less-imp-le*)
apply (*drule ord0-less-le-trans, assumption*)
apply (*cut-tac a=x in wf-not-refl[OF ord0-wf], simp*)
apply (*cut-tac ord0-linear-lemma*[of $x y$], *simp*)
apply (*auto dest: ord0-less-imp-le*)
done

1.2 Ordinal type

definition

ord0rel :: $(ord0 \times ord0)$ set **where**
ord0rel = $\{(x,y). x \leq y \wedge y \leq x\}$

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

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

instantiation *ordinal* :: *ord*
begin

definition

ordinal-less-def: $x < y \iff (\forall a \in \text{Rep-ordinal } x. \forall b \in \text{Rep-ordinal } y. a < b)$

definition

ordinal-le-def: $x \leq y \iff (\forall a \in \text{Rep-ordinal } x. \forall b \in \text{Rep-ordinal } y. a \leq b)$

instance ..

end

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

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

```

lemma equiv-ord0rel: equiv UNIV ord0rel
apply (unfold equiv-def refl-on-def sym-def trans-def ord0rel-def)
apply (auto elim: ord0-order-trans)
done

lemma Abs-ordinal-eq[simp]:
(Abs-ordinal (ord0rel “ {x}) = Abs-ordinal (ord0rel “ {y}))
  = (x ≤ y ∧ y ≤ x)
apply (simp add: Abs-ordinal-inject quotientI)
apply (simp add: eq-equiv-class-iff[OF equiv-ord0rel])
apply (simp add: ord0rel-def)
done

lemma Abs-ordinal-le[simp]:
Abs-ordinal (ord0rel “ {x}) ≤ Abs-ordinal (ord0rel “ {y}) = (x ≤ y)
apply (auto simp add: ordinal-le-def)
apply (unfold ord0rel-def)
apply (auto elim: ord0-order-trans)
done

lemma Abs-ordinal-less[simp]:
Abs-ordinal (ord0rel “ {x}) < Abs-ordinal (ord0rel “ {y}) = (x < y)
apply (auto simp add: ordinal-less-def)
apply (unfold ord0rel-def)
apply (auto elim: ord0-less-le-trans[OF rev-ord0-le-less-trans])
done

lemma ordinal-order-refl: (x::ordinal) ≤ x
by (cases x, simp)

lemma ordinal-order-trans: (x::ordinal) ≤ y ⇒ y ≤ z ⇒ x ≤ z
by (cases x, cases y, cases z, auto elim: ord0-order-trans)

lemma ordinal-order-antisym: (x::ordinal) ≤ y ⇒ y ≤ x ⇒ x = y
by (cases x, cases y, simp)

lemma ordinal-order-less-le-not-le: ((x::ordinal) < y) = (x ≤ y ∧ ¬ y ≤ x)
by (cases x, cases y, auto simp add: ord0-order-less-le)

lemma ordinal-linear: (x::ordinal) ≤ y ∨ y ≤ x
by (cases x, cases y, simp add: ord0-linear)

lemma ordinal-wf: wf {(x,y::ordinal). x < y}
apply (rule wfUNIVI)
apply (rule-tac x=x in Abs-ordinal-cases2, clarify)
apply (rule ord0-less-induct, rename-tac a)
apply (drule spec, erule mp, clarify)
apply (rule-tac x=y in Abs-ordinal-cases2, simp)

```

done

```
instance ordinal :: wellorder
  apply (rule wf-wellorderI)
  apply (rule ordinal-wf)
  apply (intro-classes)
    apply (rule ordinal-order-less-le-not-le)
    apply (rule ordinal-order-refl)
    apply (rule ordinal-order-trans, assumption+)
    apply (rule ordinal-order-antisym, assumption+)
  apply (rule ordinal-linear)
done
```

1.3 Induction over ordinals

zero and strict limits

definition

```
oZero :: ordinal where
  oZero = Abs-ordinal (ord0rel “ {ord0-Zero})
```

definition

```
oStrictLimit :: (nat  $\Rightarrow$  ordinal)  $\Rightarrow$  ordinal where
  oStrictLimit f = Abs-ordinal
    (ord0rel “ {ord0-Lim ( $\lambda n.$  SOME  $x.$   $x \in$  Rep-ordinal (f n))})
```

induction over ordinals

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

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

```
lemma less-ord0-LimI:  $f n <$  ord0-Lim f
```

```
  apply (simp add: ord0-less-def)
  apply (rule rel-compI[OF ord0-leq-refl])
  apply (rule r-into-trancl)
  apply (auto simp add: ord0-prec-def)
```

done

```
lemma less-ord0-LimD:  $x <$  ord0-Lim f  $\Longrightarrow$   $\exists n.$   $x \leq f n$ 
```

```
  apply (simp add: ord0-less-def, clarify)
  apply (erule tranclE)
  apply (drule ord0-precD, clarify)
  apply (force simp add: ord0-le-def)
  apply (drule ord0-precD, clarify)
  apply (rule-tac  $x=n$  in exI)
  apply (rule ord0-less-imp-le)
  apply (auto simp add: ord0-less-def)
```

done

lemma *some-ord0rel*: $(x, \text{SOME } y. (x,y) \in \text{ord0rel}) \in \text{ord0rel}$
by (*rule-tac* $x=x$ **in** *someI*, *simp add: ord0rel-def*)

lemma *ord0-Lim-le*:
 $\forall n. f\ n \leq g\ n \implies \text{ord0-Lim } f \leq \text{ord0-Lim } g$
apply (*rule ord0-leI*[*rule-format*])
apply (*drule less-ord0-LimD*, *clarify*)
apply (*erule ord0-le-less-trans*)
apply (*drule-tac x=n in spec*)
apply (*erule ord0-le-less-trans*)
apply (*rule less-ord0-LimI*)
done

lemma *ord0-Lim-ord0rel*:
 $\forall n. (f\ n, g\ n) \in \text{ord0rel} \implies (\text{ord0-Lim } f, \text{ord0-Lim } g) \in \text{ord0rel}$
by (*simp add: ord0rel-def ord0-Lim-le*)

lemma *Abs-ordinal-oStrictLimit*:
Abs-ordinal (*ord0rel* “ {*ord0-Lim f*}”) = *oStrictLimit* ($\lambda n. \text{Abs-ordinal} (\text{ord0rel} \text{ “ } \{f\ n\})$)
apply (*simp add: oStrictLimit-def*)
apply (*rule ord0relD*)
apply (*rule ord0-Lim-ord0rel*)
apply (*simp add: some-ord0rel*)
done

lemma *oStrictLimit-induct*:
assumes *base*: $P\ \text{oZero}$
assumes *step*: $\bigwedge f. \forall n. P\ (f\ n) \implies P\ (\text{oStrictLimit } f)$
shows $P\ a$
apply (*cases a, clarsimp*)
apply (*induct-tac z*)
apply (*rule base*[*unfolded oZero-def*])
apply (*simp add: Abs-ordinal-oStrictLimit step*)
done

order properties of 0 and strict limits

lemma *oZero-least*: $\text{oZero} \leq x$
apply (*unfold oZero-def, cases x, clarsimp*)
apply (*induct-tac z, simp, atomize*)
apply (*rule ord0-less-imp-le*)
apply (*rule ord0-le-less-trans*)
apply (*erule spec*)
apply (*rule less-ord0-LimI*)
done

lemma *oStrictLimit-ub*: $f\ n < \text{oStrictLimit } f$
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 (simp, rule ord0relD[THEN conjunct1])
apply (rule some-ord0rel)
apply (rule less-ord0-LimI)
done

```

```

lemma oStrictLimit-lub:  $\forall n. f\ n < x \implies oStrictLimit\ f \leq x$ 
apply (erule contrapos-pp, simp add: linorder-not-less linorder-not-le)
apply (cases x, simp add: oStrictLimit-def)
apply (drule less-ord0-LimD, clarify)
apply (rule-tac x=n in exI)
apply (rule-tac x=f n in Abs-ordinal-cases2, simp, rename-tac y)
apply (erule ord0-order-trans)
apply (rule ord0relD[THEN conjunct2])
apply (rule some-ord0rel)
done

```

```

lemma less-oStrictLimitD:  $x < oStrictLimit\ f \implies \exists n. x \leq f\ n$ 
apply (erule contrapos-pp)
apply (simp add: linorder-not-less linorder-not-le)
apply (erule oStrictLimit-lub)
done

```

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 (unfold oSuc-def, rule oStrictLimit-ub)

```

```

lemma oSuc-leI:  $x < y \implies oSuc\ x \leq y$ 
by (unfold oSuc-def, rule oStrictLimit-lub, simp)

```

```

instantiation ordinal :: {zero, one}
begin

```

definition

```

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

```

definition

ordinal-one-def [simp]: $(1::\text{ordinal}) = \text{oSuc } 0$

instance ..

end

2.1.1 Derived properties of 0 and oSuc

lemma *less-oSuc-eq-le*: $(x < \text{oSuc } y) = (x \leq y)$

apply (*rule iffI*)

apply (*erule contrapos-pp, simp add: linorder-not-less linorder-not-le*)

apply (*erule oSuc-leI*)

apply (*erule order-le-less-trans[OF less-oSuc]*)

done

lemma *ordinal-0-le* [iff]: $0 \leq (x::\text{ordinal})$

by (*unfold ordinal-zero-def, rule oZero-least*)

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

by (*simp add: linorder-not-less*)

lemma *ordinal-le-0* [iff]: $(x \leq 0) = (x = (0::\text{ordinal}))$

by (*simp add: order-le-less*)

lemma *ordinal-neq-0* [iff]: $(x \neq 0) = (0 < (x::\text{ordinal}))$

by (*simp add: order-less-le*)

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

by (*simp add: linorder-not-less*)

lemma *oSuc-le-eq-less*: $(\text{oSuc } x \leq y) = (x < y)$

apply (*rule iffI*)

apply (*erule order-less-le-trans[OF less-oSuc]*)

apply (*erule oSuc-leI*)

done

lemma *zero-less-oSuc* [iff]: $0 < \text{oSuc } x$

by (*rule order-le-less-trans, rule ordinal-0-le, rule less-oSuc*)

lemma *oSuc-not-0* [iff]: $\text{oSuc } x \neq 0$

by *simp*

lemma *less-oSuc0* [iff]: $(x < \text{oSuc } 0) = (x = 0)$

by (*simp add: less-oSuc-eq-le*)

lemma *oSuc-less-oSuc* [iff]: $(\text{oSuc } x < \text{oSuc } y) = (x < y)$

apply (*rule iffI*)

apply (*simp add: less-oSuc-eq-le order-less-le-trans[OF less-oSuc]*)
apply (*erule order-le-less-trans[OF oSuc-leI less-oSuc]*)
done

lemma *oSuc-eq-oSuc [iff]: (oSuc x = oSuc y) = (x = y)*
by (*safe, erule contrapos-pp, simp add: linorder-neq-iff*)

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

lemma *le-oSucE:*
 $\llbracket x \leq oSuc\ y; x \leq y \implies R; x = oSuc\ y \implies R \rrbracket \implies R$
by (*auto simp add: order-le-less less-oSuc-eq-le*)

lemma *less-oSucE:*
 $\llbracket x < oSuc\ y; x < y \implies P; x = y \implies P \rrbracket \implies P$
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 ⟹ f A < f B*

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

lemma *strict-mono-natI:*
fixes *f :: nat ⇒ 'a::order*
shows $(\bigwedge n. f\ n < f\ (Suc\ n)) \implies strict-mono\ f$
apply (*rule strict-monoI*)
apply (*drule Suc-leI*)
apply (*drule le-add-diff-inverse*)
apply (*subgoal-tac ∀ k. f A < f (Suc A + k)*)
apply (*erule subst, erule spec*)
apply (*rule allI, induct-tac k, simp*)
apply (*erule order-less-trans, simp*)
done

lemma *mono-natI:*
fixes *f :: nat ⇒ 'a::order*
shows $(\bigwedge n. f\ n \leq f\ (Suc\ n)) \implies mono\ f$
apply (*rule monoI*)
apply (*drule le-add-diff-inverse*)
apply (*subgoal-tac ∀ k. f x ≤ f (x + k)*)
apply (*erule subst, erule spec*)
apply (*rule allI, induct-tac k, simp*)
apply (*erule order-trans, simp*)
done

```

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

```

```

lemma strict-mono-monoD:
fixes  $f :: 'a::order \Rightarrow 'b::order$ 
shows  $\llbracket \text{strict-mono } f; A \leq B \rrbracket \implies f A \leq f B$ 
by (rule monoD[OF strict-mono-mono])

```

```

lemma strict-mono-cancel-eq:
fixes  $f :: 'a::linorder \Rightarrow 'b::linorder$ 
shows strict-mono  $f \implies (f x = f y) = (x = y)$ 
apply safe
apply (rule-tac x=x and y=y in linorder-cases)
apply (drule strict-monoD, assumption, simp)
apply assumption
apply (drule strict-monoD, assumption, simp)
done

