The Hereditarily Finite Sets

Lawrence C. Paulson

March 17, 2025

Abstract

The theory of hereditarily finite sets is formalised, following the development of Świerczkowski [2]. An HF set is a finite collection of other HF sets; they enjoy an induction principle and satisfy all the axioms of ZF set theory apart from the axiom of infinity, which is negated. All constructions that are possible in ZF set theory (Cartesian products, disjoint sums, natural numbers, functions) without using infinite sets are possible here. The definition of addition for the HF sets follows Kirby [1].

This development forms the foundation for the Isabelle proof of Gödel's incompleteness theorems, which has been formalised separately.

Contents

Chapter 1

The Hereditarily Finite Sets

theory *HF*
imports *HOL−Library.Nat-Bijection*
abbrevs <: = ∈
  and ~<: = ∉
begin

From "Finite sets and Gödel's Incompleteness Theorems" by S. Swierczkowski. Thanks for Brian Huffman for this development, up to the cases and induct rules.

1.1 Basic Definitions and Lemmas

typedef *hf* = *UNIV* :: *nat set* ⟨*proof*⟩

definition *hfset* :: *hf* ⇒ *hf set*
  where *hfset a* = *Abs-hf* ‘ *set-decode* (*Rep-hf a*)

definition *HF* :: *hf set* ⇒ *hf*
  where *HF A* = *Abs-hf* (*set-encode* (*Rep-hf* ‘ *A*))

definition *hinsert* :: *hf* ⇒ *hf* ⇒ *hf*
  where *hinsert a b* = *HF* (*insert a* (*hfset b*))

definition *hmem* :: *hf* ⇒ *hf* ⇒ *bool* (**infixl** ‹∈› *50*)
  where *hmem a b* ⟷ *a* ∈ *hfset b*

abbreviation *not-hmem* :: *hf* ⇒ *hf* ⇒ *bool* (**infixl** ‹∉› *50*)
  where *a* ∉ *b* ≡ ¬ *a* ∈ *b*

notation (*ASCII*)
  hmem (**infixl** ‹<:› *50*)

instantiation *hf* :: *zero*
begin

definition *Zero-hf-def*: $0 = HF\ \{\}$
instance ⟨*proof*⟩
end

lemma *Abs-hf-0* [*simp*]: *Abs-hf 0 = 0*
⟨*proof*⟩

HF Set enumerations

abbreviation *inserthf* :: $hf \Rightarrow hf \Rightarrow hf$ (**infixl** ‹◁› *60*)
where $y \triangleleft x \equiv hinsert\ x\ y$

syntax (*ASCII*)
-HFinset :: $args \Rightarrow hf$ (‹{|(-)|}›)
syntax
-HFinset :: $args \Rightarrow hf$ (‹⦃-⦄›)
syntax-consts
-HFinset ⇌ *inserthf*
translations
⦃x, y⦄ ⇌ ⦃y⦄ ◁ x
⦃x⦄ ⇌ $0 \triangleleft x$

lemma *finite-hfset* [*simp*]: *finite* (*hfset a*)
⟨*proof*⟩

lemma *HF-hfset* [*simp*]: *HF* (*hfset a*) = *a*
⟨*proof*⟩

lemma *hfset-HF* [*simp*]: *finite A* $\Longrightarrow$ *hfset* (*HF A*) = *A*
⟨*proof*⟩

lemma *inj-on-HF*: *inj-on HF* (*Collect finite*)
⟨*proof*⟩

lemma *hmem-hempty* [*simp*]: $a \notin 0$
⟨*proof*⟩

lemmas *hemptyE* [*elim*!] = *hmem-hempty* [*THEN notE*]

lemma *hmem-hinsert* [*iff*]:
$hmem\ a\ (c \triangleleft b) \longleftrightarrow a = b \lor a \in c$
⟨*proof*⟩

lemma *hf-ext*: $a = b \longleftrightarrow (\forall x.\ x \in a \longleftrightarrow x \in b)$
⟨*proof*⟩

lemma *finite-cases* [*consumes 1*, *case-names empty insert*]:
$[\![finite\ F;\ F = \{\} \Longrightarrow P;\ \bigwedge A\ x.\ [\![F = insert\ x\ A;\ x \notin A;\ finite\ A]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *hf-cases* [*cases type*: *hf*, *case-names 0 hinsert*]:
  obtains $y = 0 \mid a\ b$ **where** $y = b \triangleleft a$ **and** $a \notin b$
⟨*proof*⟩

lemma *Rep-hf-hinsert*:
  assumes $a \notin b$ **shows** *Rep-hf* (*hinsert a b*) $=$ *2* ^ (*Rep-hf a*) $+$ *Rep-hf b*
⟨*proof*⟩

1.2 Verifying the Axioms of HF

HF1

lemma *hempty-iff*: $z{=}0 \longleftrightarrow (\forall x.\ x \notin z)$
  ⟨*proof*⟩

HF2

lemma *hinsert-iff*: $z = x \triangleleft y \longleftrightarrow (\forall u.\ u \in z \longleftrightarrow u \in x \lor u = y)$
  ⟨*proof*⟩

HF induction

lemma *hf-induct* [*induct type*: *hf*, *case-names 0 hinsert*]:
  assumes [*simp*]: $P\ 0$
  $\bigwedge x\ y.\ [\![P\ x;\ P\ y;\ x \notin y]\!] \Longrightarrow P\ (y \triangleleft x)$
  shows $P\ z$
⟨*proof*⟩

HF3

lemma *hf-induct-ax*: $[\![P\ 0;\ \forall x.\ P\ x \longrightarrow (\forall y.\ P\ y \longrightarrow P\ (x \triangleleft y))]\!] \Longrightarrow P\ x$
  ⟨*proof*⟩

lemma *hf-equalityI* [*intro*]: $(\bigwedge x.\ x \in a \longleftrightarrow x \in b) \Longrightarrow a = b$
  ⟨*proof*⟩

lemma *hinsert-nonempty* [*simp*]: $A \triangleleft a \neq 0$
  ⟨*proof*⟩

lemma *hinsert-commute*: $(z \triangleleft y) \triangleleft x = (z \triangleleft x) \triangleleft y$
  ⟨*proof*⟩

lemma *hmem-HF-iff* [*simp*]: $x \in HF\ A \longleftrightarrow x \in A \land finite\ A$
  ⟨*proof*⟩

1.3 Ordered Pairs, from ZF/ZF.thy

lemma *singleton-eq-iff* [*iff*]: $\{\!|a|\!\} = \{\!|b|\!\} \longleftrightarrow a{=}b$
  ⟨*proof*⟩

lemma *doubleton-eq-iff*: $\{\!|a,b|\!\} = \{\!|c,d|\!\} \longleftrightarrow (a{=}c \land b{=}d) \lor (a{=}d \land b{=}c)$

$\langle proof \rangle$

definition *hpair* :: $hf \Rightarrow hf \Rightarrow hf$
  where $hpair\ a\ b = \{\!|\{\!|a|\!\},\{\!|a,b|\!\}|\!\}$

definition *hfst* :: $hf \Rightarrow hf$
  where $hfst\ p \equiv THE\ x.\ \exists\, y.\ p = hpair\ x\ y$

definition *hsnd* :: $hf \Rightarrow hf$
  where $hsnd\ p \equiv THE\ y.\ \exists\, x.\ p = hpair\ x\ y$

definition *hsplit* :: $[[hf,\ hf] \Rightarrow 'a,\ hf] \Rightarrow 'a{::}\{\}$ — for pattern-matching
  where $hsplit\ c \equiv \lambda p.\ c\ (hfst\ p)\ (hsnd\ p)$

Ordered Pairs, from ZF/ZF.thy

nonterminal *hfs*
syntax (*ASCII*)
  $\textit{-Tuple}$ :: $[hf,\ hfs] \Rightarrow hf$ (‹<(-,/ -)>›)
  $\textit{-hpattern}$ :: $[pttrn,\ patterns] \Rightarrow pttrn$ (‹<-,/ ->›)
syntax
  :: $hf \Rightarrow hfs$ (‹-›)
  $\textit{-Enum}$ :: $[hf,\ hfs] \Rightarrow hfs$ (‹-,/ -›)
  $\textit{-Tuple}$ :: $[hf,\ hfs] \Rightarrow hf$ $(\text{‹}\langle(\text{-,/ -})\rangle\text{›})$
  $\textit{-hpattern}$ :: $[pttrn,\ patterns] \Rightarrow pttrn$ $(\text{‹}\langle\text{-,/ -}\rangle\text{›})$
syntax-consts
  $\textit{-Enum}\ \textit{-Tuple} \rightleftharpoons hpair$ **and**
  $\textit{-hpattern} \rightleftharpoons hsplit$
translations
  $<x,\ y,\ z> \rightleftharpoons <x,\ <y,\ z>>$
  $<x,\ y> \rightleftharpoons CONST\ hpair\ x\ y$
  $<x,\ y,\ z> \rightleftharpoons <x,\ <y,\ z>>$
  $\lambda<x,y,zs>.\ b \rightleftharpoons CONST\ hsplit(\lambda x\ <y,zs>.\ b)$
  $\lambda<x,y>.\ b \rightleftharpoons CONST\ hsplit(\lambda x\ y.\ b)$

lemma *hpair-def'*: $hpair\ a\ b = \{\!|\{\!|a,a|\!\},\{\!|a,b|\!\}|\!\}$
  $\langle proof \rangle$

lemma *hpair-iff* [*simp*]: $hpair\ a\ b = hpair\ a'\ b' \longleftrightarrow a{=}a' \wedge b{=}b'$
  $\langle proof \rangle$

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

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

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

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

1.4 Unions, Comprehensions, Intersections

1.4.1 Unions

Theorem 1.5 (Existence of the union of two sets).

lemma *binary-union*: $\exists z.\ \forall u.\ u \in z \longleftrightarrow u \in x \lor u \in y$
$\langle proof\rangle$

Theorem 1.6 (Existence of the union of a set of sets).

lemma *union-of-set*: $\exists z.\ \forall u.\ u \in z \longleftrightarrow (\exists y.\ y \in x \land u \in y)$
$\langle proof\rangle$

1.4.2 Set comprehensions

Theorem 1.7, comprehension scheme

lemma *comprehension*: $\exists z.\ \forall u.\ u \in z \longleftrightarrow u \in x \land P\ u$
$\langle proof\rangle$

definition $HCollect :: (hf \Rightarrow bool) \Rightarrow hf \Rightarrow hf$ — comprehension
  where $HCollect\ P\ A = (THE\ z.\ \forall u.\ u \in z = (P\ u \land u \in A))$

syntax (*ASCII*)
  $\text{-}HCollect :: idt \Rightarrow hf \Rightarrow bool \Rightarrow hf \quad (\langle(1\{\!|\text{-}\ <:/\ \text{-}./\ \text{-}|\!\})\rangle)$
syntax
  $\text{-}HCollect :: idt \Rightarrow hf \Rightarrow bool \Rightarrow hf \quad (\langle(1\{\!|\text{-}\ \in/\ \text{-}./\ \text{-}|\!\})\rangle)$
syntax-consts
  $\text{-}HCollect \rightleftharpoons HCollect$
translations
  $\{\!|x \in A.\ P|\!\} \rightleftharpoons CONST\ HCollect\ (\lambda x.\ P)\ A$

lemma *HCollect-iff* [*iff*]: $hmem\ x\ (HCollect\ P\ A) \longleftrightarrow P\ x \land x \in A$
  $\langle proof\rangle$

lemma *HCollectI*: $a \in A \Longrightarrow P\ a \Longrightarrow hmem\ a\ \{\!|x \in A.\ P\ x|\!\}$
  $\langle proof\rangle$

lemma *HCollectE*:
  assumes $a \in \{\!|x \in A.\ P\ x|\!\}$ **obtains** $a \in A\ P\ a$
  $\langle proof\rangle$

lemma *HCollect-hempty* [*simp*]: $HCollect\ P\ 0 = 0$
  $\langle proof\rangle$

1.4.3 Union operators

instantiation $hf :: sup$

begin
  definition *sup a b* = (*THE z*. ∀ *u*. *u* ∈ *z* ⟷ *u* ∈ *a* ∨ *u* ∈ *b*)
  instance ⟨*proof*⟩
end

abbreviation *hunion* :: *hf* ⇒ *hf* ⇒ *hf* (**infixl** ‹⊔› *65*) **where**
  hunion ≡ *sup*

lemma *hunion-iff* [*iff*]: *hmem x* (*a* ⊔ *b*) ⟷ *x* ∈ *a* ∨ *x* ∈ *b*
  ⟨*proof*⟩

definition *HUnion* :: *hf* ⇒ *hf*   (‹⨆ -› [*900*] *900*)
  where *HUnion A* = (*THE z*. ∀ *u*. *u* ∈ *z* ⟷ (∃ *y*. *y* ∈ *A* ∧ *u* ∈ *y*))

lemma *HUnion-iff* [*iff*]: *hmem x* (⨆ *A*) ⟷ (∃ *y*. *y* ∈ *A* ∧ *x* ∈ *y*)
  ⟨*proof*⟩

lemma *HUnion-hempty* [*simp*]: ⨆ *0* = *0*
  ⟨*proof*⟩

lemma *HUnion-hinsert* [*simp*]: ⨆(*A* ◁ *a*) = *a* ⊔ ⨆*A*
  ⟨*proof*⟩

lemma *HUnion-hunion* [*simp*]: ⨆(*A* ⊔ *B*) = ⨆*A* ⊔ ⨆*B*
  ⟨*proof*⟩

1.4.4 Definition 1.8, Intersections

instantiation *hf* :: *inf*
begin
  definition *inf a b* = ⦃*x* ∈ *a*. *x* ∈ *b*⦄
  instance ⟨*proof*⟩
end

abbreviation *hinter* :: *hf* ⇒ *hf* ⇒ *hf* (**infixl** ‹⊓› *70*) **where**
  hinter ≡ *inf*

lemma *hinter-iff* [*iff*]: *hmem u* (*x* ⊓ *y*) ⟷ *u* ∈ *x* ∧ *u* ∈ *y*
  ⟨*proof*⟩

definition *HInter* :: *hf* ⇒ *hf*   (‹⨅ -› [*900*] *900*)
  where *HInter*(*A*) = ⦃*x* ∈ *HUnion*(*A*). ∀ *y*. *y* ∈ *A* ⟶ *x* ∈ *y*⦄

lemma *HInter-hempty* [*iff*]: ⨅ *0* = *0*
  ⟨*proof*⟩

lemma *HInter-iff* [*simp*]: *A*≠*0* ⟹ *hmem x* (⨅ *A*) ⟷ (∀ *y*. *y* ∈ *A* ⟶ *x* ∈ *y*)
  ⟨*proof*⟩

lemma *HInter-hinsert* [*simp*]: $A \neq 0 \Longrightarrow \bigsqcap(A \triangleleft a) = a \sqcap \bigsqcap A$
  ⟨*proof*⟩

1.4.5 Set Difference

instantiation *hf* :: *minus*
begin
  definition $A - B = \{\!|x \in A.\ x \notin B|\!\}$
  instance ⟨*proof*⟩
end

lemma *hdiff-iff* [*iff*]: $hmem\ u\ (x - y) \longleftrightarrow u \in x \wedge u \notin y$
  ⟨*proof*⟩

lemma *hdiff-zero* [*simp*]: **fixes** x :: *hf* **shows** $(x - 0) = x$
  ⟨*proof*⟩

lemma *zero-hdiff* [*simp*]: **fixes** x :: *hf* **shows** $(0 - x) = 0$
  ⟨*proof*⟩

lemma *hdiff-insert*: $A - (B \triangleleft a) = A - B - \{\!|a|\!\}$
  ⟨*proof*⟩

lemma *hinsert-hdiff-if*:
  $(A \triangleleft x) - B = (\mathit{if}\ x \in B\ \mathit{then}\ A - B\ \mathit{else}\ (A - B) \triangleleft x)$
  ⟨*proof*⟩

1.5 Replacement

Theorem 1.9 (Replacement Scheme).

lemma *replacement*:
  $(\forall u\ v\ v'.\ u \in x \longrightarrow R\ u\ v \longrightarrow R\ u\ v' \longrightarrow v'=v) \Longrightarrow \exists z.\ \forall v.\ v \in z \longleftrightarrow (\exists u.\ u \in x \wedge R\ u\ v)$
⟨*proof*⟩

lemma *replacement-fun*: $\exists z.\ \forall v.\ v \in z \longleftrightarrow (\exists u.\ u \in x \wedge v = f\ u)$
  ⟨*proof*⟩

definition *PrimReplace* :: $hf \Rightarrow (hf \Rightarrow hf \Rightarrow bool) \Rightarrow hf$
  where $PrimReplace\ A\ R = (THE\ z.\ \forall v.\ v \in z \longleftrightarrow (\exists u.\ u \in A \wedge R\ u\ v))$

definition *Replace* :: $hf \Rightarrow (hf \Rightarrow hf \Rightarrow bool) \Rightarrow hf$
  where $Replace\ A\ R = PrimReplace\ A\ (\lambda x\ y.\ (\exists!z.\ R\ x\ z) \wedge R\ x\ y)$

definition *RepFun* :: $hf \Rightarrow (hf \Rightarrow hf) \Rightarrow hf$
  where $RepFun\ A\ f = Replace\ A\ (\lambda x\ y.\ y = f\ x)$

syntax (*ASCII*)
  -HReplace :: [*pttrn*, *pttrn*, *hf*, *bool*] $\Rightarrow$ *hf* (‹(*1*{|- ./ -<: -, -|})›)
  -HRepFun :: [*hf*, *pttrn*, *hf*] $\Rightarrow$ *hf* (‹(*1*{|- ./ -<: -|})› [*51*,*0*,*51*])
  -HINTER :: [*pttrn*, *hf*, *hf*] $\Rightarrow$ *hf* (‹(*3INT* -<:-./ -)› *10*)
  -HUNION :: [*pttrn*, *hf*, *hf*] $\Rightarrow$ *hf* (‹(*3UN* -<:-./ -)› *10*)
syntax
  -HReplace :: [*pttrn*, *pttrn*, *hf*, *bool*] $\Rightarrow$ *hf* (‹(*1*⦃- ./ - $\in$ -, -⦄)›)
  -HRepFun :: [*hf*, *pttrn*, *hf*] $\Rightarrow$ *hf* (‹(*1*⦃- ./ - $\in$ -⦄)› [*51*,*0*,*51*])
  -HINTER :: [*pttrn*, *hf*, *hf*] $\Rightarrow$ *hf* (‹(*3*$\bigsqcap$-$\in$-./ -)› *10*)
  -HUNION :: [*pttrn*, *hf*, *hf*] $\Rightarrow$ *hf* (‹(*3*$\bigsqcup$-$\in$-./ -)› *10*)
syntax-consts
  -HReplace $\rightleftharpoons$ *Replace* **and**
  -HRepFun $\rightleftharpoons$ *RepFun* **and**
  -HINTER $\rightleftharpoons$ *HInter* **and**
  -HUNION $\rightleftharpoons$ *HUnion*
translations
  ⦃*y*. *x*$\in$*A*, *Q*⦄ $\rightleftharpoons$ *CONST Replace A* (λ*x y*. *Q*)
  ⦃*b*. *x*$\in$*A*⦄ $\rightleftharpoons$ *CONST RepFun A* (λ*x*. *b*)
  $\bigsqcap$*x*$\in$*A*. *B* $\rightleftharpoons$ *CONST HInter*(*CONST RepFun A* (λ*x*. *B*))
  $\bigsqcup$*x*$\in$*A*. *B* $\rightleftharpoons$ *CONST HUnion*(*CONST RepFun A* (λ*x*. *B*))

lemma *PrimReplace-iff*:
  assumes *sv*: $\forall\, u\ v\ v'.\ u \in A \longrightarrow R\ u\ v \longrightarrow R\ u\ v' \longrightarrow v'{=}v$
  shows $v \in (PrimReplace\ A\ R) \longleftrightarrow (\exists\, u.\ u \in A \wedge R\ u\ v)$
  ⟨*proof*⟩

lemma *Replace-iff* [*iff*]:
  $v \in Replace\ A\ R \longleftrightarrow (\exists\, u.\ u \in A \wedge R\ u\ v \wedge (\forall\, y.\ R\ u\ y \longrightarrow y{=}v))$
  ⟨*proof*⟩

lemma *Replace-0* [*simp*]: $Replace\ 0\ R = 0$
  ⟨*proof*⟩

lemma *Replace-hunion* [*simp*]: $Replace\ (A \sqcup B)\ R = Replace\ A\ R\ \sqcup\ Replace\ B$
R
  ⟨*proof*⟩

lemma *Replace-cong* [*cong*]:
  $[\![\ A{=}B;\ \bigwedge x\ y.\ x \in B \Longrightarrow P\ x\ y \longleftrightarrow Q\ x\ y\ ]\!] \Longrightarrow Replace\ A\ P = Replace\ B\ Q$
  ⟨*proof*⟩

lemma *RepFun-iff* [*iff*]: $v \in (RepFun\ A\ f) \longleftrightarrow (\exists\, u.\ u \in A \wedge v = f\ u)$
  ⟨*proof*⟩

lemma *RepFun-cong* [*cong*]:
  $[\![\ A{=}B;\ \bigwedge x.\ x \in B \Longrightarrow f(x){=}g(x)\ ]\!] \Longrightarrow RepFun\ A\ f = RepFun\ B\ g$
⟨*proof*⟩

lemma *triv-RepFun* [*simp*]: $RepFun\ A\ (\lambda x.\ x) = A$

$\langle proof \rangle$

lemma *RepFun-0* [*simp*]: $RepFun\ 0\ f = 0$
$\langle proof \rangle$

lemma *RepFun-hinsert* [*simp*]: $RepFun\ (hinsert\ a\ b)\ f = hinsert\ (f\ a)\ (RepFun\ b\ f)$
$\langle proof \rangle$

lemma *RepFun-hunion* [*simp*]:
$RepFun\ (A \sqcup B)\ f = RepFun\ A\ f \sqcup RepFun\ B\ f$
$\langle proof \rangle$

lemma *HF-HUnion*: $[\![finite\ A;\ \bigwedge x.\ x{\in}A \Longrightarrow finite\ (B\ x)]\!] \Longrightarrow HF\ (\bigcup x \in A.\ B\ x) = (\bigsqcup x{\in}HF\ A.\ HF\ (B\ x))$
$\langle proof \rangle$

1.6 Subset relation and the Lattice Properties

Definition 1.10 (Subset relation).

instantiation *hf* :: *order*
begin
definition *less-eq-hf* **where** $A \le B \longleftrightarrow (\forall x.\ x \in A \longrightarrow x \in B)$

definition *less-hf* **where** $A < B \longleftrightarrow A \le B \wedge A \ne (B{::}hf)$

instance $\langle proof \rangle$
end

1.6.1 Rules for subsets

lemma *hsubsetI* [*intro*!]:
$(\bigwedge x.\ x{\in}A \Longrightarrow x{\in}B) \Longrightarrow A \le B$
$\langle proof \rangle$

Classical elimination rule

lemma *hsubsetCE* [*elim*]: $[\![\ A \le B;\ \ c{\notin}A \Longrightarrow P;\ \ c{\in}B \Longrightarrow P\ ]\!] \Longrightarrow P$
$\langle proof \rangle$

Rule in Modus Ponens style

lemma *hsubsetD* [*elim*]: $[\![\ A \le B;\ \ c{\in}A\ ]\!] \Longrightarrow c{\in}B$
$\langle proof \rangle$

Sometimes useful with premises in this order

lemma *rev-hsubsetD*: $[\![\ c{\in}A;\ A{\le}B\ ]\!] \Longrightarrow c{\in}B$
$\langle proof \rangle$

lemma *contra-hsubsetD*: $[\![\ A \le B;\ c \notin B\ ]\!] \Longrightarrow c \notin A$

⟨*proof*⟩

lemma *rev-contra-hsubsetD*: $[\![\, c \notin B;\ \ A \le B \,]\!] \Longrightarrow c \notin A$
⟨*proof*⟩

lemma *hf-equalityE*:
fixes $A :: hf$ **shows** $A = B \Longrightarrow (A \le B \Longrightarrow B \le A \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

1.6.2 Lattice properties

instantiation *hf* :: *distrib-lattice*
begin
instance ⟨*proof*⟩
end

instantiation *hf* :: *bounded-lattice-bot*
begin
definition $bot = (0{::}hf)$
instance ⟨*proof*⟩
end

lemma *hinter-hempty-left* [*simp*]: $0 \sqcap A = 0$
⟨*proof*⟩

lemma *hinter-hempty-right* [*simp*]: $A \sqcap 0 = 0$
⟨*proof*⟩

lemma *hunion-hempty-left* [*simp*]: $0 \sqcup A = A$
⟨*proof*⟩

lemma *hunion-hempty-right* [*simp*]: $A \sqcup 0 = A$
⟨*proof*⟩

lemma *less-eq-hempty* [*simp*]: $u \le 0 \longleftrightarrow u = (0{::}hf)$
⟨*proof*⟩

lemma *less-eq-insert1-iff* [*iff*]: $(hinsert\ x\ y) \le z \longleftrightarrow x \in z \wedge y \le z$
⟨*proof*⟩

lemma *less-eq-insert2-iff*:
$z \le (hinsert\ x\ y) \longleftrightarrow z \le y \vee (\exists\, u.\ hinsert\ x\ u = z \wedge x \notin u \wedge u \le y)$
⟨*proof*⟩

lemma *zero-le* [*simp*]: $0 \le (x{::}hf)$
⟨*proof*⟩

lemma *hinsert-eq-sup*: $b \triangleleft a = b \sqcup \{\!|a|\!\}$
⟨*proof*⟩

lemma *hunion-hinsert-left*: $hinsert\ x\ A \sqcup B = hinsert\ x\ (A \sqcup B)$
⟨*proof*⟩

lemma *hunion-hinsert-right*: $B \sqcup hinsert\ x\ A = hinsert\ x\ (B \sqcup A)$
⟨*proof*⟩

lemma *hinter-hinsert-left*: $hinsert\ x\ A \sqcap B = (if\ x \in B\ then\ hinsert\ x\ (A \sqcap B)\ else\ A \sqcap B)$
⟨*proof*⟩

lemma *hinter-hinsert-right*: $B \sqcap hinsert\ x\ A = (if\ x \in B\ then\ hinsert\ x\ (B \sqcap A)\ else\ B \sqcap A)$
⟨*proof*⟩