```

```

lemma strict-mono-cancel-less:
fixes  $f :: 'a::linorder \Rightarrow 'b::linorder$ 
shows strict-mono  $f \implies (f x < f y) = (x < y)$ 
apply safe
apply (rule-tac x=x and y=y in linorder-cases)
apply assumption
apply simp
apply (drule strict-monoD, assumption, simp)
apply (simp add: strict-monoD)
done

```

```

lemma strict-mono-cancel-le:
fixes  $f :: 'a::linorder \Rightarrow 'b::linorder$ 
shows strict-mono  $f \implies (f x \leq f y) = (x \leq y)$ 
apply (auto simp add: order-le-less)
apply (simp add: strict-mono-cancel-less)
apply (simp add: strict-mono-cancel-eq)
apply (simp add: strict-monoD)
done

```

2.3 Limit ordinals

definition

```

oLimit :: ( $\text{nat} \Rightarrow \text{ordinal}$ )  $\Rightarrow$  ordinal where
oLimit  $f = (\text{LEAST } k. \forall n. f n \leq k)$ 

```

```

lemma oLimit-leI:  $\forall n. f n \leq x \implies \text{oLimit } f \leq x$ 
apply (unfold oLimit-def)

```

apply (*erule Least-le*)
done

lemma *le-oLimit [iff]: $f n \leq oLimit f$*
apply (*unfold oLimit-def*)
apply (*rule-tac x=n in spec*)
apply (*rule-tac k=oStrictLimit f in LeastI*)
apply (*clarify, rule order-less-imp-le*)
apply (*rule oStrictLimit-ub*)
done

lemma *le-oLimitI: $x \leq f n \implies x \leq oLimit f$*
by (*erule order-trans, rule le-oLimit*)

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

lemma *less-oLimitD: $x < oLimit f \implies \exists n. x < f n$*
apply (*unfold oLimit-def*)
apply (*drule not-less-Least*)
apply (*simp add: linorder-not-le*)
done

lemma *less-oLimitE:*
 $\llbracket x < oLimit f; \bigwedge n. x < f n \implies P \rrbracket \implies P$
by (*auto dest: less-oLimitD*)

lemma *le-oLimitE:*
 $\llbracket x \leq oLimit f; \bigwedge n. x \leq f n \implies R; x = oLimit f \implies R \rrbracket \implies R$
by (*auto simp add: order-le-less dest: less-oLimitD*)

lemma *oLimit-const [simp]: $oLimit (\lambda n. x) = x$*
apply (*rule order-antisym[OF - le-oLimit]*)
apply (*rule oLimit-leI, simp*)
done

lemma *strict-mono-less-oLimit:*
 $strict-mono f \implies f n < oLimit f$
apply (*rule order-less-le-trans*)
apply (*erule strict-monoD, rule lessI*)
apply (*rule le-oLimit*)
done

lemma *oLimit-eqI:*
 $\llbracket \bigwedge n. \exists m. f n \leq g m; \bigwedge n. \exists m. g n \leq f m \rrbracket \implies oLimit f = oLimit g$
apply *atomize*
apply (*rule order-antisym*)
apply (*rule oLimit-leI, clarify*)
apply (*drule spec, erule exE, erule le-oLimitI*)

```

apply (rule oLimit-leI, clarify)
apply (drule spec, erule exE, erule le-oLimitI)
done

```

```

lemma oLimit-Suc:
 $f\ 0 < oLimit\ f \implies oLimit\ (\lambda n. f\ (Suc\ n)) = oLimit\ f$ 
apply (rule oLimit-eqI)
apply (rule exI, rule order-refl)
apply (case-tac n)
apply (drule less-oLimitD, clarify, rename-tac m)
apply (case-tac m, simp)
apply (rule-tac x=nat in exI)
apply (simp add: order-less-imp-le)
apply (rule-tac x=nat in exI, simp)
done

```

```

lemma oLimit-shift:
 $\forall n. f\ n < oLimit\ f \implies oLimit\ (\lambda n. f\ (n + k)) = oLimit\ f$ 
apply (induct-tac k, simp, rename-tac k)
apply (simp only: add-Suc-right add-Suc[symmetric])
apply (rule trans[OF oLimit-Suc], simp-all)
done

```

```

lemma oLimit-shift-mono:
 $mono\ f \implies oLimit\ (\lambda n. f\ (n + k)) = oLimit\ f$ 
apply (rule oLimit-eqI)
apply (rule exI, rule order-refl)
apply (rule-tac x=n in exI)
apply (erule monoD, simp)
done

```

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  $\implies$  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  $\implies$  (oSuc w < x) = (w < x)
apply (rule iffI)
apply (erule order-less-trans[OF less-oSuc])

```

apply (*simp add: linorder-not-le[symmetric]*)
apply (*erule contrapos-nn*)
apply (*auto simp add: order-le-less less-oSuc-eq-le*)
done

lemma *limit-ordinal-oLimitI*:
 $\forall n. f n < oLimit f \implies limit\text{-ordinal} (oLimit f)$
apply (*unfold limit-ordinal-def, simp*)
apply (*rule conjI*)
apply (*rule order-le-less-trans[OF ordinal-0-le]*)
apply (*erule spec*)
apply (*clarsimp simp add: less-oSuc-eq-le*)
apply (*drule oLimit-leI*)
apply (*simp add: linorder-not-less[symmetric]*)
done

lemma *strict-mono-limit-ordinal*:
 $strict\text{-mono} f \implies limit\text{-ordinal} (oLimit f)$
apply (*rule limit-ordinal-oLimitI*)
apply (*simp add: strict-mono-less-oLimit*)
done

lemma *limit-ordinalII*:
 $\llbracket 0 < z; \forall x < z. oSuc x < z \rrbracket \implies limit\text{-ordinal} z$
apply (*erule contrapos-pp*)
apply (*unfold limit-ordinal-def, clarsimp*)
apply (*drule-tac x=y in spec, clarsimp*)
done

2.3.1 Making strict monotonic sequences

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

lemma *f-make-mono-less*:
 $\forall n. f n < oLimit f \implies f (make\text{-mono} f n) < f (make\text{-mono} f (Suc n))$
apply (*drule-tac x=make-mono f n in spec*)
apply (*drule less-oLimitD, clarsimp*)
apply (*erule LeastI*)
done

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

lemma *le-f-make-mono*:
 $\llbracket \forall n. f n < oLimit f; m \leq make\text{-mono} f n \rrbracket \implies f m \leq f (make\text{-mono} f n)$

```

apply (auto simp add: order-le-less)
apply (case-tac n, simp-all)
apply (drule not-less-Least)
apply (simp add: linorder-not-less)
apply (erule order-le-less-trans)
apply (rule LeastI)
apply (erule f-make-mono-less)
done

```

lemma *make-mono-less*:

```

 $\forall n. f\ n < oLimit\ f \implies make\ mono\ f\ n < make\ mono\ f\ (Suc\ n)$ 
apply (frule-tac n=n in f-make-mono-less)
apply (rule ccontr, simp only: linorder-not-less)
apply (drule le-f-make-mono, assumption)
apply (simp add: linorder-not-less[symmetric])
done

```

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

lemma *oLimit-make-mono-eq*:

```

 $\forall n. f\ n < oLimit\ f \implies oLimit\ (\lambda n. f\ (make\ mono\ f\ n)) = oLimit\ f$ 
apply (rule oLimit-eqI, force)
apply (rule-tac x=n in exI)
apply (rule le-f-make-mono, assumption)
apply (induct-tac n, simp)
apply (rule Suc-leI)
apply (erule order-le-less-trans)
apply (erule make-mono-less)
done

```

2.4 Induction principle for ordinals

lemma *oLimit-le-oStrictLimit*: $oLimit\ f \leq oStrictLimit\ f$

```

apply (rule oLimit-leI, clarify)
apply (rule order-less-imp-le)
apply (rule oStrictLimit-ub)
done

```

lemma *oLimit-induct*:

assumes *zero*: $P\ 0$

and *suc*: $\bigwedge x. P\ x \implies P\ (oSuc\ x)$

and *lim*: $\bigwedge f. \llbracket strict\ mono\ f; \forall n. P\ (f\ n) \rrbracket \implies P\ (oLimit\ f)$

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 (drule less-oStrictLimitD, clarify)
apply (subgoal-tac oStrictLimit\ f = oSuc\ (f\ n), simp add: suc)

```

```

apply (rule order-antisym)
apply (rule oStrictLimit-lub, clarify)
apply (simp add: less-oSuc-eq-le)
apply (erule order-trans[OF le-oLimit])
apply (rule oSuc-leI, rule oStrictLimit-ub)
apply (subgoal-tac  $\forall n. f\ n < oLimit\ f$ )
apply (subgoal-tac  $P\ (oLimit\ (\lambda n. f\ (make-mono\ f\ n)))$ )
apply (simp add: oLimit-make-mono-eq)
apply (rule lim)
apply (erule strict-mono-f-make-mono)
apply simp
apply (simp add: oStrictLimit-ub)
done

```

```

lemma ordinal-cases:
assumes zero:  $a = 0 \implies P$ 
and suc:  $\bigwedge x. a = oSuc\ x \implies P$ 
and lim:  $\bigwedge f. \llbracket strict-mono\ f; a = oLimit\ f \rrbracket \implies P$ 
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) mono: mono  $F$ 
apply (rule monoI)
apply (cut-tac  $f=nat-case\ x\ (\lambda n. y)$  in cont)
apply (subgoal-tac  $\forall x\ y. oLimit\ (nat-case\ x\ (\lambda n. y)) = max\ x\ y$ )
apply (subgoal-tac  $\forall x\ y\ n. F\ (case\ n\ of\ 0 \Rightarrow x \mid Suc\ n \Rightarrow y)$ )

```

```

      = (case n of 0 => F x | Suc n => F y))
  apply (simp add: max-def)
  apply (erule ssubst, simp)
  apply (simp split: nat.split)
  apply (clarify, rule order-antisym)
  apply (rule oLimit-leI)
  apply (simp split: nat.split add: le-maxI1 le-maxI2)
  apply (simp, safe)
  apply (rule-tac n=0 in le-oLimitI, simp)
  apply (rule-tac n=1 in le-oLimitI, simp)
done

```

```

lemma (in continuous) monoD: x ≤ y ⟹ F x ≤ F y
by (erule monoD[OF mono])

```

```

lemma continuousI:
assumes lim: ⋀f. strict-mono f ⟹ F (oLimit f) = oLimit (λn. F (f n))
assumes suc: ⋀x. F x ≤ F (oSuc x)
shows continuous F
  apply (subgoal-tac mono F)
  apply (rule continuous.intro)
  apply (case-tac ∀ n. f n < oLimit f)
  apply (subgoal-tac oLimit (λn. f (make-mono f n)) = oLimit f)
  apply (erule subst)
  apply (rule trans[OF lim])
  apply (erule strict-mono-f-make-mono)
  apply (rule oLimit-eqI)
  apply (rule exI, rule order-refl)
  apply (rule-tac x=n in exI)
  apply (erule monoD)
  apply (rule le-f-make-mono, assumption)
  apply (induct-tac n, simp)
  apply (simp add: Suc-le-eq)
  apply (erule order-le-less-trans)
  apply (erule make-mono-less)
  apply (erule oLimit-make-mono-eq)
  apply (clarsimp simp add: linorder-not-less)
  apply (drule order-antisym[OF le-oLimit], simp)
  apply (rule order-antisym[OF le-oLimit])
  apply (rule oLimit-leI[rule-format])
  apply (erule monoD)
  apply (erule subst)
  apply (rule le-oLimit)
  apply (subgoal-tac ∀ y x. x ≤ y ⟹ F x ≤ F y)
  apply (rule monoI, simp)
  apply (rule allI, rule-tac a=y in oLimit-induct)
  apply simp
  apply (clarsimp, erule le-oSucE)
  apply (drule spec, drule mp, assumption)

```

```

  apply (erule order-trans, rule suc)
  apply simp
  apply (clarsimp simp add: lim, erule le-oLimitE)
  apply (drule-tac x=n in spec)
  apply (drule-tac x=x in spec, drule mp, assumption)
  apply (erule order-trans)
  apply (rule le-oLimit)
  apply (simp add: lim)
done

```

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:  $x \leq y \implies F x \leq F y$ 
by (rule monoD)

```

```

lemma (in normal) strict-monoD:  $x < y \implies F x < F y$ 
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:  $x \leq F x$ 
  apply (rule-tac a=x in oLimit-induct)
  apply simp
  apply (rule oSuc-leI)
  apply (erule order-le-less-trans)
  apply (rule strict-monoD[OF less-oSuc])
  apply (simp add: oLimit)
  apply (rule oLimit-leI, clarify)
  apply (rule order-trans, erule spec)
  apply (rule le-oLimit)

```

done

lemma *normalI*:

assumes *lim*: $\bigwedge f. \text{strict-mono } f \implies F (\text{oLimit } f) = \text{oLimit } (\lambda n. F (f n))$

assumes *suc*: $\bigwedge x. F x < F (\text{oSuc } x)$

shows *normal F*

apply (rule *normal.intro*[*OF - normal-axioms.intro*])
 apply (simp add: *continuousI order-less-imp-le suc lim*)
 apply (subgoal-tac $\forall y x. x < y \longrightarrow F x < F y$)
 apply (rule *strict-monoI, simp*)
 apply (rule *allI, rule-tac a=y in oLimit-induct*)
 apply *simp*
 apply (clarsimp, erule *less-oSucE*)
 apply (drule *spec, drule mp, assumption*)
 apply (erule *order-less-trans, rule suc*)
 apply (simp add: *suc*)
 apply (clarsimp simp add: *lim, erule less-oLimitE*)
 apply (drule *spec, drule spec, drule mp, assumption*)
 apply (erule *order-less-le-trans*)
 apply (rule *le-oLimit*)

done

lemma *normal-range-le*:

$\llbracket \text{normal } F; \text{normal } G; \text{range } G \subseteq \text{range } F \rrbracket \implies F x \leq G x$

apply (rule-tac *a=x in oLimit-induct*)
 apply (subgoal-tac $G 0 \in \text{range } F$)
 apply (clarsimp simp add: *normal.cancel-le*)
 apply (erule *subsetD, rule rangeI*)
 apply (subgoal-tac $G (\text{oSuc } x) \in \text{range } F$)
 apply (clarsimp simp add: *normal.cancel-le*)
 apply (rename-tac *y*)
 apply (rule *oSuc-leI*)
 apply (subgoal-tac $F x < F y, simp add: \text{normal.cancel-less}$)
 apply (erule *order-le-less-trans*)
 apply (erule *subst*)
 apply (simp add: *normal.cancel-less*)
 apply (erule *subsetD, rule rangeI*)
 apply (simp only: *normal.oLimit*)
 apply (rule *oLimit-leI*[*rule-format*])
 apply (rule-tac $n=n$ in *le-oLimitI*)
 apply (erule *spec*)

done

lemma *normal-range-eq*:

$\llbracket \text{normal } F; \text{normal } G; \text{range } F = \text{range } G \rrbracket \implies F = G$

apply (rule *ext, rule order-antisym*)
 apply (simp add: *normal-range-le*)
 apply (simp add: *normal-range-le*)

done

end

4 Recursive Definitions

theory *OrdinalRec*
imports *OrdinalCont*
begin

definition

$oPrec :: \text{ordinal} \Rightarrow \text{ordinal}$ **where**
 $oPrec\ x = (\text{THE } p. x = oSuc\ p)$

lemma *oPrec-oSuc* [*simp*]: $oPrec\ (oSuc\ x) = x$
by (*unfold oPrec-def, rule the-equality, simp-all*)

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

definition

$ordinal-rec0 ::$
 $['a, \text{ordinal} \Rightarrow 'a \Rightarrow 'a, (\text{nat} \Rightarrow 'a) \Rightarrow 'a, \text{ordinal}] \Rightarrow 'a$ **where**
 $ordinal-rec0\ z\ s\ l \equiv wfrec\ \{(x,y). x < y\}\ (\lambda F\ x.$
 $\text{if } x = 0 \text{ then } z \text{ else}$
 $\text{if } (\exists p. x = oSuc\ p) \text{ then } s\ (oPrec\ x)\ (F\ (oPrec\ x)) \text{ else}$
 $(\text{THE } y. \forall f. (\forall n. f\ n < oLimit\ f) \wedge oLimit\ f = x$
 $\longrightarrow l\ (\lambda n. F\ (f\ n)) = y))$

lemma *ordinal-rec0-0*:
 $ordinal-rec0\ z\ s\ l\ 0 = z$
apply (*rule trans[OF def-wfrec[OF ordinal-rec0-def wf]]*)
apply *simp*
done

lemma *ordinal-rec0-oSuc*:
 $ordinal-rec0\ z\ s\ l\ (oSuc\ x) = s\ x\ (ordinal-rec0\ z\ s\ l\ x)$
apply (*rule trans[OF def-wfrec[OF ordinal-rec0-def wf]]*)
apply (*simp add: tfl-cut-apply*)
done

lemma *limit-ordinal-not-0*: $limit\ ordinal\ x \Longrightarrow x \neq 0$
by (*clarsimp*)

lemma *limit-ordinal-not-oSuc*: $limit\ ordinal\ x \Longrightarrow x \neq oSuc\ p$
by (*clarsimp*)

lemma *ordinal-rec0-limit-ordinal*:

```

limit-ordinal x  $\implies$  ordinal-rec0 z s l x =
  (THE y.  $\forall f. (\forall n. f n < oLimit f) \wedge oLimit f = x \longrightarrow$ 
    l ( $\lambda n. ordinal-rec0 z s l (f n)$ ) = 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: tfl-cut-apply)
  apply (simp add: tfl-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:  $\bigwedge x y z. [x << y; y << z] \implies x << z$ 
    and po-antisym:  $\bigwedge x y. [x << y; y << x] \implies x = y$ 

lemma porder-order: porder (op  $\leq$  :: 'a::order  $\Rightarrow$  'a  $\Rightarrow$  bool)
  apply (rule porder.intro)
  apply (rule order-refl)
  apply (rule order-trans, assumption+)
  apply (rule order-antisym, assumption+)
done

```

```

lemma (in porder) flip: porder ( $\lambda x y. y << x$ )
  apply (rule porder.intro)
  apply (rule po-refl)
  apply (rule po-trans, assumption+)
  apply (rule po-antisym, assumption+)
done

```

```

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:  $\bigwedge f x. \forall n. f n << x \implies lub f << x$ 

```

```

lemma (in omega-complete) lub-cong-lemma:
   $\llbracket \forall n. f n < oLimit f; \forall m. g m < oLimit g; oLimit f \leq oLimit g; \forall y < oLimit g. \forall x \leq y. F x << F y \rrbracket$ 
 $\implies lub (\lambda n. F (f n)) << lub (\lambda n. F (g n))$ 
  apply (rule is-lub-lub[rule-format])
  apply (subgoal-tac f n < oLimit g)
  apply (drule less-oLimitD, clarify, rename-tac m)
  apply (drule-tac x=g m in spec, drule mp)
  apply (erule spec)
  apply (drule-tac x=f n in spec, drule mp)

```

apply (*erule order-less-imp-le*)
apply (*erule po-trans*)
apply (*rule is-ub-lub*)
apply (*rule order-less-le-trans*)
apply (*erule spec*)
apply *assumption*
done

lemma (*in omega-complete*) *lub-cong*:
 $\llbracket \forall n. f\ n < oLimit\ f; \forall m. g\ m < oLimit\ g; oLimit\ f = oLimit\ g; \forall y < oLimit\ g. \forall x \leq y. F\ x << F\ y \rrbracket$
 $\implies lub\ (\lambda n. F\ (f\ n)) = lub\ (\lambda n. F\ (g\ n))$
apply (*rule po-antisym*)
apply (*rule lub-cong-lemma, assumption+*)
apply (*simp add: po-refl*)
apply *assumption*
apply (*rule lub-cong-lemma, assumption+*)
apply (*simp add: po-refl*)
apply (*erule sym, simp*)
done

lemma (*in omega-complete*) *ordinal-rec0-mono-lemma*:
assumes $s: \forall p\ x. x << s\ p\ x$
shows $\forall y \leq w. \forall x \leq y. ordinal-rec0\ z\ s\ lub\ x << ordinal-rec0\ z\ s\ lub\ y$
apply (*rule-tac a=w in oLimit-induct*)
apply (*simp add: po-refl*)
apply *clarify*
apply (*erule le-oSucE, simp, clarsimp*)
apply (*erule le-oSucE*)
apply (*erule spec, drule mp, rule order-refl*)
apply (*erule spec, drule mp, assumption*)
apply (*erule po-trans*)
apply (*simp add: ordinal-rec0-oSuc s*)
apply (*simp add: po-refl*)
apply *clarify*
apply (*erule le-oLimitE*)
apply *simp*
apply *clarsimp*
apply (*subgoal-tac ordinal-rec0 z s lub (oLimit f) = lub (\lambda n. ordinal-rec0 z s lub (f n))*)
apply (*erule le-oLimitE*)
apply (*erule-tac x=n in spec*)
apply (*erule spec, drule mp, rule order-refl*)
apply (*erule spec, drule mp, assumption*)
apply (*erule po-trans*)
apply (*simp, rule is-ub-lub*)
apply (*simp add: po-refl*)
apply (*simp only: ordinal-rec0-limit-ordinal strict-mono-limit-ordinal*)
apply (*rule the-equality, clarify*)

```

apply (rule lub-cong, assumption)
  apply (simp add: strict-mono-less-oLimit)
  apply assumption
apply clarify
apply (drule less-oLimitD, clarify)
apply (drule order-less-imp-le)
apply simp
apply (drule-tac x=f in spec, simp add: strict-mono-less-oLimit)
done

```

```

lemma (in omega-complete) ordinal-rec0-mono:
assumes s:  $\forall p x. x \ll s p x$ 
shows  $x \leq y \implies \text{ordinal-rec0 } z s \text{ lub } x \ll \text{ordinal-rec0 } z s \text{ lub } y$ 
  apply (rule ordinal-rec0-mono-lemma[OF s, rule-format])
  apply (rule order-refl)
apply assumption
done

```

```

lemma (in omega-complete) ordinal-rec0-oLimit:
assumes s:  $\forall p x. x \ll s p x$ 
shows  $\text{ordinal-rec0 } z s \text{ lub } (oLimit f) =$ 
   $\text{lub } (\lambda n. \text{ordinal-rec0 } z s \text{ lub } (f n))$ 
apply (case-tac  $\forall n. f n < oLimit f$ )
apply (simp add: ordinal-rec0-limit-ordinal limit-ordinal-oLimitI)
apply (rule the-equality, clarify)
apply (rule lub-cong, assumption+)
apply clarify
apply (erule ordinal-rec0-mono[OF s])
apply (drule-tac x=f in spec, simp)
apply (simp add: linorder-not-less, clarify)
apply (rule po-antisym)
apply (erule po-trans[OF ordinal-rec0-mono[OF s]])
apply (rule is-ub-lub)
apply (rule is-lub-lub[rule-format])
apply (rule ordinal-rec0-mono[OF s le-oLimit])
done

```

4.2 Recursive definitions for $\text{ordinal} \Rightarrow \text{ordinal}$

definition

```

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

```

lemma omega-complete-oLimit: $\text{omega-complete } (op \leq) \text{ oLimit}$

```

apply (rule omega-complete.intro)
apply (rule porder-order)
apply (rule omega-complete-axioms.intro)
apply (rule le-oLimit)

```

apply (*erule oLimit-leI*)
done

lemma *ordinal-rec-0* [*simp*]: *ordinal-rec z s 0 = z*
by (*unfold ordinal-rec-def, rule ordinal-rec0-0*)

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 ($\lambda n. \text{ordinal-rec } z \text{ s } (f n)$)*
apply (*unfold ordinal-rec-def*)
apply (*rule omega-complete.ordinal-rec0-oLimit*)
apply (*rule omega-complete-oLimit*)
apply (*rule s*)
done

lemma *continuous-ordinal-rec*:
assumes *s: $\forall p x. x \leq s p x$*
shows *continuous (ordinal-rec z s)*
apply (*rule continuousI*)
apply (*rule ordinal-rec-oLimit[OF s]*)
apply (*simp add: s*)
done

lemma *mono-ordinal-rec*:
assumes *s: $\forall p x. x \leq s p x$*
shows *mono (ordinal-rec z s)*
apply (*rule continuous.mono*)
apply (*rule continuous-ordinal-rec[OF s]*)
done

lemma *normal-ordinal-rec*:
assumes *s: $\forall p x. x < s p x$*
shows *normal (ordinal-rec z s)*
apply (*rule normalI*)
apply (*rule ordinal-rec-oLimit*)
apply (*simp add: s order-less-imp-le*)
apply (*simp add: s*)
done

end

5 Ordinal Arithmetic

theory *OrdinalArith*

```

imports OrdinalRec
begin

```

5.1 Addition

```

instantiation ordinal :: plus
begin

```

```

definition
  op + = (λx. ordinal-rec x (λp. oSuc))

```

```

instance ..

```

```

end

```

```

lemma normal-plus: normal (op + 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 (λn. 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]:
  ∀ x x'. x ≤ x' → x + y ≤ 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 ≤ y' ⇒ x + y ≤ x + (y'::ordinal)
by (rule normal.monoD[OF normal-plus])

```