1.7 Foundation, Cardinality, Powersets

1.7.1 Foundation

Theorem 1.13: Foundation (Regularity) Property.

lemma *foundation*:
assumes z: $z \neq 0$ **shows** $\exists w.\ w \in z \wedge w \sqcap z = 0$
⟨*proof*⟩

lemma *hmem-not-refl*: $x \notin x$
⟨*proof*⟩

lemma *hmem-not-sym*: $\neg\ (x \in y \wedge y \in x)$
⟨*proof*⟩

lemma *hmem-ne*: $x \in y \Longrightarrow x \neq y$
⟨*proof*⟩

lemma *hmem-Sup-ne*: $x \in y \Longrightarrow \bigsqcup x \neq y$
⟨*proof*⟩

lemma *hpair-neq-fst*: $\langle a,b \rangle \neq a$
⟨*proof*⟩

lemma *hpair-neq-snd*: $\langle a,b \rangle \neq b$
⟨*proof*⟩

lemma *hpair-nonzero* [*simp*]: $\langle x,y \rangle \neq 0$
⟨*proof*⟩

lemma *zero-notin-hpair*: $0 \notin \langle x,y \rangle$
⟨*proof*⟩

1.7.2 Cardinality

First we need to hack the underlying representation

lemma *hfset-0* [*simp*]: $hfset\ 0 = \{\}$
⟨*proof*⟩

lemma *hfset-hinsert*: $hfset\ (b \triangleleft a) = insert\ a\ (hfset\ b)$
⟨*proof*⟩

lemma *hfset-hdiff*: $hfset\ (x - y) = hfset\ x - hfset\ y$
⟨*proof*⟩

definition *hcard* :: $hf \Rightarrow nat$
where $hcard\ x = card\ (hfset\ x)$

lemma *hcard-0* [*simp*]: $hcard\ 0 = 0$
⟨*proof*⟩

lemma *hcard-hinsert-if*: $hcard\ (hinsert\ x\ y) = (if\ x \in y\ then\ hcard\ y\ else\ Suc\ (hcard\ y))$
⟨*proof*⟩

lemma *hcard-union-inter*: $hcard\ (x \sqcup y) + hcard\ (x \sqcap y) = hcard\ x + hcard\ y$
⟨*proof*⟩

lemma *hcard-hdiff1-less*: $x \in z \Longrightarrow hcard\ (z - \{\!|x|\!\}) < hcard\ z$
⟨*proof*⟩

1.7.3 Powerset Operator

Theorem 1.11 (Existence of the power set).

lemma *powerset*: $\exists z.\ \forall u.\ u \in z \longleftrightarrow u \le x$
⟨*proof*⟩

definition *HPow* :: $hf \Rightarrow hf$
where $HPow\ x = (THE\ z.\ \forall u.\ u \in z \longleftrightarrow u \le x)$

lemma *HPow-iff* [*iff*]: $u \in HPow\ x \longleftrightarrow u \le x$
⟨*proof*⟩

lemma *HPow-mono*: $x \le y \Longrightarrow HPow\ x \le HPow\ y$
⟨*proof*⟩

lemma *HPow-mono-strict*: $x < y \Longrightarrow HPow\ x < HPow\ y$
⟨*proof*⟩

lemma *HPow-mono-iff* [*simp*]: $HPow\ x \le HPow\ y \longleftrightarrow x \le y$
⟨*proof*⟩

lemma *HPow-mono-strict-iff* [*simp*]: *HPow x* < *HPow y* ⟷ *x* < *y*
⟨*proof*⟩

1.8 Bounded Quantifiers

definition *HBall* :: *hf* ⇒ (*hf* ⇒ *bool*) ⇒ *bool* **where**
HBall A P ⟷ (∀ *x*. *x* ∈ *A* ⟶ *P x*) — bounded universal quantifiers

definition *HBex* :: *hf* ⇒ (*hf* ⇒ *bool*) ⇒ *bool* **where**
HBex A P ⟷ (∃ *x*. *x* ∈ *A* ∧ *P x*) — bounded existential quantifiers

syntax (*ASCII*)
-HBall :: *pttrn* ⇒ *hf* ⇒ *bool* ⇒ *bool* (‹(*3ALL* -<:-./ -)› [*0*, *0*, *10*] *10*)
-HBex :: *pttrn* ⇒ *hf* ⇒ *bool* ⇒ *bool* (‹(*3EX* -<:-./ -)› [*0*, *0*, *10*] *10*)
-HBex1 :: *pttrn* ⇒ *hf* ⇒ *bool* ⇒ *bool* (‹(*3EX*! -<:-./ -)› [*0*, *0*, *10*] *10*)
syntax
-HBall :: *pttrn* ⇒ *hf* ⇒ *bool* ⇒ *bool* (‹(*3*∀ -∈-./ -)› [*0*, *0*, *10*] *10*)
-HBex :: *pttrn* ⇒ *hf* ⇒ *bool* ⇒ *bool* (‹(*3*∃ -∈-./ -)› [*0*, *0*, *10*] *10*)
-HBex1 :: *pttrn* ⇒ *hf* ⇒ *bool* ⇒ *bool* (‹(*3*∃!-∈-./ -)› [*0*, *0*, *10*] *10*)
syntax-consts
-HBall ⇌ *HBall* **and**
-HBex ⇌ *HBex* **and**
-HBex1 ⇌ *Ex1*
translations
∀ *x*∈*A*. *P* ⇌ *CONST HBall A* (λ*x*. *P*)
∃ *x*∈*A*. *P* ⇌ *CONST HBex A* (λ*x*. *P*)
∃!*x*∈*A*. *P* ⇀ ∃!*x*. *x*∈*A* ∧ *P*

lemma *hball-cong* [*cong*]:
⟦ *A*=*A*′; ⋀*x*. *x* ∈ *A*′ ⟹ *P*(*x*) ⟷ *P*′(*x*) ⟧ ⟹ (∀ *x*∈*A*. *P*(*x*)) ⟷ (∀ *x*∈*A*′. *P*′(*x*))
⟨*proof*⟩

lemma *hballI* [*intro*!]: (⋀*x*. *x*∈*A* ⟹ *P x*) ⟹ ∀ *x*∈*A*. *P x*
⟨*proof*⟩

lemma *hbspec* [*dest?*]: ∀ *x*∈*A*. *P x* ⟹ *x*∈*A* ⟹ *P x*
⟨*proof*⟩

lemma *hballE* [*elim*]: ∀ *x*∈*A*. *P x* ⟹ (*P x* ⟹ *Q*) ⟹ (*x* ∉ *A* ⟹ *Q*) ⟹ *Q*
⟨*proof*⟩

lemma *hbex-cong* [*cong*]:
⟦ *A*=*A*′; ⋀*x*. *x* ∈ *A*′ ⟹ *P*(*x*) ⟷ *P*′(*x*) ⟧ ⟹ (∃ *x*∈*A*. *P*(*x*)) ⟷ (∃ *x*∈*A*′. *P*′(*x*))
⟨*proof*⟩

lemma *hbexI* [*intro*]: *P x* ⟹ *x*∈*A* ⟹ ∃ *x*∈*A*. *P x*
and *rev-hbexI* [*intro?*]: *x*∈*A* ⟹ *P x* ⟹ ∃ *x*∈*A*. *P x*

and *bexCI*: $(\forall x{\in}A.\ \neg\ P\ x \Longrightarrow P\ a) \Longrightarrow a{\in}A \Longrightarrow \exists x{\in}A.\ P\ x$
and *hbexE* [*elim*!]: $\exists x{\in}A.\ P\ x \Longrightarrow (\bigwedge x.\ x{\in}A \Longrightarrow P\ x \Longrightarrow Q) \Longrightarrow Q$
⟨*proof*⟩

lemma *hball-triv* [*simp*]: $(\forall x{\in}A.\ P) = ((\exists x.\ x{\in}A) \longrightarrow P)$
and *hbex-triv* [*simp*]: $(\exists x{\in}A.\ P) = ((\exists x.\ x{\in}A) \wedge P)$
— Dual form for existentials.
⟨*proof*⟩

lemma *hbex-triv-one-point1* [*simp*]: $(\exists x{\in}A.\ x = a) = (a{\in}A)$
⟨*proof*⟩

lemma *hbex-triv-one-point2* [*simp*]: $(\exists x{\in}A.\ a = x) = (a{\in}A)$
⟨*proof*⟩

lemma *hbex-one-point1* [*simp*]: $(\exists x{\in}A.\ x = a \wedge P\ x) = (a{\in}A \wedge P\ a)$
⟨*proof*⟩

lemma *hbex-one-point2* [*simp*]: $(\exists x{\in}A.\ a = x \wedge P\ x) = (a{\in}A \wedge P\ a)$
⟨*proof*⟩

lemma *hball-one-point1* [*simp*]: $(\forall x{\in}A.\ x = a \longrightarrow P\ x) = (a{\in}A \longrightarrow P\ a)$
⟨*proof*⟩

lemma *hball-one-point2* [*simp*]: $(\forall x{\in}A.\ a = x \longrightarrow P\ x) = (a{\in}A \longrightarrow P\ a)$
⟨*proof*⟩

lemma *hball-conj-distrib*:
$(\forall x{\in}A.\ P\ x \wedge Q\ x) \longleftrightarrow ((\forall x{\in}A.\ P\ x) \wedge (\forall x{\in}A.\ Q\ x))$
⟨*proof*⟩

lemma *hbex-disj-distrib*:
$(\exists x{\in}A.\ P\ x \vee Q\ x) \longleftrightarrow ((\exists x{\in}A.\ P\ x) \vee (\exists x{\in}A.\ Q\ x))$
⟨*proof*⟩

lemma *hb-all-simps* [*simp*, *no-atp*]:
$\bigwedge A\ P\ Q.\ (\forall x \in A.\ P\ x \vee Q) \longleftrightarrow ((\forall x \in A.\ P\ x) \vee Q)$
$\bigwedge A\ P\ Q.\ (\forall x \in A.\ P \vee Q\ x) \longleftrightarrow (P \vee (\forall x \in A.\ Q\ x))$
$\bigwedge A\ P\ Q.\ (\forall x \in A.\ P \longrightarrow Q\ x) \longleftrightarrow (P \longrightarrow (\forall x \in A.\ Q\ x))$
$\bigwedge A\ P\ Q.\ (\forall x \in A.\ P\ x \longrightarrow Q) \longleftrightarrow ((\exists x \in A.\ P\ x) \longrightarrow Q)$
$\bigwedge P.\ (\forall x \in \mathit{0}.\ P\ x) \longleftrightarrow \mathit{True}$
$\bigwedge a\ B\ P.\ (\forall x \in B \triangleleft a.\ P\ x) \longleftrightarrow (P\ a \wedge (\forall x \in B.\ P\ x))$
$\bigwedge P\ Q.\ (\forall x \in \mathit{HCollect}\ Q\ A.\ P\ x) \longleftrightarrow (\forall x \in A.\ Q\ x \longrightarrow P\ x)$
$\bigwedge A\ P.\ (\neg\ (\forall x \in A.\ P\ x)) \longleftrightarrow (\exists x \in A.\ \neg\ P\ x)$
⟨*proof*⟩

lemma *hb-ex-simps* [*simp*, *no-atp*]:
$\bigwedge A\ P\ Q.\ (\exists x \in A.\ P\ x \wedge Q) \longleftrightarrow ((\exists x \in A.\ P\ x) \wedge Q)$
$\bigwedge A\ P\ Q.\ (\exists x \in A.\ P \wedge Q\ x) \longleftrightarrow (P \wedge (\exists x \in A.\ Q\ x))$

$\bigwedge P.\ (\exists x \in 0.\ P\ x) \longleftrightarrow False$
$\bigwedge a\ B\ P.\ (\exists x \in B \triangleleft a.\ P\ x) \longleftrightarrow (P\ a \vee (\exists x \in B.\ P\ x))$
$\bigwedge P\ Q.\ (\exists x \in HCollect\ Q\ A.\ P\ x) \longleftrightarrow (\exists x \in A.\ Q\ x \wedge P\ x)$
$\bigwedge A\ P.\ (\neg(\exists x \in A.\ P\ x)) \longleftrightarrow (\forall x \in A.\ \neg\ P\ x)$
$\langle proof \rangle$

lemma *le-HCollect-iff*: $A \leq \{\!|x \in B.\ P\ x|\!\} \longleftrightarrow A \leq B \wedge (\forall x \in A.\ P\ x)$
$\langle proof \rangle$