```

lemma ordinal-plus-mono:
  ⟦x ≤ x'; y ≤ y'⟧ ⇒ x + y ≤ x' + (y'::ordinal)
by (rule order-trans[OF ordinal-plus-monoL ordinal-plus-monoR])

```

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

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

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

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

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

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

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

lemma *ordinal-plus-not-0*: $(0 < x + y) = (0 < x \vee 0 < (y::\text{ordinal}))$
apply *safe*
apply *simp*
apply (*erule order-less-le-trans, rule ordinal-le-plusR*)
apply (*erule order-less-le-trans, rule ordinal-le-plusL*)
done

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

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

5.2 Subtraction

instantiation *ordinal* :: *minus*
begin

definition

minus-ordinal-def:

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

instance ..

end

lemma *continuous-minus*: *continuous* ($\lambda x. x - y$)
apply (*unfold minus-ordinal-def*)
apply (*rule continuous-ordinal-rec*)
apply (*simp add: order-less-imp-le*)
done

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

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

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

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

lemma *ordinal-oSuc-minus2*: $x < y \implies \text{oSuc } x - y = x - y$
by (*simp add: minus-ordinal-def linorder-not-le[symmetric]*)

lemma *ordinal-minus-eq-0* [*rule-format, simp*]:
 $x \leq y \longrightarrow x - y = (0::\text{ordinal})$
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*]: $(x + y) - x = (y::\text{ordinal})$
by (*rule-tac a=y in oLimit-induct, simp-all*)

lemma *ordinal-plus-minus2* [*simp*]: $x \leq y \implies x + (y - x) = (y::\text{ordinal})$
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*: $x = y + z \implies x - y = (z::\text{ordinal})$
by *simp*

lemma *ordinal-minus-less-eq* [*simp*]:

```

(y::ordinal) ≤ x ⇒ (x − y < z) = (x < y + z)
apply (subgoal-tac (x − y < z) = (y + (x − y) < y + z), simp)
apply (simp only: ordinal-plus-left-cancel-less)
done

```

```

lemma ordinal-minus-le-eq [simp]:
(x − y ≤ z) = (x ≤ y + (z::ordinal))
apply (rule-tac x=x and y=y in linorder-le-cases)
apply (simp, erule order-trans, simp)
apply (subgoal-tac (x − y ≤ z) = (y + (x − y) ≤ y + z), simp)
apply (simp only: ordinal-plus-left-cancel-le)
done

```

```

lemma ordinal-minus-monoL: x ≤ y ⇒ x − z ≤ y − (z::ordinal)
by (erule continuous.monoD[OF continuous-minus])

```

```

lemma ordinal-minus-monoR: x ≤ y ⇒ z − y ≤ z − (x::ordinal)
apply (rule-tac x=y and y=z in linorder-le-cases)
apply (subst ordinal-minus-le-eq)
apply (subgoal-tac x + (z − x) ≤ y + (z − x))
apply (erule order-trans, assumption, simp)
apply (erule ordinal-plus-monoL)
apply simp
done

```

5.3 Multiplication

```

instantiation ordinal :: times
begin

```

```

definition
times-ordinal-def: op * = (λx. ordinal-rec 0 (λp w. w + x))

```

```

instance ..

```

```

end

```

```

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

```

```

lemma normal-times: 0 < x ⇒ normal (op * x)
apply (unfold times-ordinal-def)
apply (rule normal-ordinal-rec[rule-format], rename-tac y)
apply (subgoal-tac y + 0 < y + x, simp)
apply (simp only: ordinal-plus-left-cancel-less)
done

```

```

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

```

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

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

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

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

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

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

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

lemma *ordinal-times-monoL* [*rule-format*]:
 $\forall x\ x'. x \leq x' \longrightarrow x * y \leq x' * (y::ordinal)$
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*:
 $\llbracket x \leq x'; y \leq y' \rrbracket \Longrightarrow x * y \leq x' * (y'::ordinal)$
by (*rule order-trans[OF ordinal-times-monoL ordinal-times-monoR]*)

lemma *ordinal-times-strict-monoR*:
 $\llbracket y < y'; 0 < x \rrbracket \Longrightarrow x * y < x * (y'::ordinal)$
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]: $0 < y \implies x \leq x * (y::\text{ordinal})$
by (*drule ordinal-times-monoR*[*OF oSuc-leI*], *simp*)

lemma *ordinal-less-timesR*: $\llbracket 0 < x; \text{oSuc } 0 < y \rrbracket \implies x < x * (y::\text{ordinal})$
by (*drule ordinal-times-strict-monoR*, *assumption*, *simp*)

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

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

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

lemma *ordinal-times-eq-0*:
 $((x::\text{ordinal}) * y = 0) = (x = 0 \vee y = 0)$
apply (*rule iffI*)
apply (*erule contrapos-pp*, *clarsimp*)
apply (*drule oSuc-leI*)
apply (*erule order-less-le-trans*)
apply (*drule ordinal-times-monoL*, *simp*)
apply *auto*
done

lemma *ordinal-times-not-0* [simp]:
 $((0::\text{ordinal}) < x * y) = (0 < x \wedge 0 < y)$
by (*rule not-inject*, *simp add: ordinal-times-eq-0*)

5.4 Exponentiation

definition

exp-ordinal :: $[\text{ordinal}, \text{ordinal}] \Rightarrow \text{ordinal}$ (**infixr** ** 75) **where**
 $op ** = (\lambda x. \text{if } 0 < x \text{ then } \text{ordinal-rec } 1 (\lambda p w. w * x)$
 $\quad \text{else } (\lambda y. \text{if } y = 0 \text{ then } 1 \text{ else } 0))$

lemma *continuous-exp*: $0 < x \implies \text{continuous } (op ** x)$
by (*simp add: exp-ordinal-def continuous-ordinal-rec*)

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

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

```

lemma ordinal-exp-oLimit [simp]:
   $0 < x \implies x ** oLimit f = oLimit (\lambda n. x ** f n)$ 
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]:  $(0 < x ** y) = (0 < x \vee y = 0)$ 
  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]:  $(x ** y = 0) = (x = 0 \wedge 0 < y)$ 
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]:
   $\forall x x'. x \leq x' \longrightarrow x ** y \leq x' ** (y::ordinal)$ 
  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:  $oSuc 0 < x \implies normal (op ** x)$ 

```

```

apply (frule-tac order-less-trans[OF less-oSuc])
apply (rule normalI, simp, rename-tac y)
apply (subgoal-tac x ** y * 1 < x ** y * x, simp)
apply (subst ordinal-times-left-cancel-less)
apply simp
apply simp
done

```

```

lemma ordinal-exp-monoR:
 $\llbracket 0 < x; y \leq y' \rrbracket \implies x ** y \leq x ** (y'::ordinal)$ 
by (rule continuous.monoD[OF continuous-exp])

```

```

lemma ordinal-exp-mono:
 $\llbracket 0 < x'; x \leq x'; y \leq y' \rrbracket \implies x ** y \leq x' ** (y'::ordinal)$ 
by (rule order-trans[OF ordinal-exp-monoL ordinal-exp-monoR])

```

```

lemma ordinal-exp-strict-monoR:
 $\llbracket oSuc\ 0 < x; y < y' \rrbracket \implies x ** y < x ** (y'::ordinal)$ 
by (rule normal.strict-monoD[OF normal-exp])

```

```

lemma ordinal-le-expR [simp]:  $0 < y \implies x \leq x ** (y::ordinal)$ 
apply (subgoal-tac x ** oSuc 0  $\leq$  x ** y)
apply (simp del: ordinal-exp-oSuc)
apply (case-tac x = 0, simp)
apply (rule ordinal-exp-monoR, simp-all add: oSuc-leI)
done

```

```

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

```

```

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

```

```

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

```

end

6 Inverse Functions

```

theory OrdinalInverse
imports OrdinalArith
begin

```

```

lemma (in normal) oInv-ex:

```

$F 0 \leq a \implies \exists q. F q \leq a \wedge a < F (oSuc q)$
apply (subgoal-tac $\forall z. a < F z \longrightarrow (\exists q < z. F q \leq a \wedge a < F (oSuc q))$)
apply (drule-tac $x=oSuc a$ **in** spec, drule mp)
apply (rule-tac $y=F a$ **in** order-le-less-trans)
apply (rule increasing)
apply (rule strict-monoD[OF less-oSuc])
apply force
apply (rule allI, rule-tac $a=z$ **in** oLimit-induct)
apply simp
apply clarsimp
apply (case-tac $a < F x$)
apply clarsimp
apply (rule-tac $x=q$ **in** exI)
apply (simp add: order-less-trans[OF - less-oSuc])
apply (rule-tac $x=x$ **in** exI, simp)
apply (clarsimp simp add: oLimit)
apply (drule less-oLimitD, clarify)
apply (drule spec, drule mp, assumption)
apply (clarify, rule-tac $x=q$ **in** exI)
apply (simp add: order-less-le-trans[OF - le-oLimit])
done

lemma oInv-uniq:
 $\llbracket mono (F::ordinal \Rightarrow ordinal);$
 $F x \leq a \wedge a < F (oSuc x); F y \leq a \wedge a < F (oSuc y) \rrbracket$
 $\implies x = y$
apply clarify
apply (rule-tac $x=x$ **and** $y=y$ **in** linorder-cases)
apply (subgoal-tac $a < a$, simp)
apply (erule-tac $y=F (oSuc x)$ **in** order-less-le-trans)
apply (rule-tac $y=F y$ **in** order-trans)
apply (erule monoD, erule oSuc-leI)
apply assumption
apply assumption
apply (subgoal-tac $a < a$, simp)
apply (erule-tac $y=F (oSuc y)$ **in** order-less-le-trans)
apply (rule-tac $y=F x$ **in** order-trans)
apply (erule monoD, erule oSuc-leI)
apply assumption
done

definition

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

lemma (**in** normal) oInv-bounds:

$F 0 \leq a \implies F (oInv F a) \leq a \wedge a < F (oSuc (oInv F a))$
apply (simp add: oInv-def)
apply (rule theI')

```

apply (rule ex-ex1I)
apply (simp add: oInv-ex)
apply (simp add: oInv-uniq[OF mono])
done

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

lemma (in normal) oInv-bound2:
 $a < F\ (oSuc\ (oInv\ F\ a))$ 
apply (case-tac  $F\ 0 \leq a$ )
apply (simp only: oInv-bounds[THEN conjunct2])
apply (simp add: oInv-def, simp add: linorder-not-le)
apply (erule order-less-le-trans)
apply (simp add: cancel-le)
done

lemma (in normal) oInv-equality:
 $\llbracket F\ x \leq a; a < F\ (oSuc\ x) \rrbracket \implies oInv\ F\ a = x$ 
apply (subgoal-tac  $F\ 0 \leq a$ )
apply (simp add: oInv-def)
apply (rule the-equality)
apply simp
apply (simp add: oInv-uniq[OF mono])
apply (rule-tac  $y = F\ x$  in order-trans)
apply (simp add: cancel-le)
apply assumption
done

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

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

lemma (in normal) oInv-eq-0:  $a \leq F\ 0 \implies oInv\ F\ a = 0$ 
apply (case-tac  $F\ 0 \leq a$ )
apply (rule oInv-equality')
apply (simp only: order-antisym)
apply (simp add: oInv-def)
done

lemma (in normal) oInv-less:
 $\llbracket F\ 0 \leq a; a < F\ z \rrbracket \implies oInv\ F\ a < z$ 
apply (subst cancel-less[symmetric])
apply (simp only: order-le-less-trans[OF oInv-bound1])
done

```

```

lemma (in normal) le-oInv:
   $F z \leq a \implies z \leq oInv F a$ 
  apply (subst less-oSuc-eq-le[symmetric])
  apply (subst cancel-less[symmetric])
  apply (erule order-le-less-trans)
  apply (rule oInv-bound2)
done

```

```

lemma (in normal) less-oInvD:
   $x < oInv F a \implies F (oSuc x) \leq a$ 
  apply (case-tac  $F 0 \leq a$ )
  apply (rule order-trans[OF - oInv-bound1])
  apply (simp add: cancel-le oSuc-leI)
  apply assumption
  apply (simp add: oInv-def)
done

```

```

lemma (in normal) oInv-le:
   $a < F (oSuc x) \implies oInv F a \leq x$ 
  apply (erule contrapos-pp)
  apply (simp add: linorder-not-less linorder-not-le less-oInvD)
done

```

```

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

```

```

lemma (in normal) oInv-decreasing:
   $F 0 \leq x \implies oInv F x \leq x$ 
  apply (subst cancel-le[symmetric])
  apply (rule-tac  $y=x$  in order-trans)
  apply (erule oInv-bound1)
  apply (rule increasing)
done

```

6.1 Division

```

instantiation ordinal :: {Divides.div}
begin

```

definition*div-ordinal-def:* $x \text{ div } y = (\text{if } 0 < y \text{ then } oInv (op * y) x \text{ else } 0)$ **definition***mod-ordinal-def:* $x \text{ mod } y = ((x::ordinal) - y * (x \text{ div } y))$ **instance ..****end****lemma ordinal-divI:** $\llbracket x = y * q + r; r < y \rrbracket \implies x \text{ div } y = (q::ordinal)$ **apply** (*simp add: div-ordinal-def, safe*)**apply** (*simp add: normal.oInv-equality[OF normal-times]*)**done****lemma ordinal-times-div-le:** $y * (x \text{ div } y) \leq (x::ordinal)$ **apply** (*simp add: div-ordinal-def, safe*)**apply** (*erule normal.oInv-bound1[OF normal-times]*)**apply** *simp***done****lemma ordinal-less-times-div-plus:** $0 < y \implies x < y * (x \text{ div } y) + (y::ordinal)$ **apply** (*simp add: div-ordinal-def*)**apply** (*subst ordinal-times-oSuc[symmetric]*)**apply** (*erule normal.oInv-bound2[OF normal-times]*)**done****lemma ordinal-modI:** $\llbracket x = y * q + r; r < y \rrbracket \implies x \text{ mod } y = (r::ordinal)$ **apply** (*unfold mod-ordinal-def*)**apply** (*rule ordinal-minusI*)**apply** (*simp add: ordinal-divI*)**done****lemma ordinal-mod-less:** $0 < y \implies x \text{ mod } y < (y::ordinal)$ **apply** (*unfold mod-ordinal-def*)**apply** (*simp add: ordinal-times-div-le*)**apply** (*simp add: div-ordinal-def*)**apply** (*subst ordinal-times-oSuc[symmetric]*)**apply** (*erule normal.oInv-bound2[OF normal-times]*)**done****lemma ordinal-div-plus-mod:** $y * (x \text{ div } y) + (x \text{ mod } y) = (x::ordinal)$ **apply** (*simp add: mod-ordinal-def*)**apply** (*rule ordinal-plus-minus2*)**apply** (*rule ordinal-times-div-le*)**done**

lemma *ordinal-div-less*: $x < y * z \implies x \text{ div } y < (z::\text{ordinal})$
apply (*auto simp add: div-ordinal-def*)
apply (*simp add: normal.oInv-less[OF normal-times]*)
done

lemma *ordinal-le-div*: $\llbracket 0 < y; y * z \leq x \rrbracket \implies (z::\text{ordinal}) \leq x \text{ div } y$
apply (*auto simp add: div-ordinal-def*)
apply (*simp add: normal.le-oInv[OF normal-times]*)
done

lemma *ordinal-mono-div*: *mono* ($\lambda x. x \text{ div } y::\text{ordinal}$)
apply (*case-tac y = 0*)
apply (*simp add: div-ordinal-def monoI*)
apply (*simp add: div-ordinal-def normal.mono-oInv[OF normal-times]*)
done

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

lemma *ordinal-div-decreasing*: $(x::\text{ordinal}) \text{ div } y \leq x$
apply (*auto simp add: div-ordinal-def*)
apply (*simp add: normal.oInv-decreasing[OF normal-times]*)
done

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

lemma *ordinal-mod-0*: $x \text{ mod } 0 = (x::\text{ordinal})$
by (*simp add: mod-ordinal-def*)

6.2 Derived properties of division

lemma *ordinal-div-1 [simp]*: $x \text{ div } \text{oSuc } 0 = x$
by (*rule-tac r=0 in ordinal-divI, simp-all*)

lemma *ordinal-mod-1 [simp]*: $x \text{ mod } \text{oSuc } 0 = 0$
by (*rule-tac q=x in ordinal-modI, simp-all*)

lemma *ordinal-div-self [simp]*: $0 < x \implies x \text{ div } x = (1::\text{ordinal})$
by (*rule-tac r=0 in ordinal-divI, simp-all*)

lemma *ordinal-mod-self [simp]*: $x \text{ mod } x = (0::\text{ordinal})$
apply (*case-tac x=0, simp add: ordinal-mod-0, simp*)
apply (*rule-tac q=1 in ordinal-modI, simp-all*)
done

lemma *ordinal-div-greater [simp]*: $x < y \implies x \text{ div } y = (0::\text{ordinal})$
by (*rule-tac r=x in ordinal-divI, simp-all*)

lemma *ordinal-mod-greater* [*simp*]: $x < y \implies x \text{ mod } y = (x::\text{ordinal})$
by (*rule-tac* $q=0$ **in** *ordinal-modI*, *simp-all*)

lemma *ordinal-0-div* [*simp*]: $0 \text{ div } x = (0::\text{ordinal})$
by (*case-tac* $x=0$, *simp add: ordinal-div-0*, *simp*)

lemma *ordinal-0-mod* [*simp*]: $0 \text{ mod } x = (0::\text{ordinal})$
by (*case-tac* $x=0$, *simp add: ordinal-mod-0*, *simp*)

lemma *ordinal-1-dvd* [*simp*]: $\text{oSuc } 0 \text{ dvd } x$
by (*rule-tac* $k=x$ **in** *dvdI*, *simp*)

lemma *ordinal-dvd-mod*: $y \text{ dvd } x = (x \text{ mod } y = (0::\text{ordinal}))$
apply *safe*
apply (*erule* *dvdE*)
apply (*case-tac* $y=0$, *simp add: ordinal-mod-0*, *simp*)
apply (*rule* *ordinal-modI*, *simp*, *simp*)
apply (*cut-tac* $x=x$ **and** $y=y$ **in** *ordinal-div-plus-mod*)
apply (*rule-tac* $k=x \text{ div } y$ **in** *dvdI*, *simp*)
done

lemma *ordinal-dvd-times-div*:
 $y \text{ dvd } x \implies y * (x \text{ div } y) = (x::\text{ordinal})$
apply (*cut-tac* $x=x$ **and** $y=y$ **in** *ordinal-div-plus-mod*)
apply (*simp add: ordinal-dvd-mod*)
done

lemma *ordinal-dvd-oLimit*: $\forall n. x \text{ dvd } f n \implies x \text{ dvd } \text{oLimit } f$
apply (*rule-tac* $k=\text{oLimit } (\lambda n. f n \text{ div } x)$ **in** *dvdI*)
apply (*simp add: ordinal-dvd-times-div*)
done

6.3 Logarithms

definition

oLog :: *ordinal* \Rightarrow *ordinal* \Rightarrow *ordinal* **where**
oLog $b = (\lambda x. \text{if } 1 < b \text{ then } \text{oInv } (\text{op } ** b) x \text{ else } 0)$

lemma *ordinal-oLogI*:

$\llbracket b ** y \leq x; x < b ** y * b \rrbracket \implies \text{oLog } b x = y$
apply (*rule-tac* $x=1$ **and** $y=b$ **in** *linorder-cases*, *simp-all*)
apply (*simp add: oLog-def normal.oInv-equality[OF normal-exp]*)
done

lemma *ordinal-exp-oLog-le*:

$\llbracket 0 < x; \text{oSuc } 0 < b \rrbracket \implies b ** (\text{oLog } b x) \leq x$
apply (*simp add: oLog-def*)
apply (*frule-tac order-less-trans[OF less-oSuc]*)

apply (*simp add: normal.oInv-bound1*[*OF normal-exp*] *oSuc-leI*)
done

lemma ordinal-less-exp-oLog:
 $oSuc\ 0 < b \implies x < b ** (oLog\ b\ x) * b$
apply (*simp add: oLog-def*)
apply (*subst ordinal-exp-oSuc*[*symmetric*])
apply (*erule normal.oInv-bound2*[*OF normal-exp*])
done

lemma ordinal-oLog-less:
 $\llbracket 0 < x; oSuc\ 0 < b; x < b ** y \rrbracket \implies oLog\ b\ x < y$
apply (*simp add: oLog-def*)
apply (*frule-tac order-less-trans*[*OF less-oSuc*])
apply (*simp add: normal.oInv-less*[*OF normal-exp*] *oSuc-leI*)
done

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

lemma ordinal-oLogI2:
 $\llbracket oSuc\ 0 < b; x = b ** y * q + r; 0 < q; q < b; r < b ** y \rrbracket \implies oLog\ b\ x = y$
apply *simp*
apply (*rule ordinal-oLogI*)
apply (*rule-tac y=b ** y * q in order-trans, simp, simp*)
apply (*rule order-less-le-trans*)
apply (*erule ordinal-plus-strict-monoR*)
apply (*subst ordinal-times-oSuc*[*symmetric*])
apply (*rule ordinal-times-monoR*)
apply (*erule oSuc-leI*)
done

lemma ordinal-div-exp-oLog-less:
 $oSuc\ 0 < b \implies x\ div\ (b ** oLog\ b\ x) < b$
apply (*frule-tac order-less-trans*[*OF less-oSuc*])
apply (*case-tac x=0, simp-all*)
apply (*rule ordinal-div-less*)
by (*rule 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*: $oSuc\ 0 < b \implies oLog\ b\ (b\ **\ x) = x$
by (*simp add: oLog-def normal.oInv-inverse[OF normal-exp]*)

lemma *ordinal-oLog-self*: $oSuc\ 0 < b \implies oLog\ b\ b = oSuc\ 0$
apply (*subgoal-tac oLog b (b ** oSuc 0) = oSuc 0*)
apply (*simp only: ordinal-exp-1*)
apply (*simp only: ordinal-oLog-exp*)
done

lemma *ordinal-mono-oLog*: *mono* (*oLog b*)
apply (*case-tac oSuc 0 < b*)
apply (*simp add: oLog-def normal.mono-oInv[OF normal-exp]*)
apply (*simp add: oLog-def monoI*)
done

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

lemma *ordinal-oLog-decreasing*: $oLog\ b\ x \leq x$
apply (*rule-tac x=b and y=1 in linorder-cases*)
apply (*simp add: ordinal-oLog-base-0*)
apply (*simp add: ordinal-oLog-base-1*)
apply (*case-tac x = 0*)
apply (*simp add: ordinal-oLog-0*)
apply (*simp add: oLog-def*)
apply (*simp add: normal.oInv-decreasing[OF normal-exp] oSuc-leI*)
done

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

lemma *oFix-fixed*:
 $\llbracket continuous\ F; a \leq F\ a \rrbracket \implies F\ (oFix\ F\ a) = oFix\ F\ a$
apply (*unfold oFix-def*)

```

apply (simp only: continuousD)
apply (rule order-antisym)
apply (rule oLimit-leI, clarify)
apply (rule-tac n=Suc n in le-oLimitI, simp)
apply (rule oLimit-leI, clarify)
apply (rule-tac n=n in le-oLimitI)
apply (induct-tac n, simp)
apply (simp add: continuous.monoD)
done

```

```

lemma oFix-least:
 $\llbracket \text{mono } F; F x = x; a \leq x \rrbracket \implies \text{oFix } F a \leq x$ 
apply (unfold oFix-def)
apply (rule oLimit-leI, clarify)
apply (induct-tac n, simp-all)
apply (erule subst)
apply (erule monoD, assumption)
done

```

```

lemma mono-oFix:  $\text{mono } F \implies \text{mono } (\text{oFix } F)$ 
apply (rule monoI, unfold oFix-def)
apply (subgoal-tac  $\forall n. \text{iter } n F x \leq \text{iter } n F y$ )
apply (rule oLimit-leI, clarify)
apply (rule-tac n=n in le-oLimitI, erule spec)
apply (rule allI, induct-tac n)
apply simp
apply (simp add: monoD)
done

```

```

lemma less-oFixD:
 $\llbracket x < \text{oFix } F a; \text{mono } F; F x = x \rrbracket \implies x < a$ 
apply (simp add: linorder-not-le[symmetric])
apply (erule contrapos-nn)
by (rule oFix-least)

```

```

lemma less-oFixI:  $a < F a \implies a < \text{oFix } F a$ 
apply (unfold oFix-def)
apply (erule order-less-le-trans)
apply (rule-tac n=1 in le-oLimitI)
apply simp
done

```

```

lemma le-oFix:  $a \leq \text{oFix } F a$ 
apply (unfold oFix-def)
apply (rule-tac n=0 in le-oLimitI)
apply simp
done

```

```

lemma le-oFix1:  $F a \leq \text{oFix } F a$ 

```

```

apply (unfold oFix-def)
apply (rule-tac n=1 in le-oLimitI)
apply simp
done

```

```

lemma less-oFix-0D:
[[x < oFix F 0; mono F]] ==> x < F x
apply (unfold oFix-def, drule less-oLimitD, clarify)
apply (erule-tac P=x < iter n F 0 in rev-mp)
apply (induct-tac n, auto simp add: linorder-not-less)
apply (erule order-less-le-trans)
apply (erule monoD, assumption)
done

```

```

lemma zero-less-oFix-eq: (0 < oFix F 0) = (0 < F 0)
apply (safe)
apply (erule contrapos-pp)
apply (simp only: linorder-not-less oFix-def)
apply (rule oLimit-leI[rule-format])
apply (induct-tac n, simp, simp)
apply (erule less-oFixI)
done

```