1.9 Relations and Functions

definition *is-hpair* :: *hf* ⇒ *bool*
where $is\text{-}hpair\ z = (\exists x\ y.\ z = \langle x,y \rangle)$

definition *hconverse* :: *hf* ⇒ *hf*
where $hconverse(r) = \{\!|z.\ w \in r,\ \exists x\ y.\ w = \langle x,y \rangle \wedge z = \langle y,x \rangle|\!\}$

definition *hdomain* :: *hf* ⇒ *hf*
where $hdomain(r) = \{\!|x.\ w \in r,\ \exists y.\ w = \langle x,y \rangle|\!\}$

definition *hrange* :: *hf* ⇒ *hf*
where $hrange(r) = hdomain(hconverse(r))$

definition *hrelation* :: *hf* ⇒ *bool*
where $hrelation(r) = (\forall z.\ z \in r \longrightarrow is\text{-}hpair\ z)$

definition *hrestrict* :: *hf* ⇒ *hf* ⇒ *hf*
— Restrict the relation r to the domain A
where $hrestrict\ r\ A = \{\!|z \in r.\ \exists x \in A.\ \exists y.\ z = \langle x,y \rangle|\!\}$

definition *nonrestrict* :: *hf* ⇒ *hf* ⇒ *hf*
where $nonrestrict\ r\ A = \{\!|z \in r.\ \forall x \in A.\ \forall y.\ z \neq \langle x,y \rangle|\!\}$

definition *hfunction* :: *hf* ⇒ *bool*
where $hfunction(r) = (\forall x\ y.\ \langle x,y \rangle \in r \longrightarrow (\forall y'.\ \langle x,y' \rangle \in r \longrightarrow y=y'))$

definition *app* :: *hf* ⇒ *hf* ⇒ *hf*
where $app\ f\ x = (THE\ y.\ \langle x,\ y \rangle \in f)$

lemma *hrestrict-iff* [*iff*]:
$z \in hrestrict\ r\ A \longleftrightarrow z \in r \wedge (\exists\ x\ y.\ z = \langle x,\ y \rangle \wedge x \in A)$
$\langle proof \rangle$

lemma *hrelation-0* [*simp*]: *hrelation 0*
$\langle proof \rangle$

lemma *hrelation-restr* [*iff*]: *hrelation* (*hrestrict r x*)
$\langle proof \rangle$

lemma *hrelation-hunion* [*simp*]: *hrelation* $(f \sqcup g) \longleftrightarrow$ *hrelation* $f \wedge$ *hrelation* g
⟨*proof*⟩

lemma *hfunction-restr*: *hfunction* $r \Longrightarrow$ *hfunction* (*hrestrict* r x)
⟨*proof*⟩

lemma *hdomain-restr* [*simp*]: *hdomain* (*hrestrict* r x) $=$ *hdomain* $r \sqcap x$
⟨*proof*⟩

lemma *hdomain-0* [*simp*]: *hdomain* $0 = 0$
⟨*proof*⟩

lemma *hdomain-ins* [*simp*]: *hdomain* $(r \triangleleft \langle x, y\rangle) =$ *hdomain* $r \triangleleft x$
⟨*proof*⟩

lemma *hdomain-hunion* [*simp*]: *hdomain* $(f \sqcup g) =$ *hdomain* $f \sqcup$ *hdomain* g
⟨*proof*⟩

lemma *hdomain-not-mem* [*iff*]: ⟨*hdomain* r, a⟩ $\notin r$
⟨*proof*⟩

lemma *app-singleton* [*simp*]: *app* $\{\!|\langle x, y\rangle|\!\}$ $x = y$
⟨*proof*⟩

lemma *app-equality*: *hfunction* $f \Longrightarrow \langle x, y\rangle \in f \Longrightarrow$ *app* f $x = y$
⟨*proof*⟩

lemma *app-ins2*: $x' \neq x \Longrightarrow$ *app* $(f \triangleleft \langle x, y\rangle)$ $x' =$ *app* f x'
⟨*proof*⟩

lemma *hfunction-0* [*simp*]: *hfunction* 0
⟨*proof*⟩

lemma *hfunction-ins*: *hfunction* $f \Longrightarrow x \notin$ *hdomain* $f \Longrightarrow$ *hfunction* $(f \triangleleft \langle x, y\rangle)$
⟨*proof*⟩

lemma *hdomainI*: $\langle x, y\rangle \in f \Longrightarrow x \in$ *hdomain* f
⟨*proof*⟩

lemma *hfunction-hunion*: *hdomain* $f \sqcap$ *hdomain* $g = 0$
$\Longrightarrow$ *hfunction* $(f \sqcup g) \longleftrightarrow$ *hfunction* $f \wedge$ *hfunction* g
⟨*proof*⟩

lemma *app-hrestrict* [*simp*]: $x \in A \Longrightarrow$ *app* (*hrestrict* f A) $x =$ *app* f x
⟨*proof*⟩

1.10 Operations on families of sets

definition *HLambda* :: $hf \Rightarrow (hf \Rightarrow hf) \Rightarrow hf$

where $HLambda\ A\ b = RepFun\ A\ (\lambda x.\ \langle x,\ b\ x\rangle)$

definition $HSigma :: hf \Rightarrow (hf \Rightarrow hf) \Rightarrow hf$
where $HSigma\ A\ B = (\bigsqcup x{\in}A.\ \bigsqcup y{\in}B(x).\ \{\!|\langle x,y\rangle|\!\})$

definition $HPi :: hf \Rightarrow (hf \Rightarrow hf) \Rightarrow hf$
where $HPi\ A\ B = \{\!|\ f \in HPow(HSigma\ A\ B).\ A \leq hdomain(f) \wedge hfunction(f)|\!\}$

syntax (*ASCII*)
-*PROD* :: [*pttrn*, *hf*, *hf*] ⇒ *hf* (‹(3PROD -<:-./ -)› *10*)
-*SUM* :: [*pttrn*, *hf*, *hf*] ⇒ *hf* (‹(3SUM -<:-./ -)› *10*)
-*lam* :: [*pttrn*, *hf*, *hf*] ⇒ *hf* (‹(3lam -<:-./ -)› *10*)
syntax
-*PROD* :: [*pttrn*, *hf*, *hf*] ⇒ *hf* (‹(3∏ -∈-./ -)› *10*)
-*SUM* :: [*pttrn*, *hf*, *hf*] ⇒ *hf* (‹(3∑ -∈-./ -)› *10*)
-*lam* :: [*pttrn*, *hf*, *hf*] ⇒ *hf* (‹(3λ-∈-./ -)› *10*)
syntax-consts
-*PROD* ⇌ *HPi* **and**
-*SUM* ⇌ *HSigma* **and**
-*lam* ⇌ *HLambda*
translations
$\prod x{\in}A.\ B \rightleftharpoons CONST\ HPi\ A\ (\lambda x.\ B)$
$\sum x{\in}A.\ B \rightleftharpoons CONST\ HSigma\ A\ (\lambda x.\ B)$
$\lambda x{\in}A.\ f \rightleftharpoons CONST\ HLambda\ A\ (\lambda x.\ f)$

1.10.1 Rules for Unions and Intersections of families

lemma *HUN-iff* [*simp*]: $b \in (\bigsqcup x{\in}A.\ B(x)) \longleftrightarrow (\exists\, x{\in}A.\ b \in B(x))$
⟨*proof*⟩

lemma *HUN-I*: $[\![\ a \in A;\ \ b \in B(a)\ ]\!] \Longrightarrow b \in (\bigsqcup x{\in}A.\ B(x))$
⟨*proof*⟩

lemma *HUN-E* [*elim*!]: **assumes** $b \in (\bigsqcup x{\in}A.\ B(x))$ **obtains** x **where** $x \in A$ $\ b \in B(x)$
⟨*proof*⟩

lemma *HINT-iff*: $b \in (\bigsqcap x{\in}A.\ B(x)) \longleftrightarrow (\forall\, x{\in}A.\ b \in B(x)) \wedge A{\neq}0$
⟨*proof*⟩

lemma *HINT-I*: $[\![\ \bigwedge x.\ x \in A \Longrightarrow b \in B(x);\ \ A{\neq}0\ ]\!] \Longrightarrow b \in (\bigsqcap x{\in}A.\ B(x))$
⟨*proof*⟩

lemma *HINT-E*: $[\![\ b \in (\bigsqcap x{\in}A.\ B(x));\ \ a \in A\ ]\!] \Longrightarrow b \in B(a)$
⟨*proof*⟩

1.10.2 Generalized Cartesian product

lemma *HSigma-iff* [*simp*]: $\langle a,b\rangle \in$ *HSigma A B* $\longleftrightarrow a \in A \wedge b \in B(a)$
⟨*proof*⟩

lemma *HSigmaI* [*intro*!]: $[\![\ a \in A;\ \ b \in B(a)\ ]\!] \Longrightarrow \langle a,b\rangle \in$ *HSigma A B*
⟨*proof*⟩

lemmas *HSigmaD1* = *HSigma-iff* [*THEN iffD1*, *THEN conjunct1*]
lemmas *HSigmaD2* = *HSigma-iff* [*THEN iffD1*, *THEN conjunct2*]

The general elimination rule

lemma *HSigmaE* [*elim*!]:
assumes $c \in$ *HSigma A B*
obtains *x y* **where** $x \in A$ $y \in B(x)$ $c=\langle x,y\rangle$
⟨*proof*⟩

lemma *HSigmaE2* [*elim*!]:
assumes $\langle a,b\rangle \in$ *HSigma A B* **obtains** $a \in A$ **and** $b \in B(a)$
⟨*proof*⟩

lemma *HSigma-empty1* [*simp*]: *HSigma 0 B* = *0*
⟨*proof*⟩

instantiation *hf* :: *times*
begin
definition $A * B =$ *HSigma A* $(\lambda x.\ B)$
instance ⟨*proof*⟩
end

lemma *times-iff* [*simp*]: $\langle a,b\rangle \in A * B \longleftrightarrow a \in A \wedge b \in B$
⟨*proof*⟩

lemma *timesI* [*intro*!]: $[\![\ a \in A;\ \ b \in B\ ]\!] \Longrightarrow \langle a,b\rangle \in A * B$
⟨*proof*⟩

lemmas *timesD1* = *times-iff* [*THEN iffD1*, *THEN conjunct1*]
lemmas *timesD2* = *times-iff* [*THEN iffD1*, *THEN conjunct2*]

The general elimination rule

lemma *timesE* [*elim*!]:
assumes *c*: $c \in A * B$
obtains *x y* **where** $x \in A$ $y \in B$ $c=\langle x,y\rangle$ ⟨*proof*⟩

...and a specific one

lemma *timesE2* [*elim*!]:
assumes $\langle a,b\rangle \in A * B$ **obtains** $a \in A$ **and** $b \in B$
⟨*proof*⟩

lemma *times-empty1* [*simp*]: $0 * B = (0{::}hf)$

$\langle proof \rangle$

lemma *times-empty2* [*simp*]: $A*0 = (0::hf)$
$\langle proof \rangle$

lemma *times-empty-iff*: $A*B=0 \longleftrightarrow A=0 \vee B=(0::hf)$
$\langle proof \rangle$

instantiation *hf* :: *mult-zero*
begin
instance $\langle proof \rangle$
end

1.11 Disjoint Sum

instantiation *hf* :: *zero-neq-one*
begin
definition *One-hf-def*: $1 = \{\!|0|\!\}$
instance $\langle proof \rangle$
end

instantiation *hf* :: *plus*
begin
definition $A + B = (\{\!|0|\!\} * A) \sqcup (\{\!|1|\!\} * B)$
instance $\langle proof \rangle$
end

definition *Inl* :: $hf \Rightarrow hf$ **where**
$Inl(a) \equiv \langle 0,a \rangle$

definition *Inr* :: $hf \Rightarrow hf$ **where**
$Inr(b) \equiv \langle 1,b \rangle$

lemmas *sum-defs* = *plus-hf-def Inl-def Inr-def*

lemma *Inl-nonzero* [*simp*]:$Inl\ x \neq 0$
$\langle proof \rangle$

lemma *Inr-nonzero* [*simp*]:$Inr\ x \neq 0$
$\langle proof \rangle$

Introduction rules for the injections (as equivalences)

lemma *Inl-in-sum-iff* [*iff*]: $Inl(a) \in A+B \longleftrightarrow a \in A$
$\langle proof \rangle$

lemma *Inr-in-sum-iff* [*iff*]: $Inr(b) \in A+B \longleftrightarrow b \in B$
$\langle proof \rangle$

Elimination rule

lemma *sumE* [*elim*!]:
assumes *u*: $u \in A{+}B$
obtains *x* **where** $x \in A$ $u{=}Inl(x)$ | *y* **where** $y \in B$ $u{=}Inr(y)$ ⟨*proof*⟩