```

lemma oFix-eq-self: F a = a ==> oFix F a = a
apply (unfold oFix-def)
apply (subgoal-tac  $\forall n$ . iter n F a = a, simp)
apply (rule allI, induct-tac n, simp-all)
done

```

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**
 $oDeriv F = ordinal-rec (oFix F 0) (\lambda p x. oFix F (oSuc x))$

```

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))
apply (unfold oDeriv-def)
apply (rule ordinal-rec-oLimit, clarify)
apply (rule order-trans[OF order-less-imp-le[OF less-oSuc]])

```

apply (*rule le-oFix*)
done

lemma *oDeriv-fixed*:
 $normal\ F \implies F\ (oDeriv\ F\ n) = oDeriv\ F\ n$
apply (*rule-tac a=n in oLimit-induct, simp-all*)
apply (*rule oFix-fixed*)
apply (*erule normal.continuous*)
apply *simp*
apply (*rule oFix-fixed*)
apply (*erule normal.continuous*)
apply (*erule normal.increasing*)
apply (*simp add: normal.oLimit*)
done

lemma *oDeriv-fixedD*:
 $\llbracket oDeriv\ F\ x = x; normal\ F \rrbracket \implies F\ x = x$
by (*erule subst, erule oDeriv-fixed*)

lemma *normal-oDeriv*:
 $normal\ (oDeriv\ F)$
apply (*rule normalI, simp-all*)
apply (*rule order-less-le-trans[OF less-oSuc]*)
apply (*rule le-oFix*)
done

lemma *oDeriv-increasing*:
 $continuous\ F \implies F\ x \leq oDeriv\ F\ x$
apply (*rule-tac a=x in oLimit-induct*)
apply (*simp add: le-oFix1*)
apply *simp*
apply (*rule order-trans[OF - le-oFix1]*)
apply (*erule continuous.monoD*)
apply *simp*
apply (*rule normal.increasing*)
apply (*rule normal-oDeriv*)
apply (*simp add: continuousD*)
apply (*rule oLimit-leI[rule-format]*)
apply (*rule-tac n=n in le-oLimitI*)
apply (*erule spec*)
done

lemma *oDeriv-total*:
 $\llbracket normal\ F; F\ x = x \rrbracket \implies \exists n. x = oDeriv\ F\ n$
apply (*subgoal-tac $\exists n. oDeriv\ F\ n \leq x \wedge x < oDeriv\ F\ (oSuc\ n)$*)
apply *clarsimp*
apply (*drule less-oFixD*)
apply (*erule normal.mono*)
apply *assumption*

```

apply (rule-tac  $x=n$  in  $exI$ , simp add: less-oSuc-eq-le)
apply (rule normal.oInv-ex[OF normal-oDeriv])
apply (simp add: oFix-least normal.mono)
done

```

```

lemma range-oDeriv:
normal  $F \implies \text{range } (oDeriv\ F) = \{x. F\ x = x\}$ 
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 (rule strict-mono-natI, simp)

```

```

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 $x * \text{ordinal-of-nat } y = \text{ordinal-of-nat } (x * y)$
by (*induct* y) (*simp-all* *add*: *add-commute*)

lemma *ordinal-of-nat-exp* [*simp*]:
ordinal-of-nat $x ** \text{ordinal-of-nat } y = \text{ordinal-of-nat } (x \wedge y)$
by (*induct* y , *cases* x) (*simp-all* *add*: *mult-commute*)

lemma *oSuc-plus-ordinal-of-nat*:
oSuc $x + \text{ordinal-of-nat } n = \text{ordinal-of-nat } (x + \text{ordinal-of-nat } n)$
by (*induct* n) *simp-all*

lemma *less-ordinal-of-nat*:
 $(x < \text{ordinal-of-nat } n) = (\exists m. x = \text{ordinal-of-nat } m \wedge m < n)$
apply (*induct* n)
apply *simp*
apply (*safe*, *simp-all* *del*: *ordinal-of-nat.simps*)
apply (*auto* *elim*: *less-oSucE*)
done

lemma *le-ordinal-of-nat*:
 $(x \leq \text{ordinal-of-nat } n) = (\exists m. x = \text{ordinal-of-nat } m \wedge m \leq n)$
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*: $x < \omega \implies \exists n. x = \text{ordinal-of-nat } n$
apply (*unfold* *omega-def*)
apply (*drule* *less-oLimitD*)
apply (*clarsimp* *simp* *add*: *less-ordinal-of-nat*)
done

lemma *omega-leI*: $\forall n. \text{ordinal-of-nat } n \leq x \implies \omega \leq x$
by (*unfold* *omega-def*, *erule* *oLimit-leI*)

lemma *nat-le-omega* [*simp*]: *ordinal-of-nat* $n \leq \omega$
by (*unfold* *omega-def*, *rule* *le-oLimit*)

lemma *nat-less-omega* [*simp*]: *ordinal-of-nat* $n < \omega$
apply (*rule-tac* $y = \text{ordinal-of-nat } (\text{Suc } n)$ **in** *order-less-le-trans*, *simp*)
apply (*rule* *nat-le-omega*)
done

lemma *zero-less-omega* [*simp*]: $0 < \omega$
by (*cut-tac* $n=0$ **in** *nat-less-omega*, *simp*)

```

lemma limit-ordinal-omega: limit-ordinal  $\omega$ 
  apply (rule limit-ordinalI[rule-format], simp)
  apply (drule less-omegaD, clarify)
  apply (subgoal-tac ordinal-of-nat (Suc  $n$ )  $< \omega$ , simp)
  apply (simp only: nat-less-omega)
done

lemma Least-limit-ordinal: (LEAST  $x$ . limit-ordinal  $x$ ) =  $\omega$ 
  apply (rule Least-equality)
  apply (rule limit-ordinal-omega)
  apply (erule contrapos-pp)
  apply (simp add: linorder-not-le)
  apply (drule less-omegaD, erule exE)
  apply (simp add: not-limit-ordinal-nat)
done

lemma range f = range ordinal-of-nat  $\implies$  oLimit f =  $\omega$ 
  apply (rule order-antisym)
  apply (rule oLimit-leI, clarify)
  apply (drule equalityD1)
  apply (drule-tac  $c=f\ n$  in subsetD, simp)
  apply clarsimp
  apply (rule omega-leI, clarify)
  apply (drule equalityD2)
  apply (drule-tac  $c=ordinal-of-nat\ n$  in subsetD, simp)
  apply clarsimp
done

### 8.3 Arithmetic properties of $\omega$

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

lemma oSuc-plus-omega [simp]: oSuc  $x + \omega = x + \omega$ 
  apply (simp add: omega-def)
  apply (rule oLimit-eqI)
  apply (rule-tac  $x=Suc\ n$  in exI)
  apply (simp add: oSuc-plus-ordinal-of-nat)
  apply (rule-tac  $x=n$  in exI)
  apply (simp add: oSuc-plus-ordinal-of-nat order-less-imp-le)
done

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

lemma ordinal-of-nat-times-omega [simp]:
   $0 < k \implies$  ordinal-of-nat  $k * \omega = \omega$ 
  apply (simp add: omega-def)

```

```

apply (rule oLimit-eqI)
apply (rule-tac exI, rule order-refl)
apply (rule-tac x=n in exI, simp)
done

```

```

lemma ordinal-plus-times-omega:  $x + x * \omega = x * \omega$ 
apply (subgoal-tac  $x + x * \omega = x * (1 + \omega)$ , simp)
apply (simp del: oSuc-plus-omega add: ordinal-times-distrib)
done

```

```

lemma ordinal-plus-absorb:  $x * \omega \leq y \implies x + y = y$ 
apply (drule ordinal-plus-minus2)
apply (erule subst)
apply (simp only: ordinal-plus-assoc[symmetric] ordinal-plus-times-omega)
done

```

```

lemma ordinal-less-plusL:  $y < x * \omega \implies y < x + y$ 
apply (case-tac  $x = 0$ , simp-all)
apply (drule ordinal-div-less)
apply (drule less-omegaD, clarify)
apply (rule-tac  $y=x * (1 + \text{ordinal-of-nat } n)$  in order-less-le-trans)
apply (simp add: oSuc-plus-ordinal-of-nat)
apply (erule subst)
apply (erule ordinal-less-times-div-plus)
apply (simp add: ordinal-times-distrib)
apply (erule subst)
apply (rule ordinal-times-div-le)
done

```

```

lemma ordinal-plus-absorb-iff:  $(x + y = y) = (x * \omega \leq y)$ 
apply safe
apply (rule ccontr, simp add: linorder-not-le)
apply (drule ordinal-less-plusL, simp)
apply (erule ordinal-plus-absorb)
done

```

```

lemma ordinal-less-plusL-iff:  $(y < x + y) = (y < x * \omega)$ 
apply safe
apply (rule ccontr, simp add: linorder-not-less)
apply (drule ordinal-plus-absorb, simp)
apply (erule ordinal-less-plusL)
done

```

8.4 Additive principal ordinals

```

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

```

```

lemma (in additive-principal) sum-less:
   $\llbracket x < a; y < a \rrbracket \implies x + y < a$ 
  by (drule sum-eq, erule subst, simp)

lemma (in additive-principal) times-nat-less:
   $x < a \implies x * \text{ordinal-of-nat } n < a$ 
  apply (induct-tac n)
  apply (simp add: not-0)
  apply (simp add: sum-less)
  done

lemma not-additive-principal-0:  $\neg \text{additive-principal } 0$ 
  by (clarify, drule additive-principal.not-0, simp)

lemma additive-principal-oSuc:
  additive-principal (oSuc a) = (a = 0)
  apply safe
  apply (rule ccontr, simp)
  apply (subgoal-tac a + oSuc 0 < oSuc a, simp)
  apply (erule additive-principal.sum-less, simp-all)
  apply (simp add: additive-principal-def)
  done

lemma additive-principal-intro2 [rule-format]:
  assumes not-0:  $0 < a$ 
  shows  $(\forall x < a. \forall y < a. x + y < a) \longrightarrow \text{additive-principal } a$ 
  apply (simp add: additive-principal-def not-0)
  apply (rule-tac a=a in oLimit-induct)
  apply simp
  apply clarsimp
  apply (drule-tac x=x in spec, simp)
  apply (drule-tac x=1 in spec, simp)
  apply (simp add: linorder-not-less)
  apply clarsimp
  apply (rule order-antisym)
  apply (rule oLimit-leI, clarify)
  apply (rule order-less-imp-le)
  apply (simp add: strict-mono-less-oLimit)
  apply (rule oLimit-leI, clarify)
  apply (rule-tac n=n in le-oLimitI)
  apply (rule ordinal-le-plusL)
  done

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

lemma additive-principal-omega: additive-principal  $\omega$ 
  apply (rule additive-principal.intro)

```

```

  apply (rule zero-less-omega)
  apply (drule less-omegaD, clarify)
  apply (rule ordinal-of-nat-plus-omega)
done

```

```

lemma additive-principal-times-omega:
0 < x  $\implies$  additive-principal (x *  $\omega$ )
  apply (rule additive-principal.intro)
  apply simp
  apply (simp add: omega-def)
  apply (drule less-oLimitD, clarify, rename-tac k)
  apply (drule-tac x=b in order-less-imp-le)
  apply (rule oLimit-eqI)
  apply (rule-tac x=k + n in exI)
  apply (erule order-trans[OF ordinal-plus-monoL])
  apply (simp add: ordinal-times-distrib[symmetric])
  apply (rule-tac x=n in exI, simp)
done

```

```

lemma additive-principal-oLimit:
 $\forall n.$  additive-principal (f n)  $\implies$  additive-principal (oLimit f)
  apply (rule additive-principal.intro)
  apply (rule-tac n=0 in less-oLimitI)
  apply (simp add: additive-principal.not-0)
  apply simp
  apply (drule less-oLimitD, clarify, rename-tac k)
  apply (rule oLimit-eqI)
  apply (rule-tac x=f n and y = f k in linorder-le-cases)
  apply (rule-tac x=k in exI)
  apply (rule-tac y=b + f k in order-trans, simp)
  apply (simp add: additive-principal.sum-eq)
  apply (rule-tac x=n in exI)
  apply (drule order-less-le-trans, assumption)
  apply (simp add: additive-principal.sum-eq)
  apply (rule-tac x=n in exI, simp)
done

```

```

lemma additive-principal-omega-exp: additive-principal ( $\omega ** x$ )
  apply (rule-tac a=x in oLimit-induct)
  apply (simp add: additive-principal-1)
  apply (simp add: additive-principal-times-omega)
  apply (simp add: additive-principal-oLimit)
done

```

```

lemma (in additive-principal) omega-exp:  $\exists x.$  a =  $\omega ** x$ 
  apply (subgoal-tac  $\exists x.$   $\omega ** x \leq a \wedge a < \omega ** (oSuc x)$ )
  prefer 2
  apply (rule normal.oInv-ex)
  apply (rule normal-exp, simp)

```

```

apply (simp add: oSuc-le-eq-less not-0)
apply (auto simp add: order-le-less)
apply (subgoal-tac  $a < a$ , simp)
apply (rule order-less-trans)
apply (rule-tac  $y = \omega ** x$  in ordinal-less-times-div-plus)
apply simp
apply (drule ordinal-div-less)
apply (drule less-omegaD, clarify)
apply (drule-tac  $n = \text{Suc } n$  in times-nat-less)
apply simp
done

```

```

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

```

```

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

```

```

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

```

8.5 Cantor normal form

```

lemma cnf-lemma:  $x > 0 \implies x - \omega ** oLog \omega x < x$ 
apply (subst ordinal-minus-less-eq)
apply (erule ordinal-exp-oLog-le, simp)
apply (rule ordinal-less-plusL)
apply (rule ordinal-less-exp-oLog, simp)
done

```

```

primrec from-cnf where
  from-cnf [] = 0
  | from-cnf (x # xs) =  $\omega ** x + \text{from-cnf } xs$ 

```

```

function to-cnf where
  [simp del]: to-cnf  $x = (\text{if } x = 0 \text{ then } [] \text{ else }
    oLog \omega x \# \text{to-cnf } (x - \omega ** oLog \omega x))$ 
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:

```

$0 < x \implies \text{to-cnf } x = \text{oLog } \omega \ x \ \# \ \text{to-cnf } (x - \omega \ ** \ \text{oLog } \omega \ x)$
by (*simp add: to-cnf.simps[of x]*)

lemma *to-cnf-eq-Cons*: $\text{to-cnf } x = a \ \# \ \text{list} \implies a = \text{oLog } \omega \ x$
by (*case-tac x = 0, simp, simp add: to-cnf-not-0*)

lemma *to-cnf-inverse*: $\text{from-cnf } (\text{to-cnf } x) = x$
apply (*rule wf-induct[OF wf], simp*)
apply (*case-tac x = 0, simp-all*)
apply (*simp add: to-cnf-not-0*)
apply (*simp add: cnf-lemma*)
apply (*rule ordinal-plus-minus2*)
apply (*erule ordinal-exp-oLog-le, simp*)
done

primrec *normalize-cnf* **where**
normalize-cnf-Nil: $\text{normalize-cnf } [] = []$
| *normalize-cnf-Cons*: $\text{normalize-cnf } (x \ \# \ xs) =$
(*case xs of [] \Rightarrow [x] | y # ys \Rightarrow*
(*if x < y then [] else [x]*) @ *normalize-cnf xs*)

lemma *from-cnf-normalize-cnf*: $\text{from-cnf } (\text{normalize-cnf } xs) = \text{from-cnf } xs$
apply (*induct-tac xs, simp-all*)
apply (*case-tac list, simp, clarsimp simp del: normalize-cnf-Cons*)
apply (*simp add: ordinal-plus-assoc[symmetric] absorb-omega-exp2*)
done

lemma *normalize-cnf-to-cnf*: $\text{normalize-cnf } (\text{to-cnf } x) = \text{to-cnf } x$
apply (*rule-tac a=x in wf-induct[OF wf], simp*)
apply (*case-tac x = 0, simp-all*)
apply (*drule spec, drule mp, erule cnf-lemma*)
apply (*simp add: to-cnf-not-0*)
apply (*case-tac to-cnf (x - $\omega \ ** \ \text{oLog } \omega \ x$), simp-all*)
apply (*drule to-cnf-eq-Cons, simp add: linorder-not-less*)
apply (*rule ordinal-oLog-monoR*)
apply (*rule order-less-imp-le*)
apply (*erule cnf-lemma*)
done

alternate form of CNF

lemma *cnf2-lemma*:
 $0 < x \implies x \ \text{mod } \omega \ ** \ \text{oLog } \omega \ x < x$
apply (*rule order-less-le-trans*)
apply (*rule ordinal-mod-less, simp*)
apply (*erule ordinal-exp-oLog-le, simp*)
done

primrec *from-cnf2* **where**
from-cnf2 [] = 0

| $\text{from-cnf2 } (x \# xs) = \omega ** \text{fst } x * \text{ordinal-of-nat } (\text{snd } x) + \text{from-cnf2 } xs$

function *to-cnf2* **where**

[*simp del*]: $\text{to-cnf2 } x = (\text{if } x = 0 \text{ then } [] \text{ else } (\text{oLog } \omega x, \text{inv ordinal-of-nat } (x \text{ div } (\omega ** \text{oLog } \omega x))) \# \text{to-cnf2 } (x \text{ mod } (\omega ** \text{oLog } \omega x)))$

by *pat-completeness auto*

termination by (*relation* $\{(x,y). x < y\}$)

(*simp-all add: wf cnf2-lemma*)

lemma *to-cnf2-0* [*simp*]: $\text{to-cnf2 } 0 = []$

by (*simp add: to-cnf2.simps*)

lemma *to-cnf2-not-0*:

$0 < x \implies \text{to-cnf2 } x =$

($\text{oLog } \omega x, \text{inv ordinal-of-nat } (x \text{ div } (\omega ** \text{oLog } \omega x))) \# \text{to-cnf2 } (x \text{ mod } (\omega ** \text{oLog } \omega x))$)

by (*simp add: to-cnf2.simps[of x]*)

lemma *to-cnf2-eq-Cons*: $\text{to-cnf2 } x = (a,b) \# \text{list} \implies a = \text{oLog } \omega x$

by (*case-tac x = 0, simp, simp add: to-cnf2-not-0*)

lemma *ordinal-of-nat-of-ordinal*:

$x < \omega \implies \text{ordinal-of-nat } (\text{inv ordinal-of-nat } x) = x$

apply (*rule f-inv-into-f*)

apply (*simp add: image-def*)

apply (*erule less-omegaD*)

done

lemma *to-cnf2-inverse*: $\text{from-cnf2 } (\text{to-cnf2 } x) = x$

apply (*rule wf-induct[OF wf], simp*)

apply (*case-tac x = 0, simp-all*)

apply (*simp add: to-cnf2-not-0*)

apply (*simp add: cnf2-lemma*)

apply (*drule-tac x=x mod $\omega ** \text{oLog } \omega x$ in spec*)

apply (*simp add: cnf2-lemma*)

apply (*subst ordinal-of-nat-of-ordinal*)

apply (*rule ordinal-div-less*)

apply (*rule ordinal-less-exp-oLog, simp*)

apply (*rule ordinal-div-plus-mod*)

done

primrec *is-normalized2* **where**

is-normalized2-Nil: $\text{is-normalized2 } [] = \text{True}$

| *is-normalized2-Cons*: $\text{is-normalized2 } (x \# xs) =$

($\text{case } xs \text{ of } [] \Rightarrow \text{True} \mid y \# ys \Rightarrow \text{fst } y < \text{fst } x \wedge \text{is-normalized2 } ys$)

lemma *is-normalized2-to-cnf2*: $\text{is-normalized2 } (\text{to-cnf2 } x)$

```

apply (rule-tac a=x in wf-induct[OF wf], simp)
apply (case-tac x = 0, simp-all)
apply (drule spec, drule mp, erule cnf2-lemma)
apply (simp add: to-cnf2-not-0)
apply (case-tac x mod  $\omega$  ** oLog  $\omega$  x = 0, simp-all)
apply (case-tac to-cnf2 (x mod  $\omega$  ** oLog  $\omega$  x), simp-all)
apply (case-tac a, simp)
apply (drule to-cnf2-eq-Cons, simp)
apply (erule ordinal-oLog-less, simp)
apply (rule ordinal-mod-less, simp)
done

```

8.6 Epsilon 0

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

lemma *less-omega-exp*: $x < \varepsilon_0 \implies x < \omega ** x$
apply (unfold *epsilon0-def*)
apply (erule *less-oFix-0D*)
apply (rule *continuous.mono*)
apply (rule *continuous-exp*)
apply (rule *zero-less-omega*)
done

lemma *omega-exp-epsilon0*: $\omega ** \varepsilon_0 = \varepsilon_0$
apply (unfold *epsilon0-def*)
apply (rule *oFix-fixed*)
apply (rule *continuous-exp*)
apply (rule *zero-less-omega*)
apply *simp*
done

lemma *oLog-omega-less*: $\llbracket 0 < x; x < \varepsilon_0 \rrbracket \implies \text{oLog } \omega x < x$
apply (erule *ordinal-oLog-less*)
apply *simp*
apply (erule *less-omega-exp*)
done

end

9 Veblen Hierarchies

```

theory OrdinalVeblen
imports OrdinalOmega
begin

```

9.1 Closed, unbounded sets

locale *normal-set* =
fixes $A :: \text{ordinal set}$

assumes *closed*: $\bigwedge g. \forall n. g\ n \in A \implies oLimit\ g \in A$
and *unbounded*: $\bigwedge x. \exists y \in A. x < y$

lemma (**in** *normal-set*) *less-next*: $x < (LEAST\ z. z \in A \wedge x < z)$
apply (*rule LeastI2-ex*)
apply (*fold Bex-def, rule unbounded*)
apply (*erule conjunct2*)
done

lemma (**in** *normal-set*) *mem-next*: $(LEAST\ z. z \in A \wedge x < z) \in A$
apply (*rule LeastI2-ex*)
apply (*fold Bex-def, rule unbounded*)
apply (*erule conjunct1*)
done

lemma (**in** *normal*) *normal-set-range*: *normal-set* (*range F*)
apply (*rule normal-set.intro*)
apply (*simp add: image-def*)
apply (*rule-tac x=oLimit* ($\lambda n. LEAST\ x. g\ n = F\ x$) **in** *exI*)
apply (*simp only: oLimit*)
apply (*rule-tac f=oLimit in arg-cong*)
apply (*rule ext*)
apply (*rule LeastI-ex*)
apply (*erule spec*)
apply (*rule-tac x=F* (*oSuc x*) **in** *bexI*)
apply (*rule order-le-less-trans* [*OF increasing*])
apply (*simp add: cancel-less*)
apply (*rule rangeI*)
done

lemma *oLimit-mem-INTER*:
 $\llbracket \forall n. \text{normal-set } (A\ n); \forall n. A\ (Suc\ n) \subseteq A\ n; \forall n. f\ n \in A\ n; \text{mono } f \rrbracket$
 $\implies oLimit\ f \in (\bigcap n. A\ n)$
apply (*clarsimp, rename-tac k*)
apply (*subgoal-tac oLimit* ($\lambda n. f\ (n + k) \in A\ k$)
apply (*simp add: oLimit-shift-mono*)
apply (*rule normal-set.closed* [*rule-format*], *erule spec*)
apply (*rule-tac A=A* ($n + k$) **in** *subsetD*)
apply (*induct-tac n, simp, rename-tac m*)
apply (*rule-tac B=A* ($m + k$) **in** *subset-trans, simp, simp*)
apply (*erule spec*)
done

lemma *normal-set-INTER*:
 $\llbracket \forall n. \text{normal-set } (A\ n); \forall n. A\ (Suc\ n) \subseteq A\ n \rrbracket \implies \text{normal-set } (\bigcap n. A\ n)$

```

apply (rule normal-set.intro)
apply (clarsimp simp add: normal-set.closed)
apply (rule-tac x=oLimit ( $\lambda n. LEAST y. y \in A \ n \wedge x < y$ ) in bexI)
apply (rule-tac y=LEAST  $y. y \in A \ n \wedge x < y$  in order-less-le-trans)
apply (simp only: normal-set.less-next)
apply (rule le-oLimit)
apply (rule oLimit-mem-INTER, assumption+)
apply (simp add: normal-set.mem-next)
apply (rule mono-natI)
apply (rule Least-le)
apply (rule conjI)
apply (rule subsetD, erule spec)
apply (simp only: normal-set.mem-next)
apply (simp only: normal-set.less-next)
done

```

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)  $\Rightarrow$  (ordinal  $\Rightarrow$  ordinal) where
ordering A = ordinal-rec (LEAST  $z. z \in A$ ) ( $\lambda p x. LEAST z. z \in A \wedge x < z$ )

```

lemma ordering-0:

```

ordering A 0 = (LEAST  $z. z \in A$ )
by (simp add: ordering-def)

```

lemma ordering-oSuc:

```

ordering A (oSuc x) = (LEAST  $z. z \in A \wedge ordering A \ x < z$ )
by (simp add: ordering-def)

```

lemma (**in** normal-set) normal-ordering: normal (ordering A)

```

apply (unfold ordering-def)
apply (rule normal-ordinal-rec [rule-format])
apply (rule less-next)
done

```

lemma (**in** normal-set) ordering-oLimit:

```

ordering A (oLimit f) = oLimit ( $\lambda n. ordering A \ (f \ n)$ )
apply (rule normal.oLimit)
apply (rule normal-ordering)
done

```

lemma (**in** normal) ordering-range: ordering (range F) = F

```

apply (rule ext, rule-tac a=x in oLimit-induct)
apply (simp add: ordering-0)
apply (rule Least-equality)
apply (rule rangeI)

```