Injection and freeness equivalences, for rewriting

lemma *Inl-iff* [*iff*]: $Inl(a){=}Inl(b) \longleftrightarrow a{=}b$
⟨*proof*⟩

lemma *Inr-iff* [*iff*]: $Inr(a){=}Inr(b) \longleftrightarrow a{=}b$
⟨*proof*⟩

lemma *Inl-Inr-iff* [*iff*]: $Inl(a){=}Inr(b) \longleftrightarrow False$
⟨*proof*⟩

lemma *Inr-Inl-iff* [*iff*]: $Inr(b){=}Inl(a) \longleftrightarrow False$
⟨*proof*⟩

lemma *sum-empty* [*simp*]: $0{+}0 = (0{::}hf)$
⟨*proof*⟩

lemma *sum-iff*: $u \in A{+}B \longleftrightarrow (\exists\, x.\ x \in A \wedge u{=}Inl(x)) \vee (\exists\, y.\ y \in B \wedge u{=}Inr(y))$
⟨*proof*⟩

lemma *sum-subset-iff*:
fixes *A* :: *hf* **shows** $A{+}B \le C{+}D \longleftrightarrow A{\le}C \wedge B{\le}D$
⟨*proof*⟩

lemma *sum-equal-iff*:
fixes *A* :: *hf* **shows** $A{+}B = C{+}D \longleftrightarrow A{=}C \wedge B{=}D$
⟨*proof*⟩

definition *is-hsum* :: $hf \Rightarrow bool$
where $is\text{-}hsum\ z = (\exists\, x.\ z = Inl\ x \vee z = Inr\ x)$

definition *sum-case* :: $(hf \Rightarrow 'a) \Rightarrow (hf \Rightarrow 'a) \Rightarrow hf \Rightarrow 'a$
where
$sum\text{-}case\ f\ g\ a \equiv$
$THE\ z.\ (\forall\, x.\ a = Inl\ x \longrightarrow z = f\ x) \wedge (\forall\, y.\ a = Inr\ y \longrightarrow z = g\ y) \wedge (\neg$
$is\text{-}hsum\ a \longrightarrow z = undefined)$

lemma *sum-case-Inl* [*simp*]: $sum\text{-}case\ f\ g\ (Inl\ x) = f\ x$
⟨*proof*⟩

lemma *sum-case-Inr* [*simp*]: $sum\text{-}case\ f\ g\ (Inr\ y) = g\ y$
⟨*proof*⟩

lemma *sum-case-non* [*simp*]: $\neg\ is\text{-}hsum\ a \Longrightarrow sum\text{-}case\ f\ g\ a = undefined$
⟨*proof*⟩

lemma *is-hsum-cases*: $(\exists x.\ z = \mathit{Inl}\ x \vee z = \mathit{Inr}\ x) \vee \neg\ \mathit{is\text{-}hsum}\ z$
⟨*proof*⟩

lemma *sum-case-split*:
$P\ (\mathit{sum\text{-}case}\ f\ g\ a) \longleftrightarrow (\forall x.\ a = \mathit{Inl}\ x \longrightarrow P(f\ x)) \wedge (\forall y.\ a = \mathit{Inr}\ y \longrightarrow P(g\ y)) \wedge (\neg\ \mathit{is\text{-}hsum}\ a \longrightarrow P\ \mathit{undefined})$
⟨*proof*⟩

lemma *sum-case-split-asm*:
$P\ (\mathit{sum\text{-}case}\ f\ g\ a) \longleftrightarrow \neg\ ((\exists x.\ a = \mathit{Inl}\ x \wedge \neg\ P(f\ x)) \vee (\exists y.\ a = \mathit{Inr}\ y \wedge \neg\ P(g\ y)) \vee (\neg\ \mathit{is\text{-}hsum}\ a \wedge \neg\ P\ \mathit{undefined}))$
⟨*proof*⟩

end

Chapter 2

Ordinals, Sequences and Ordinal Recursion

theory *Ordinal* **imports** *HF*
begin

2.1 Ordinals

2.1.1 Basic Definitions

Definition 2.1. We say that x is transitive if every element of x is a subset of x.

definition
  $Transset :: hf \Rightarrow bool$ **where**
    $Transset(x) \equiv \forall y.\ y \in x \longrightarrow y \leq x$

lemma *Transset-sup*: $Transset\ x \Longrightarrow Transset\ y \Longrightarrow Transset\ (x \sqcup y)$
  ⟨*proof*⟩

lemma *Transset-inf*: $Transset\ x \Longrightarrow Transset\ y \Longrightarrow Transset\ (x \sqcap y)$
  ⟨*proof*⟩

lemma *Transset-hinsert*: $Transset\ x \Longrightarrow y \leq x \Longrightarrow Transset\ (x \triangleleft y)$
  ⟨*proof*⟩

In HF, the ordinals are simply the natural numbers. But the definitions are the same as for transfinite ordinals.

definition
  $Ord :: hf \Rightarrow bool$ **where**
    $Ord(k) \equiv Transset(k) \wedge (\forall x \in k.\ Transset(x))$

2.1.2 Definition 2.2 (Successor).

definition

succ :: *hf* ⇒ *hf* **where**
succ(*x*) ≡ *hinsert x x*

lemma *succ-iff* [*simp*]: $x \in succ\ y \longleftrightarrow x=y \lor x \in y$
⟨*proof*⟩

lemma *succ-ne-self* [*simp*]: $i \neq succ\ i$
⟨*proof*⟩

lemma *succ-notin-self*: $succ\ i \notin i$
⟨*proof*⟩

lemma *succE* [*elim?*]: **assumes** $x \in succ\ y$ **obtains** $x=y \mid x \in y$
⟨*proof*⟩

lemma *hmem-succ-ne*: $succ\ x \in y \Longrightarrow x \neq y$
⟨*proof*⟩

lemma *hball-succ* [*simp*]: $(\forall x \in succ\ k.\ P\ x) \longleftrightarrow P\ k \land (\forall x \in k.\ P\ x)$
⟨*proof*⟩

lemma *hbex-succ* [*simp*]: $(\exists x \in succ\ k.\ P\ x) \longleftrightarrow P\ k \lor (\exists x \in k.\ P\ x)$
⟨*proof*⟩

lemma *One-hf-eq-succ*: $1 = succ\ 0$
⟨*proof*⟩

lemma *zero-hmem-one* [*iff*]: $x \in 1 \longleftrightarrow x = 0$
⟨*proof*⟩

lemma *hball-One* [*simp*]: $(\forall x{\in}1.\ P\ x) = P\ 0$
⟨*proof*⟩

lemma *hbex-One* [*simp*]: $(\exists x{\in}1.\ P\ x) = P\ 0$
⟨*proof*⟩

lemma *hpair-neq-succ* [*simp*]: $\langle x,y\rangle \neq succ\ k$
⟨*proof*⟩

lemma *succ-neq-hpair* [*simp*]: $succ\ k \neq \langle x,y\rangle$
⟨*proof*⟩

lemma *hpair-neq-one* [*simp*]: $\langle x,y\rangle \neq 1$
⟨*proof*⟩

lemma *one-neq-hpair* [*simp*]: $1 \neq \langle x,y\rangle$
⟨*proof*⟩

lemma *hmem-succ-self* [*simp*]: $k \in succ\ k$

⟨*proof*⟩

lemma *hmem-succ*: $l \in k \Longrightarrow l \in succ\ k$
⟨*proof*⟩

Theorem 2.3.

lemma *Ord-0* [*iff*]: *Ord 0*
⟨*proof*⟩

lemma *Ord-succ*: $Ord(k) \Longrightarrow Ord(succ(k))$
⟨*proof*⟩

lemma *Ord-1* [*iff*]: *Ord 1*
⟨*proof*⟩

lemma *OrdmemD*: $Ord(k) \Longrightarrow j \in k \Longrightarrow j \le k$
⟨*proof*⟩

lemma *Ord-trans*: $[\![\ i{\in}j;\ \ j{\in}k;\ \ Ord(k)\ ]\!] \Longrightarrow i{\in}k$
⟨*proof*⟩

lemma *hmem-0-Ord*:
assumes *k*: $Ord(k)$ **and** *knz*: $k \neq 0$ **shows** $0 \in k$
⟨*proof*⟩

lemma *Ord-in-Ord*: $[\![\ Ord(k);\ \ m \in k\ ]\!] \Longrightarrow Ord(m)$
⟨*proof*⟩

2.1.3 Induction, Linearity, etc.

lemma *Ord-induct* [*consumes 1*, *case-names step*]:
assumes *k*: $Ord(k)$
and *step*: $\bigwedge x.[\![\ Ord(x);\ \ \bigwedge y.\ y \in x \Longrightarrow P(y)\ ]\!] \Longrightarrow P(x)$
shows $P(k)$
⟨*proof*⟩

Theorem 2.4 (Comparability of ordinals).

lemma *Ord-linear*: $Ord(k) \Longrightarrow Ord(l) \Longrightarrow k{\in}l \lor k{=}l \lor l{\in}k$
⟨*proof*⟩

The trichotomy law for ordinals

lemma *Ord-linear-lt*:
assumes *o*: $Ord(k)\ Ord(l)$
obtains (*lt*) $k{\in}l$ | (*eq*) $k{=}l$ | (*gt*) $l{\in}k$
⟨*proof*⟩

lemma *Ord-linear2*:
assumes *o*: $Ord(k)\ Ord(l)$
obtains (*lt*) $k{\in}l$ | (*ge*) $l \le k$

$\langle proof \rangle$

lemma *Ord-linear-le*:
assumes *o*: $Ord(k)$ $Ord(l)$
obtains (*le*) $k \le l$ | (*ge*) $l \le k$
$\langle proof \rangle$

lemma *hunion-less-iff* [*simp*]: $[\![Ord\ i;\ Ord\ j]\!] \Longrightarrow i \sqcup j < k \longleftrightarrow i<k \wedge j<k$
$\langle proof \rangle$

Theorem 2.5

lemma *Ord-mem-iff-lt*: $Ord(k) \Longrightarrow Ord(l) \Longrightarrow k \in l \longleftrightarrow k < l$
$\langle proof \rangle$

lemma *le-succE*: $succ\ i \le succ\ j \Longrightarrow i \le j$
$\langle proof \rangle$

lemma *le-succ-iff*: $Ord\ i \Longrightarrow Ord\ j \Longrightarrow succ\ i \le succ\ j \longleftrightarrow i \le j$
$\langle proof \rangle$

lemma *succ-inject-iff* [*iff*]: $succ\ i = succ\ j \longleftrightarrow i = j$
$\langle proof \rangle$

lemma *mem-succ-iff* [*simp*]: $Ord\ j \Longrightarrow succ\ i \in succ\ j \longleftrightarrow i \in j$
$\langle proof \rangle$

lemma *Ord-mem-succ-cases*:
assumes $Ord(k)$ $l \in k$
shows $succ\ l = k \vee succ\ l \in k$
$\langle proof \rangle$

2.1.4 Supremum and Infimum

lemma *Ord-Union* [*intro,simp*]: $[\![\ \bigwedge i.\ i \in A \Longrightarrow Ord(i)\ ]\!] \Longrightarrow Ord(\bigsqcup A)$
$\langle proof \rangle$

lemma *Ord-Inter* [*intro,simp*]:
assumes $\bigwedge i.\ i \in A \Longrightarrow Ord(i)$ **shows** $Ord(\bigsqcap A)$
$\langle proof \rangle$

Theorem 2.7. Every set x of ordinals is ordered by the binary relation <. Moreover if x = 0 then x has a smallest and a largest element.

lemma *hmem-Sup-Ords*: $[\![A \neq 0;\ \bigwedge i.\ i \in A \Longrightarrow Ord(i)]\!] \Longrightarrow \bigsqcup A \in A$
$\langle proof \rangle$

lemma *hmem-Inf-Ords*: $[\![A \neq 0;\ \bigwedge i.\ i \in A \Longrightarrow Ord(i)]\!] \Longrightarrow \bigsqcap A \in A$
$\langle proof \rangle$

lemma *Ord-pred*: $[\![Ord(k);\ k \neq 0]\!] \Longrightarrow succ(\bigsqcup k) = k$

⟨*proof*⟩

lemma *Ord-cases* [*cases type*: *hf*, *case-names 0 succ*]:
assumes *Ok*: *Ord*(k)
obtains $k = 0$ | l **where** *Ord l succ l* $= k$
⟨*proof*⟩

lemma *Ord-induct2* [*consumes 1*, *case-names 0 succ*, *induct type*: *hf*]:
assumes k: *Ord*(k)
and P: $P\ 0\ \bigwedge k.\ Ord\ k \Longrightarrow P\ k \Longrightarrow P\ (succ\ k)$
shows $P\ k$
⟨*proof*⟩

lemma *Ord-succ-iff* [*iff*]: $Ord\ (succ\ k) = Ord\ k$
⟨*proof*⟩

lemma [*simp*]: $succ\ k \neq 0$
⟨*proof*⟩

lemma *Ord-Sup-succ-eq* [*simp*]: $Ord\ k \Longrightarrow \bigsqcup(succ\ k) = k$
⟨*proof*⟩

lemma *Ord-lt-succ-iff-le*: $Ord\ k \Longrightarrow Ord\ l \Longrightarrow k < succ\ l \longleftrightarrow k \leq l$
⟨*proof*⟩

lemma *zero-in-Ord*: $Ord\ k \Longrightarrow k{=}0 \vee 0 \in k$
⟨*proof*⟩

lemma *hpair-neq-Ord*: $Ord\ k \Longrightarrow \langle x,y\rangle \neq k$
⟨*proof*⟩

lemma *hpair-neq-Ord′*: **assumes** k: *Ord k* **shows** $k \neq \langle x,y\rangle$
⟨*proof*⟩

lemma *Not-Ord-hpair* [*iff*]: $\neg\ Ord\ \langle x,y\rangle$
⟨*proof*⟩

lemma *is-hpair* [*simp*]: *is-hpair* $\langle x,y\rangle$
⟨*proof*⟩

lemma *Ord-not-hpair*: $Ord\ x \Longrightarrow \neg$ *is-hpair* x
⟨*proof*⟩

lemma *zero-in-succ* [*simp*,*intro*]: $Ord\ i \Longrightarrow 0 \in succ\ i$
⟨*proof*⟩

2.1.5 Converting Between Ordinals and Natural Numbers

fun *ord-of* :: *nat* ⇒ *hf*

where
  ord-of 0 = 0
| *ord-of (Suc k) = succ (ord-of k)*

lemma *Ord-ord-of [simp]: Ord (ord-of k)*
  ⟨*proof*⟩

lemma *ord-of-inject [iff]: ord-of i = ord-of j ⟷ i=j*
⟨*proof*⟩

lemma *ord-of-minus-1: n > 0 ⟹ ord-of n = succ (ord-of (n − 1))*
  ⟨*proof*⟩

definition *nat-of-ord :: hf ⇒ nat*
  where *nat-of-ord x = (THE n. x = ord-of n)*

lemma *nat-of-ord-ord-of [simp]: nat-of-ord (ord-of n) = n*
  ⟨*proof*⟩

lemma *nat-of-ord-0 [simp]: nat-of-ord 0 = 0*
  ⟨*proof*⟩

lemma *ord-of-nat-of-ord [simp]: Ord x ⟹ ord-of (nat-of-ord x) = x*
⟨*proof*⟩

lemma *nat-of-ord-inject: Ord x ⟹ Ord y ⟹ nat-of-ord x = nat-of-ord y ⟷ x = y*
  ⟨*proof*⟩

lemma *nat-of-ord-succ [simp]: Ord x ⟹ nat-of-ord (succ x) = Suc (nat-of-ord x)*
  ⟨*proof*⟩

lemma *inj-ord-of: inj-on ord-of A*
  ⟨*proof*⟩

lemma *hfset-ord-of: hfset (ord-of n) = ord-of ‘ {0..<n}*
  ⟨*proof*⟩

lemma *bij-betw-ord-of: bij-betw ord-of {0..<n} (hfset (ord-of n))*
  ⟨*proof*⟩

lemma *bij-betw-ord-ofI:*
  bij-betw h A {0..<n} ⟹ bij-betw (ord-of ∘ h) A (hfset (ord-of n))
  ⟨*proof*⟩

2.2 Sequences and Ordinal Recursion

Definition 3.2 (Sequence).

definition *Seq* :: *hf* ⇒ *hf* ⇒ *bool*
  where *Seq s k* ⟷ *hrelation s* ∧ *hfunction s* ∧ *k* ≤ *hdomain s*

lemma *Seq-0* [*iff*]: *Seq 0 0*
  ⟨*proof*⟩

lemma *Seq-succ-D*: *Seq s* (*succ k*) ⟹ *Seq s k*
  ⟨*proof*⟩

lemma *Seq-Ord-D*: *Seq s k* ⟹ *l* ∈ *k* ⟹ *Ord k* ⟹ *Seq s l*
  ⟨*proof*⟩

lemma *Seq-restr*: *Seq s* (*succ k*) ⟹ *Seq* (*hrestrict s k*) *k*
  ⟨*proof*⟩

lemma *Seq-Ord-restr*: ⟦*Seq s k*; *l* ∈ *k*; *Ord k*⟧ ⟹ *Seq* (*hrestrict s l*) *l*
  ⟨*proof*⟩

lemma *Seq-ins*: ⟦*Seq s k*; *k* ∉ *hdomain s*⟧ ⟹ *Seq* (*s* ◃ ⟨*k*, *y*⟩) (*succ k*)
  ⟨*proof*⟩

definition *insf* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf*
  where *insf s k y* ≡ *nonrestrict s* ⦃*k*⦄ ◃ ⟨*k*, *y*⟩

lemma *hrelation-insf*: *hrelation s* ⟹ *hrelation* (*insf s k y*)
  ⟨*proof*⟩

lemma *hfunction-insf*: *hfunction s* ⟹ *hfunction* (*insf s k y*)
  ⟨*proof*⟩

lemma *hdomain-insf*: *k* ≤ *hdomain s* ⟹ *succ k* ≤ *hdomain* (*insf s k y*)
  ⟨*proof*⟩

lemma *Seq-insf*: *Seq s k* ⟹ *Seq* (*insf s k y*) (*succ k*)
  ⟨*proof*⟩

lemma *Seq-succ-iff*: *Seq s* (*succ k*) ⟷ *Seq s k* ∧ (∃ *y*. ⟨*k*, *y*⟩ ∈ *s*)
  ⟨*proof*⟩

lemma *nonrestrictD*: *a* ∈ *nonrestrict s X* ⟹ *a* ∈ *s*
  ⟨*proof*⟩

lemma *hpair-in-nonrestrict-iff* [*simp*]:
  ⟨*a*,*b*⟩ ∈ *nonrestrict s X* ⟷ ⟨*a*,*b*⟩ ∈ *s* ∧ ¬ *a* ∈ *X*
  ⟨*proof*⟩

lemma *app-nonrestrict-Seq*: *Seq s k* ⟹ *z* ∉ *X* ⟹ *app* (*nonrestrict s X*) *z* = *app s z*
  ⟨*proof*⟩

lemma *app-insf-Seq*: $Seq\ s\ k \Longrightarrow app\ (insf\ s\ k\ y)\ k = y$
⟨*proof*⟩

lemma *app-insf2-Seq*: $Seq\ s\ k \Longrightarrow k' \neq k \Longrightarrow app\ (insf\ s\ k\ y)\ k' = app\ s\ k'$
⟨*proof*⟩

lemma *app-insf-Seq-if*: $Seq\ s\ k \Longrightarrow app\ (insf\ s\ k\ y)\ k' = (if\ k' = k\ then\ y\ else\ app\ s\ k')$
⟨*proof*⟩

lemma *Seq-imp-eq-app*: $[\![Seq\ s\ d;\ \langle x,y\rangle \in s]\!] \Longrightarrow app\ s\ x = y$
⟨*proof*⟩

lemma *Seq-iff-app*: $[\![Seq\ s\ d;\ x \in d]\!] \Longrightarrow \langle x,y\rangle \in s \longleftrightarrow app\ s\ x = y$
⟨*proof*⟩

lemma *Exists-iff-app*: $Seq\ s\ d \Longrightarrow x \in d \Longrightarrow (\exists\, y.\ \langle x,\ y\rangle \in s \wedge P\ y) = P\ (app\ s\ x)$
⟨*proof*⟩

lemma *Ord-trans2*: $[\![i2 \in i;\ i \in j;\ \ j \in k;\ \ Ord\ k]\!] \Longrightarrow i2 \in k$
⟨*proof*⟩

definition *ord-rec-Seq* :: $hf \Rightarrow (hf \Rightarrow hf) \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$
where
$ord\text{-}rec\text{-}Seq\ T\ G\ s\ k\ y \longleftrightarrow$
$(Seq\ s\ k \wedge y = G\ (app\ s\ (\bigsqcup k)) \wedge app\ s\ 0 = T \wedge$
$(\forall\, n.\ succ\ n \in k \longrightarrow app\ s\ (succ\ n) = G\ (app\ s\ n)))$

lemma *Seq-succ-insf*:
assumes *s*: $Seq\ s\ (succ\ k)$ **shows** $\exists\ y.\ s = insf\ s\ k\ y$
⟨*proof*⟩

lemma *ord-rec-Seq-succ-iff*:
assumes *k*: $Ord\ k$ **and** *knz*: $k \neq 0$
shows $ord\text{-}rec\text{-}Seq\ T\ G\ s\ (succ\ k)\ z \longleftrightarrow (\exists\ s'\ y.\ ord\text{-}rec\text{-}Seq\ T\ G\ s'\ k\ y \wedge z = G\ y \wedge s = insf\ s'\ k\ y)$
⟨*proof*⟩

lemma *ord-rec-Seq-functional*:
$Ord\ k \Longrightarrow k \neq 0 \Longrightarrow ord\text{-}rec\text{-}Seq\ T\ G\ s\ k\ y \Longrightarrow ord\text{-}rec\text{-}Seq\ T\ G\ s'\ k\ y' \Longrightarrow y' = y$
⟨*proof*⟩

definition *ord-recp* :: $hf \Rightarrow (hf \Rightarrow hf) \Rightarrow (hf \Rightarrow hf) \Rightarrow hf \Rightarrow hf \Rightarrow bool$
where
$ord\text{-}recp\ T\ G\ H\ x\ y =$
$(if\ x{=}0\ then\ y = T$

else
if $Ord(x)$ *then* $\exists$ *s. ord-rec-Seq T G s x y*
else $y = H\ x$)

lemma *ord-recp-functional*: *ord-recp T G H x y* $\Longrightarrow$ *ord-recp T G H x y'* $\Longrightarrow$ $y' = y$
⟨*proof*⟩

lemma *ord-recp-succ-iff*:
assumes *k*: *Ord k* **shows** *ord-recp T G H* (*succ k*) $z \longleftrightarrow (\exists\, y.\ z = G\ y \wedge$ *ord-recp T G H k y*)
⟨*proof*⟩

definition *ord-rec* :: $hf \Rightarrow (hf \Rightarrow hf) \Rightarrow (hf \Rightarrow hf) \Rightarrow hf \Rightarrow hf$
where
ord-rec T G H x = (*THE y. ord-recp T G H x y*)

lemma *ord-rec-0* [*simp*]: *ord-rec T G H 0* = *T*
⟨*proof*⟩

lemma *ord-recp-total*: $\exists\, y$. *ord-recp T G H x y*
⟨*proof*⟩

lemma *ord-rec-succ* [*simp*]:
assumes *k*: *Ord k* **shows** *ord-rec T G H* (*succ k*) = *G* (*ord-rec T G H k*)
⟨*proof*⟩

lemma *ord-rec-non* [*simp*]: $\neg$ *Ord x* $\Longrightarrow$ *ord-rec T G H x* = *H x*
⟨*proof*⟩

end

Chapter 3

V-Sets, Epsilon Closure, Ranks

theory *Rank* **imports** *Ordinal*
begin

3.1 V-sets

Definition 4.1

definition $Vset :: hf \Rightarrow hf$
  where $Vset\ x = ord\text{-}rec\ 0\ HPow\ (\lambda z.\ 0)\ x$

lemma *Vset-0* [*simp*]: $Vset\ 0 = 0$
  ⟨*proof*⟩

lemma *Vset-succ* [*simp*]: $Ord\ k \Longrightarrow Vset\ (succ\ k) = HPow\ (Vset\ k)$
  ⟨*proof*⟩

lemma *Vset-non* [*simp*]: $\neg\ Ord\ x \Longrightarrow Vset\ x = 0$
  ⟨*proof*⟩

Theorem 4.2(a)

lemma *Vset-mono-strict*:
  assumes $Ord\ m\ n \in m$ **shows** $Vset\ n < Vset\ m$
⟨*proof*⟩

lemma *Vset-mono*: $[\![Ord\ m;\ n \le m]\!] \Longrightarrow Vset\ n \le Vset\ m$
  ⟨*proof*⟩

Theorem 4.2(b)

lemma *Vset-Transset*: $Ord\ m \Longrightarrow Transset\ (Vset\ m)$
  ⟨*proof*⟩

lemma *Ord-sup* [*simp*]: $Ord\ k \Longrightarrow Ord\ l \Longrightarrow Ord\ (k \sqcup l)$

⟨*proof*⟩

lemma *Ord-inf* [*simp*]: $Ord\ k \Longrightarrow Ord\ l \Longrightarrow Ord\ (k \sqcap l)$
⟨*proof*⟩

Theorem 4.3

lemma *Vset-universal*: $\exists\, n.\ Ord\ n \wedge x \in Vset\ n$
⟨*proof*⟩

3.2 Least Ordinal Operator

Definition 4.4. For every x, let rank(x) be the least ordinal n such that...

lemma *Ord-minimal*:
$Ord\ k \Longrightarrow P\ k \Longrightarrow \exists\, n.\ Ord\ n \wedge P\ n \wedge (\forall\, m.\ Ord\ m \wedge P\ m \longrightarrow n \leq m)$
⟨*proof*⟩

lemma *OrdLeastI*: $Ord\ k \Longrightarrow P\ k \Longrightarrow P(LEAST\ n.\ Ord\ n \wedge P\ n)$
⟨*proof*⟩

lemma *OrdLeast-le*: $Ord\ k \Longrightarrow P\ k \Longrightarrow (LEAST\ n.\ Ord\ n \wedge P\ n) \leq k$
⟨*proof*⟩

lemma *OrdLeast-Ord*:
assumes *Ord k P k***shows** $Ord(LEAST\ n.\ Ord\ n \wedge P\ n)$
⟨*proof*⟩

3.3 Rank Function

definition $rank :: hf \Rightarrow hf$
where $rank\ x = (LEAST\ n.\ Ord\ n \wedge x \in Vset\ (succ\ n))$

lemma [*simp*]: $rank\ 0 = 0$
⟨*proof*⟩

lemma *in-Vset-rank*: $a \in Vset(succ(rank\ a))$
⟨*proof*⟩

lemma *Ord-rank* [*simp*]: $Ord\ (rank\ a)$
⟨*proof*⟩

lemma *le-Vset-rank*: $a \leq Vset(rank\ a)$
⟨*proof*⟩

lemma *VsetI*: $succ(rank\ a) \leq k \Longrightarrow Ord\ k \Longrightarrow a \in Vset\ k$
⟨*proof*⟩

lemma *Vset-succ-rank-le*: $Ord\ k \Longrightarrow a \in Vset\ (succ\ k) \Longrightarrow rank\ a \leq k$
⟨*proof*⟩

lemma *Vset-rank-lt*: **assumes** *a*: $a \in$ *Vset k* **shows** *rank a* $<$ *k*
⟨*proof*⟩

Theorem 4.5

theorem *rank-lt*: $a \in b \Longrightarrow$ *rank*$(a) <$ *rank*(b)
⟨*proof*⟩

lemma *rank-mono*: $x \le y \Longrightarrow$ *rank x* $\le$ *rank y*
⟨*proof*⟩

lemma *rank-sup* [*simp*]: *rank* $(a \sqcup b) =$ *rank a* $\sqcup$ *rank b*
⟨*proof*⟩

lemma *rank-singleton* [*simp*]: *rank* $\{\!|a|\!\}$ $=$ *succ*(*rank a*)
⟨*proof*⟩

lemma *rank-hinsert* [*simp*]: *rank* $(b \triangleleft a) =$ *rank b* $\sqcup$ *succ*(*rank a*)
⟨*proof*⟩

Definition 4.6. The transitive closure of x is the minimal transitive set y such that $x \le y$.