```

  apply (clarsimp simp add: cancel-le)
  apply (simp add: ordering-oSuc)
  apply (rule Least-equality)
  apply (simp add: cancel-less)
  apply (clarsimp simp add: cancel-le cancel-less oSuc-leI)
  apply (subst normal-set.ordering-oLimit)
  apply (rule normal-set-range)
  apply (simp add: oLimit)
done

```

```

lemma (in normal-set) ordering-mem: ordering A x ∈ A
  apply (rule-tac a=x in oLimit-induct)
  apply (subst ordering-0)
  apply (rule LeastI-ex)
  apply (cut-tac unbounded, force)
  apply (subst ordering-oSuc)
  apply (rule mem-next)
  apply (subst ordering-oLimit)
  apply (erule closed)
done

```

```

lemma (in normal-set) range-ordering-lemma:
  ∀ y. y ∈ A ⟶ y < ordering A x ⟶ y ∈ range (ordering A)
  apply (simp add: image-def)
  apply (rule-tac a=x in oLimit-induct, safe)
  apply (simp add: ordering-0)
  apply (drule not-less-Least, simp)
  apply (simp add: ordering-oSuc)
  apply (drule not-less-Least, simp)
  apply (force simp add: linorder-not-less order-le-less)
  apply (simp add: ordering-oLimit)
  apply (drule less-oLimitD, clarsimp)
done

```

```

lemma (in normal-set) range-ordering: range (ordering A) = A
  apply (safe intro!: ordering-mem)
  apply (erule-tac x=oSuc x in range-ordering-lemma[rule-format])
  apply (rule order-less-le-trans[OF less-oSuc])
  apply (rule normal.increasing[OF normal-ordering])
done

```

```

lemma ordering-INTER-0:
  [∀ n. normal-set (A n); ∀ n. A (Suc n) ⊆ A n]
  ⟹ ordering (⋂ n. A n) 0 = oLimit (λn. ordering (A n) 0)
  apply (subst ordering-0)
  apply (rule Least-equality)
  apply (rule oLimit-mem-INTER, assumption+)
  apply (simp add: normal-set.ordering-mem)
  apply (rule mono-natI)

```

```

apply (simp add: ordering-0)
apply (rule Least-le)
apply (rule subsetD, erule spec)
apply (drule-tac x=Suc n in spec)
apply (drule normal-set.unbounded, clarify)
apply (erule LeastI)
apply (rule oLimit-leI[rule-format])
apply (simp add: ordering-0)
apply (rule Least-le)
apply (erule spec)
done

```

9.3 Critical ordinals

definition

critical-set :: ordinal set \Rightarrow ordinal \Rightarrow ordinal set **where**
critical-set A =
 ordinal-rec0 A ($\lambda p x. x \cap \text{range } (oDeriv \text{ (ordering } x))$) ($\lambda f. \bigcap n. f n$)

lemma *critical-set-0*:

critical-set A 0 = A
by (unfold *critical-set-def*, rule *ordinal-rec0-0*)

lemma *critical-set-oSuc-lemma*:

critical-set A (oSuc n) =
critical-set A n \cap range (oDeriv (ordering (critical-set A n)))
by (unfold *critical-set-def*, rule *ordinal-rec0-oSuc*)

lemma *omega-complete-INTER*:

omega-complete ($\lambda x y. y \subseteq x$) (INTER UNIV)
apply (rule *omega-complete.intro*)
apply (rule *porder.flip*)
apply (rule *porder-order*)
apply (rule *omega-complete-axioms.intro*)
apply fast
apply fast
done

lemma *critical-set-oLimit*:

critical-set A (oLimit f) = ($\bigcap n. \text{critical-set } A (f n)$)
apply (unfold *critical-set-def*)
apply (rule *omega-complete.ordinal-rec0-oLimit*)
apply (rule *omega-complete-INTER*)
apply fast
done

lemma *critical-set-mono*:

$x \leq y \Rightarrow \text{critical-set } A y \subseteq \text{critical-set } A x$
apply (unfold *critical-set-def*)

```

apply (rule omega-complete.ordinal-rec0-mono
        [OF omega-complete-INTER])
apply fast
apply assumption
done

```

```

lemma (in normal-set) range-oDeriv-subset:
range (oDeriv (ordering A))  $\subseteq$  A
apply (clarsimp, rename-tac x)
apply (cut-tac n=x in oDeriv-fixed[OF normal-ordering])
apply (erule subst)
apply (rule ordering-mem)
done

```

```

lemma normal-set-critical-set:
normal-set A  $\implies$  normal-set (critical-set A x)
apply (rule-tac a=x in oLimit-induct)
apply (simp only: critical-set-0)
apply (simp only: critical-set-oSuc-lemma)
apply (subst Int-absorb1)
apply (erule normal-set.range-oDeriv-subset)
apply (rule normal.normal-set-range)
apply (rule normal-oDeriv)
apply (simp only: critical-set-oLimit)
apply (erule normal-set-INTER)
apply (rule allI, rule critical-set-mono)
apply (simp add: strict-mono-monoD)
done

```

```

lemma critical-set-oSuc:
normal-set A
 $\implies$  critical-set A (oSuc x) = range (oDeriv (ordering (critical-set A x)))
apply (simp only: critical-set-oSuc-lemma)
apply (rule Int-absorb1)
apply (rule normal-set.range-oDeriv-subset)
apply (erule normal-set-critical-set)
done

```

9.4 Veblen hierarchy over a normal function

definition

$oVeblen$:: (ordinal \Rightarrow ordinal) \Rightarrow ordinal \Rightarrow ordinal \Rightarrow ordinal **where**
 $oVeblen$ F = (λx . ordering (critical-set (range F) x))

```

lemma (in normal) oVeblen-0: oVeblen F 0 = F
apply (unfold oVeblen-def)
apply (subst critical-set-0)
apply (rule ordering-range)
done

```

```

lemma (in normal) oVeblen-oSuc:
oVeblen F (oSuc x) = oDeriv (oVeblen F x)
apply (unfold oVeblen-def)
apply (subst critical-set-oSuc)
apply (rule normal-set-range)
apply (rule normal.ordering-range)
apply (rule normal-oDeriv)
done

lemma (in normal) oVeblen-oLimit:
oVeblen F (oLimit f) = ordering ( $\bigcap n.$  range (oVeblen F (f n)))
apply (unfold oVeblen-def)
apply (subst critical-set-oLimit)
apply (cut-tac normal-set-range)
apply (simp add: normal-set.range-ordering[OF normal-set-critical-set])
done

lemma (in normal) normal-oVeblen:
normal (oVeblen F x)
apply (unfold oVeblen-def)
apply (rule normal-set.normal-ordering)
apply (rule normal-set-critical-set)
apply (rule normal-set-range)
done

lemma (in normal) continuous-oVeblen-0:
continuous ( $\lambda x.$  oVeblen F x 0)
apply (rule continuousI)
apply (simp add: oVeblen-def critical-set-oLimit)
apply (rule ordering-INTER-0[rule-format])
apply (rule normal-set-critical-set)
apply (rule normal-set-range)
apply (rule critical-set-mono)
apply (simp add: strict-mono-monoD)
apply (simp only: oVeblen-oSuc)
apply (rule oDeriv-increasing)
apply (rule normal.continuous)
apply (rule normal-oVeblen)
done

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

lemma (in normal) normal-oVeblen-0:
0 < F 0  $\implies$  normal ( $\lambda x.$  oVeblen F x 0)
apply (rule normalI)
apply (rule oVeblen-oLimit-0)

```

```

apply (simp only: oVeblen-oSuc)
apply (subst oDeriv-fixed[OF normal-oVeblen, symmetric])
apply (rule normal.strict-monoD[OF normal-oVeblen])
apply (simp add: zero-less-oFix-eq)
apply (erule order-less-le-trans)
apply (subgoal-tac oVeblen F 0 0 ≤ oVeblen F x 0)
  apply (simp add: oVeblen-0)
apply (rule continuous.monoD[OF - ordinal-0-le])
apply (rule continuous-oVeblen-0)
done

```

```

lemma (in normal) range-oVeblen:
range (oVeblen F x) = critical-set (range F) x
  apply (unfold oVeblen-def)
  apply (rule normal-set.range-ordering)
  apply (rule normal-set-critical-set)
  apply (rule normal-set-range)
done

```

```

lemma (in normal) range-oVeblen-subset:
x ≤ y ⇒ range (oVeblen F y) ⊆ range (oVeblen F x)
  apply (simp only: range-oVeblen)
  apply (erule critical-set-mono)
done

```

```

lemma (in normal) oVeblen-fixed:
 $\forall x < y. \forall a. oVeblen F x (oVeblen F y a) = oVeblen F y a$ 
  apply (rule-tac a=y in oLimit-induct)
  apply simp
  apply (clarsimp simp only: oVeblen-oSuc)
  apply (erule less-oSucE)
  apply (drule spec, drule mp, assumption)
  apply (drule-tac x=oDeriv (oVeblen F x) a in spec)
  apply (simp add: oDeriv-fixed normal-oVeblen)
  apply simp
  apply (rule oDeriv-fixed)
  apply (rule normal-oVeblen)
  apply clarsimp
  apply (erule less-oLimitE)
  apply (drule spec, drule spec, drule mp, assumption)
  apply (subgoal-tac oVeblen F (oLimit f) a ∈ range (oVeblen F (f n)))
  apply clarsimp
  apply (rule-tac A=range (oVeblen F (oLimit f)) in subsetD)
  apply (rule range-oVeblen-subset)
  apply (rule le-oLimit)
  apply (rule rangeI)
done

```

```

lemma (in normal) critical-set-fixed:

```

```

0 < z ==> range (oVeblen F z) = {x. ∀ y < z. oVeblen F y x = x}
apply (rule equalityI)
apply (clarsimp simp add: oVeblen-fixed)
apply (erule rev-mp)
apply (rule-tac a=z in oLimit-induct)
apply simp
apply clarsimp
apply (simp add: oVeblen-oSuc range-oDeriv normal-oVeblen)
apply clarsimp
apply (simp add: range-oVeblen)
apply (clarsimp simp add: critical-set-oLimit)
apply (rule-tac A=critical-set (range F) (f (Suc n)) in subsetD)
apply (rule critical-set-mono)
apply (simp add: strict-mono-monoD)
apply (drule-tac x=Suc n in spec, drule mp)
apply (rule-tac y=f n in order-le-less-trans, simp)
apply (erule OrdinalInduct.strict-monoD, simp)
apply (erule subsetD, clarsimp)
apply (drule spec, erule mp)
apply (erule order-less-le-trans)
apply (rule le-oLimit)
done

```

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

```

lemma oDeriv-id: oDeriv id = id
apply (rule ext, rule-tac a=x in oLimit-induct)
apply (simp add: oFix-eq-self)
apply (simp add: oFix-eq-self)
apply simp
done

```

```

lemma oFix-plus: oFix ( $\lambda x. a + x$ ) 0 = a *  $\omega$ 
apply (simp add: oFix-def omega-def)
apply (rule-tac f=oLimit in arg-cong)
apply (rule ext, induct-tac n, simp)
apply (simp, rename-tac n)
apply (induct-tac n, simp)
apply (simp add: ordinal-plus-assoc[symmetric])
done

```

```

lemma oDeriv-plus: oDeriv (op + a) = (op + (a *  $\omega$ ))
apply (rule ext, rule-tac a=x in oLimit-induct)
apply (simp add: oFix-plus)
apply (simp add: oFix-eq-self
          ordinal-plus-assoc[symmetric]
          ordinal-plus-times-omega)
apply simp
done

```

```

lemma oVeblen-1-plus: oVeblen (op + 1) x = (op + ( $\omega$  ** x))
apply (rule-tac a=x in wf-induct[OF wf], simp)
apply (rule-tac a=x in ordinal-cases)
  apply (simp add: normal.oVeblen-0[OF normal-plus])
  apply (simp add: normal.oVeblen-oSuc[OF normal-plus])
  apply (simp add: oDeriv-plus)
apply clarsimp
apply (rule normal-range-eq)
  apply (rule normal.normal-oVeblen[OF normal-plus])
  apply (rule normal-plus)
apply (subst normal.critical-set-fixed[OF normal-plus])
apply (rule-tac y=f 0 in order-le-less-trans, simp)
apply (simp add: strict-mono-less-oLimit)
apply safe
  apply (simp add: image-def)
  apply (rule exI, rule ordinal-plus-minus2[symmetric])
  apply (rule oLimit-leI[rule-format])
  apply (subgoal-tac  $\omega$  ** f n + x = x, erule subst, simp)
  apply (drule-tac x=f n in spec)
  apply (simp add: strict-mono-less-oLimit)
apply simp
apply (simp only: ordinal-exp-oLimit[symmetric] zero-less-omega)
apply (simp only: ordinal-plus-assoc[symmetric])
apply (simp only: absorb-omega-exp2)
done

end

```