3.4 Epsilon Closure

definition
eclose :: *hf* $\Rightarrow$ *hf* **where**
eclose X $= \bigsqcap \{\!| Y \in$ *HPow*(*Vset* (*rank X*)). *Transset Y* $\wedge X \le Y |\!\}$

lemma *eclose-facts*:
shows *Transset-eclose*: *Transset* (*eclose X*)
and *le-eclose*: $X \le$ *eclose X*
⟨*proof*⟩

lemma *eclose-minimal*:
assumes *Y*: *Transset Y* $X \le Y$ **shows** *eclose X* $\le Y$
⟨*proof*⟩

lemma *eclose-0* [*simp*]: *eclose 0 = 0*
⟨*proof*⟩

lemma *eclose-sup* [*simp*]: *eclose* $(a \sqcup b) =$ *eclose a* $\sqcup$ *eclose b*
⟨*proof*⟩

lemma *eclose-singleton* [*simp*]: *eclose* $\{\!|a|\!\}$ $=$ (*eclose a*) $\triangleleft$ *a*
⟨*proof*⟩

lemma *eclose-hinsert* [*simp*]: *eclose* $(b \triangleleft a) =$ *eclose b* $\sqcup$ (*eclose a* $\triangleleft$ *a*)
⟨*proof*⟩

lemma *eclose-succ* [*simp*]: $eclose\ (succ\ a) = eclose\ a \triangleleft a$
⟨*proof*⟩

lemma *fst-in-eclose* [*simp*]: $x \in eclose\ \langle x,\ y\rangle$
⟨*proof*⟩

lemma *snd-in-eclose* [*simp*]: $y \in eclose\ \langle x,\ y\rangle$
⟨*proof*⟩

Theorem 4.7. rank(x) = rank(cl(x)).

lemma *rank-eclose* [*simp*]: $rank\ (eclose\ x) = rank\ x$
⟨*proof*⟩

3.5 Epsilon-Recursion

Theorem 4.9. Definition of a function by recursion on rank.

lemma *hmem-induct* [*case-names step*]:
assumes *ih*: $\bigwedge x.\ (\bigwedge y.\ y \in x \Longrightarrow P\ y) \Longrightarrow P\ x$ **shows** $P\ x$
⟨*proof*⟩

definition
hmem-rel :: (*hf* ∗ *hf*) *set* **where**
$hmem\text{-}rel = trancl\ \{(x,y).\ x \in y\}$

lemma *wf-hmem-rel*: *wf hmem-rel*
⟨*proof*⟩

lemma *hmem-eclose-le*: $y \in x \Longrightarrow eclose\ y \le eclose\ x$
⟨*proof*⟩

lemma *hmem-rel-iff-hmem-eclose*: $(x,y) \in hmem\text{-}rel \longleftrightarrow x \in eclose\ y$
⟨*proof*⟩

definition *hmemrec* :: $((hf \Rightarrow 'a) \Rightarrow hf \Rightarrow 'a) \Rightarrow hf \Rightarrow 'a$ **where**
$hmemrec\ G \equiv wfrec\ hmem\text{-}rel\ G$

definition *ecut* :: $(hf \Rightarrow 'a) \Rightarrow hf \Rightarrow hf \Rightarrow 'a$ **where**
$ecut\ f\ x \equiv (\lambda y.\ if\ y \in eclose\ x\ then\ f\ y\ else\ undefined)$

lemma *hmemrec*: $hmemrec\ G\ a = G\ (ecut\ (hmemrec\ G)\ a)\ a$
⟨*proof*⟩

This form avoids giant explosions in proofs.

lemma *def-hmemrec*: $f \equiv hmemrec\ G \Longrightarrow f\ a = G\ (ecut\ (hmemrec\ G)\ a)\ a$
⟨*proof*⟩

lemma *ecut-apply*: $y \in eclose\ x \Longrightarrow ecut\ f\ x\ y = f\ y$

⟨*proof*⟩

lemma *RepFun-ecut*: $y \leq z \Longrightarrow RepFun\ y\ (ecut\ f\ z) = RepFun\ y\ f$
⟨*proof*⟩

Now, a stronger induction rule, for the transitive closure of membership

lemma *hmem-rel-induct* [*case-names step*]:
assumes *ih*: $\bigwedge x.\ (\bigwedge y.\ (y,x) \in hmem\text{-}rel \Longrightarrow P\ y) \Longrightarrow P\ x$ **shows** $P\ x$
⟨*proof*⟩

lemma *rank-HUnion-less*: $x \neq 0 \Longrightarrow rank\ (\bigsqcup x) < rank\ x$
⟨*proof*⟩

corollary *Sup-ne*: $x \neq 0 \Longrightarrow \bigsqcup x \neq x$
⟨*proof*⟩

end

Chapter 4

An Application: Finite Automata

theory *Finite-Automata* **imports** *Ordinal*
begin

The point of this example is that the HF sets are closed under disjoint sums and Cartesian products, allowing the theory of finite state machines to be developed without issues of polymorphism or any tricky encodings of states.

record $'a\ fsm = states :: hf$
$init :: hf$
$final :: hf$
$next :: hf \Rightarrow 'a \Rightarrow hf \Rightarrow bool$

inductive $reaches :: ['a\ fsm, hf, 'a\ list, hf] \Rightarrow bool$
where
Nil: $st \in states\ fsm \Longrightarrow reaches\ fsm\ st\ []\ st$
| $Cons$: $[\![next\ fsm\ st\ x\ st''; reaches\ fsm\ st''\ xs\ st'; st \in states\ fsm]\!] \Longrightarrow reaches\ fsm\ st\ (x\#xs)\ st'$

declare *reaches.intros* [*intro*]
inductive-simps *reaches-Nil* [*simp*]: $reaches\ fsm\ st\ []\ st'$
inductive-simps *reaches-Cons* [*simp*]: $reaches\ fsm\ st\ (x\#xs)\ st'$

lemma *reaches-imp-states*: $reaches\ fsm\ st\ xs\ st' \Longrightarrow st \in states\ fsm \wedge st' \in states\ fsm$
$\langle proof \rangle$

lemma *reaches-append-iff*:
$reaches\ fsm\ st\ (xs@ys)\ st' \longleftrightarrow (\exists\, st''.\ reaches\ fsm\ st\ xs\ st'' \wedge reaches\ fsm\ st''\ ys\ st')$
$\langle proof \rangle$

definition $accepts :: 'a\ fsm \Rightarrow 'a\ list \Rightarrow bool$ **where**

$accepts\ fsm\ xs \equiv \exists\, st\ st'.\ reaches\ fsm\ st\ xs\ st' \wedge st \in init\ fsm \wedge st' \in final\ fsm$

definition *regular* :: *'a list set* $\Rightarrow$ *bool* **where**
$regular\ S \equiv \exists fsm.\ S = \{xs.\ accepts\ fsm\ xs\}$

definition *Null* **where**
$Null = (\!|states = 0,\ init = 0,\ final = 0,\ next = \lambda st\ x\ st'.\ False|\!)$

theorem *regular-empty*: *regular* $\{\}$
$\langle proof \rangle$

abbreviation *NullStr* **where**
$NullStr \equiv (\!|states = 1,\ init = 1,\ final = 1,\ next = \lambda st\ x\ st'.\ False|\!)$

theorem *regular-emptystr*: *regular* $\{[]\}$
$\langle proof \rangle$

abbreviation *SingStr* **where**
$SingStr\ a \equiv (\!|states = \{|0,\ 1|\},\ init = \{|0|\},\ final = \{|1|\},\ next = \lambda st\ x\ st'.\ st{=}0 \wedge x{=}a \wedge st'{=}1|\!)$

theorem *regular-singstr*: *regular* $\{[a]\}$
$\langle proof \rangle$

definition *Reverse* **where**
$Reverse\ fsm = (\!|states = states\ fsm,\ init = final\ fsm,\ final = init\ fsm,$
$next = \lambda st\ x\ st'.\ next\ fsm\ st'\ x\ st|\!)$

lemma *Reverse-Reverse-ident* [*simp*]: $Reverse\ (Reverse\ fsm) = fsm$
$\langle proof \rangle$

lemma *reaches-Reverse-iff* [*simp*]:
$reaches\ (Reverse\ fsm)\ st\ (rev\ xs)\ st' \longleftrightarrow reaches\ fsm\ st'\ xs\ st$
$\langle proof \rangle$

lemma *reaches-Reverse-iff2* [*simp*]:
$reaches\ (Reverse\ fsm)\ st'\ xs\ st \longleftrightarrow reaches\ fsm\ st\ (rev\ xs)\ st'$
$\langle proof \rangle$

lemma [*simp*]: $init\ (Reverse\ fsm) = final\ fsm$
$\langle proof \rangle$

lemma [*simp*]: $final\ (Reverse\ fsm) = init\ fsm$
$\langle proof \rangle$

lemma *accepts-Reverse*: $rev\ `\ \{xs.\ accepts\ fsm\ xs\} = \{xs.\ accepts\ (Reverse\ fsm)\ xs\}$
$\langle proof \rangle$

theorem *regular-rev*: *regular* $S \Longrightarrow$ *regular* (*rev* ' S)
⟨*proof*⟩

definition *Times* **where**
Times fsm1 fsm2 = (|*states* = *states fsm1* ∗ *states fsm2*,
init = *init fsm1* ∗ *init fsm2*,
final = *final fsm1* ∗ *final fsm2*,
next = λ*st x st'*. (∃ *st1 st2 st1' st2'*. *st* = ⟨*st1*,*st2*⟩ ∧ *st'* = ⟨*st1'*,*st2'*⟩ ∧
next fsm1 st1 x st1' ∧ *next fsm2 st2 x st2'*)|)

lemma *states-Times* [*simp*]: *states* (*Times fsm1 fsm2*) = *states fsm1* ∗ *states fsm2*
⟨*proof*⟩

lemma *init-Times* [*simp*]: *init* (*Times fsm1 fsm2*) = *init fsm1* ∗ *init fsm2*
⟨*proof*⟩

lemma *final-Times* [*simp*]: *final* (*Times fsm1 fsm2*) = *final fsm1* ∗ *final fsm2*
⟨*proof*⟩

lemma *next-Times*: *next* (*Times fsm1 fsm2*) ⟨*st1*,*st2*⟩ *x st'* ⟷
(∃ *st1' st2'*. *st'* = ⟨*st1'*,*st2'*⟩ ∧ *next fsm1 st1 x st1'* ∧ *next fsm2 st2 x st2'*)
⟨*proof*⟩

lemma *reaches-Times-iff* [*simp*]:
reaches (*Times fsm1 fsm2*) ⟨*st1*,*st2*⟩ *xs* ⟨*st1'*,*st2'*⟩ ⟷
reaches fsm1 st1 xs st1' ∧ *reaches fsm2 st2 xs st2'*
⟨*proof*⟩

lemma *accepts-Times-iff* [*simp*]:
accepts (*Times fsm1 fsm2*) *xs* ⟷ *accepts fsm1 xs* ∧ *accepts fsm2 xs*
⟨*proof*⟩

theorem *regular-Int*:
assumes S: *regular* S **and** T: *regular* T **shows** *regular* ($S \cap T$)
⟨*proof*⟩

definition *Plus* **where**
Plus fsm1 fsm2 = (|*states* = *states fsm1* + *states fsm2*,
init = *init fsm1* + *init fsm2*,
final = *final fsm1* + *final fsm2*,
next = λ*st x st'*. (∃ *st1 st1'*. *st* = *Inl st1* ∧ *st'* = *Inl st1'* ∧ *next fsm1 st1 x st1'*) ∨
(∃ *st2 st2'*. *st* = *Inr st2* ∧ *st'* = *Inr st2'* ∧ *next fsm2 st2 x st2'*)|)

lemma *states-Plus* [*simp*]: *states* (*Plus fsm1 fsm2*) = *states fsm1* + *states fsm2*
⟨*proof*⟩

lemma *init-Plus* [*simp*]: *init* (*Plus fsm1 fsm2*) = *init fsm1* + *init fsm2*
⟨*proof*⟩

lemma *final-Plus* [*simp*]: *final* (*Plus fsm1 fsm2*) = *final fsm1* + *final fsm2*
⟨*proof*⟩

lemma *next-Plus1*: *next* (*Plus fsm1 fsm2*) (*Inl st1*) *x st'* ⟷ (∃ *st1'*. *st'* = *Inl st1'* ∧ *next fsm1 st1 x st1'*)
⟨*proof*⟩

lemma *next-Plus2*: *next* (*Plus fsm1 fsm2*) (*Inr st2*) *x st'* ⟷ (∃ *st2'*. *st'* = *Inr st2'* ∧ *next fsm2 st2 x st2'*)
⟨*proof*⟩

lemma *reaches-Plus-iff1* [*simp*]:
reaches (*Plus fsm1 fsm2*) (*Inl st1*) *xs st'* ⟷
(∃ *st1'*. *st'* = *Inl st1'* ∧ *reaches fsm1 st1 xs st1'*)
⟨*proof*⟩

lemma *reaches-Plus-iff2* [*simp*]:
reaches (*Plus fsm1 fsm2*) (*Inr st2*) *xs st'* ⟷
(∃ *st2'*. *st'* = *Inr st2'* ∧ *reaches fsm2 st2 xs st2'*)
⟨*proof*⟩

lemma *reaches-Plus-iff* [*simp*]:
reaches (*Plus fsm1 fsm2*) *st xs st'* ⟷
(∃ *st1 st1'*. *st* = *Inl st1* ∧ *st'* = *Inl st1'* ∧ *reaches fsm1 st1 xs st1'*) ∨
(∃ *st2 st2'*. *st* = *Inr st2* ∧ *st'* = *Inr st2'* ∧ *reaches fsm2 st2 xs st2'*)
⟨*proof*⟩

lemma *accepts-Plus-iff* [*simp*]:
accepts (*Plus fsm1 fsm2*) *xs* ⟷ *accepts fsm1 xs* ∨ *accepts fsm2 xs*
⟨*proof*⟩

lemma *regular-Un*:
assumes *S*: *regular S* **and** *T*: *regular T* **shows** *regular* (*S* ∪ *T*)
⟨*proof*⟩

end
theory *Finitary*
imports *Ordinal*
begin

class *finitary* =
fixes *hf-of* :: *'a* ⇒ *hf*
assumes *inj*: *inj hf-of*
begin
lemma *hf-of-eq-iff* [*simp*]: *hf-of x* = *hf-of y* ⟷ *x*=*y*
⟨*proof*⟩

end

instantiation *unit* :: *finitary*
begin
definition *hf-of-unit-def*: $hf\text{-}of\ (u{::}unit) \equiv 0$
instance
⟨*proof*⟩
end

instantiation *bool* :: *finitary*
begin
definition *hf-of-bool-def*: $hf\text{-}of\ b \equiv if\ b\ then\ 1\ else\ 0$
instance
⟨*proof*⟩
end

instantiation *nat* :: *finitary*
begin
definition *hf-of-nat-def*: $hf\text{-}of \equiv ord\text{-}of$
instance
⟨*proof*⟩
end

instantiation *int* :: *finitary*
begin
definition *hf-of-int-def*:
$hf\text{-}of\ i \equiv if\ i{\geq}(0{::}int)\ then\ \langle 0,\ hf\text{-}of\ (nat\ i)\rangle\ else\ \langle 1,\ hf\text{-}of\ (nat\ (-i))\rangle$
instance
⟨*proof*⟩
end

Strings are char lists, and are not considered separately.

instantiation *char* :: *finitary*
begin
definition *hf-of-char-def*:
$hf\text{-}of\ x \equiv hf\text{-}of\ (of\text{-}char\ x\ ::\ nat)$
instance
⟨*proof*⟩
end

instantiation *prod* :: (*finitary*,*finitary*) *finitary*
begin
definition *hf-of-prod-def*:
$hf\text{-}of \equiv \lambda(x,y).\ \langle hf\text{-}of\ x,\ hf\text{-}of\ y\rangle$
instance
⟨*proof*⟩
end

instantiation *sum* :: (*finitary*,*finitary*) *finitary*

begin
  definition *hf-of-sum-def*:
    $hf\text{-}of \equiv case\text{-}sum\ (HF.Inl\ o\ hf\text{-}of)\ (HF.Inr\ o\ hf\text{-}of)$
  instance
    ⟨*proof*⟩
end

instantiation *option* :: (*finitary*) *finitary*
begin
  definition *hf-of-option-def*:
    $hf\text{-}of \equiv case\text{-}option\ 0\ (\lambda x.\ \{\!|hf\text{-}of\ x|\!\})$
  instance
    ⟨*proof*⟩
end

instantiation *list* :: (*finitary*) *finitary*
begin
  primrec *hf-of-list* **where**
    $hf\text{-}of\text{-}list\ Nil = 0$
  $|\ hf\text{-}of\text{-}list\ (x\#xs) = \langle hf\text{-}of\ x,\ hf\text{-}of\text{-}list\ xs\rangle$

lemma [*simp*]: **fixes** x :: *'a list* **shows** $hf\text{-}of\ x = hf\text{-}of\ y \Longrightarrow x = y$
  ⟨*proof*⟩

instance
  ⟨*proof*⟩
end

end

Chapter 5

Addition, Sequences and their Concatenation

theory *OrdArith* **imports** *Rank*
begin

5.1 Generalised Addition — Also for Ordinals

Source: Laurence Kirby, Addition and multiplication of sets Math. Log. Quart. 53, No. 1, 52-65 (2007) / DOI 10.1002/malq.200610026 http://faculty.baruch.cuny.edu/lkirby/mlqarticlejan2007.pdf

definition
  hadd :: $hf \Rightarrow hf \Rightarrow hf$ (**infixl** ‹@+› *65*) **where**
    $hadd\ x \equiv hmemrec\ (\lambda f\ z.\ x \sqcup RepFun\ z\ f)$

lemma *hadd*: x @+ $y = x \sqcup RepFun\ y\ (\lambda z.\ x$ @+ $z)$
  ⟨*proof*⟩

lemma *hmem-hadd-E*:
  assumes l: $l \in x$ @+ y
  obtains $l \in x \mid z$ **where** $z \in y$ $l = x$ @+ z
  ⟨*proof*⟩

lemma *hadd-0-right* [*simp*]: x @+ $0 = x$
  ⟨*proof*⟩

lemma *hadd-hinsert-right*: x @+ $hinsert\ y\ z = hinsert\ (x$ @+ $y)\ (x$ @+ $z)$
  ⟨*proof*⟩

lemma *hadd-succ-right* [*simp*]: x @+ $succ\ y = succ\ (x$ @+ $y)$
  ⟨*proof*⟩

lemma *not-add-less-right*: $\neg\ (x$ @+ $y < x)$
⟨*proof*⟩

lemma *not-add-mem-right*: $\neg$ $(x$ @+ $y \in x)$
⟨*proof*⟩

lemma *hadd-0-left* [*simp*]: *0* @+ $x = x$
⟨*proof*⟩

lemma *hadd-succ-left* [*simp*]: *Ord* $y \Longrightarrow$ *succ* x @+ y = *succ* $(x$ @+ $y)$
⟨*proof*⟩

lemma *hadd-assoc*: $(x$ @+ $y)$ @+ $z = x$ @+ $(y$ @+ $z)$
⟨*proof*⟩

lemma *RepFun-hadd-disjoint*: $x \sqcap$ *RepFun* y ((@+) x) = *0*
⟨*proof*⟩

5.1.1 Cancellation laws for addition

lemma *Rep-le-Cancel*: $x \sqcup$ *RepFun* y ((@+) x) $\leq x \sqcup$ *RepFun* z ((@+) x)
$\Longrightarrow$ *RepFun* y ((@+) x) $\leq$ *RepFun* z ((@+) x)
⟨*proof*⟩

lemma *hadd-cancel-right* [*simp*]: x @+ $y = x$ @+ $z \longleftrightarrow y=z$
⟨*proof*⟩

lemma *RepFun-hadd-cancel*: *RepFun* y ($\lambda z.$ x @+ z) = *RepFun* z ($\lambda z.$ x @+ z) $\longleftrightarrow y=z$
⟨*proof*⟩

lemma *hadd-hmem-cancel* [*simp*]: x @+ $y \in x$ @+ $z \longleftrightarrow y \in z$
⟨*proof*⟩

lemma *ord-of-add*: *ord-of* $(i+j)$ = *ord-of* i @+ *ord-of* j
⟨*proof*⟩

lemma *Ord-hadd*: *Ord* $x \Longrightarrow$ *Ord* $y \Longrightarrow$ *Ord* $(x$ @+ $y)$
⟨*proof*⟩

lemma *hmem-self-hadd* [*simp*]: *k1* $\in$ *k1* @+ *k2* $\longleftrightarrow$ *0* $\in$ *k2*
⟨*proof*⟩

lemma *hadd-commute*: *Ord* $x \Longrightarrow$ *Ord* $y \Longrightarrow x$ @+ $y = y$ @+ x
⟨*proof*⟩

lemma *hadd-cancel-left* [*simp*]: *Ord* $x \Longrightarrow y$ @+ $x = z$ @+ $x \longleftrightarrow y=z$
⟨*proof*⟩

5.1.2 The predecessor function

definition *pred* :: *hf* $\Rightarrow$ *hf*

where $pred\ x \equiv (THE\ y.\ succ\ y = x \vee x{=}0 \wedge y{=}0)$

lemma *pred-succ* [*simp*]: $pred\ (succ\ x) = x$
⟨*proof*⟩

lemma *pred-0* [*simp*]: $pred\ 0 = 0$
⟨*proof*⟩

lemma *succ-pred* [*simp*]: $Ord\ x \Longrightarrow x \neq 0 \Longrightarrow succ\ (pred\ x) = x$
⟨*proof*⟩

lemma *pred-mem* [*simp*]: $Ord\ x \Longrightarrow x \neq 0 \Longrightarrow pred\ x \in x$
⟨*proof*⟩

lemma *Ord-pred* [*simp*]: $Ord\ x \Longrightarrow Ord\ (pred\ x)$
⟨*proof*⟩

lemma *hadd-pred-right*: $Ord\ y \Longrightarrow y \neq 0 \Longrightarrow x\ @{+}\ pred\ y = pred\ (x\ @{+}\ y)$
⟨*proof*⟩

lemma *Ord-pred-HUnion*: $Ord(k) \Longrightarrow pred\ k = \bigsqcup k$
⟨*proof*⟩

5.2 A Concatentation Operation for Sequences

definition *shift* :: $hf \Rightarrow hf \Rightarrow hf$
where $shift\ f\ delta = \{\!|v\ .\ u \in f,\ \exists\, n\ y.\ u = \langle n,\ y\rangle \wedge v = \langle delta\ @{+}\ n,\ y\rangle|\!\}$

lemma *shiftD*: $x \in shift\ f\ delta \Longrightarrow \exists\, u.\ u \in f \wedge x = \langle delta\ @{+}\ hfst\ u,\ hsnd\ u\rangle$
⟨*proof*⟩

lemma *hmem-shift-iff*: $\langle m,\ y\rangle \in shift\ f\ delta \longleftrightarrow (\exists\, n.\ m = delta\ @{+}\ n \wedge \langle n,\ y\rangle \in f)$
⟨*proof*⟩

lemma *hmem-shift-add-iff* [*simp*]: $\langle delta\ @{+}\ n,\ y\rangle \in shift\ f\ delta \longleftrightarrow \langle n,\ y\rangle \in f$
⟨*proof*⟩

lemma *hrelation-shift* [*simp*]: $hrelation\ (shift\ f\ delta)$
⟨*proof*⟩

lemma *app-shift* [*simp*]: $app\ (shift\ f\ k)\ (k\ @{+}\ j) = app\ f\ j$
⟨*proof*⟩

lemma *hfunction-shift-iff* [*simp*]: $hfunction\ (shift\ f\ delta) = hfunction\ f$
⟨*proof*⟩

lemma *hdomain-shift-add*: $hdomain\ (shift\ f\ delta) = \{\!|delta\ @{+}\ n\ .\ n \in hdomain\ f|\!\}$

⟨*proof*⟩

lemma *hdomain-shift-disjoint*: $delta \sqcap hdomain\ (shift\ f\ delta) = 0$
⟨*proof*⟩

definition *seq-append* :: $hf \Rightarrow hf \Rightarrow hf \Rightarrow hf$
where $seq\text{-}append\ k\ f\ g \equiv hrestrict\ f\ k \sqcup shift\ g\ k$

lemma *hrelation-seq-append* [*simp*]: $hrelation\ (seq\text{-}append\ k\ f\ g)$
⟨*proof*⟩

lemma *Seq-append*:
assumes $Seq\ s1\ k1\ Seq\ s2\ k2$
shows $Seq\ (seq\text{-}append\ k1\ s1\ s2)\ (k1\ @+\ k2)$
⟨*proof*⟩

lemma *app-hunion1*: $x \notin hdomain\ g \Longrightarrow app\ (f \sqcup g)\ x = app\ f\ x$
⟨*proof*⟩

lemma *app-hunion2*: $x \notin hdomain\ f \Longrightarrow app\ (f \sqcup g)\ x = app\ g\ x$
⟨*proof*⟩

lemma *Seq-append-app1*: $Seq\ s\ k \Longrightarrow l \in k \Longrightarrow app\ (seq\text{-}append\ k\ s\ s')\ l = app\ s\ l$
⟨*proof*⟩

lemma *Seq-append-app2*: $Seq\ s1\ k1 \Longrightarrow Seq\ s2\ k2 \Longrightarrow l = k1\ @+\ j \Longrightarrow app\ (seq\text{-}append\ k1\ s1\ s2)\ l = app\ s2\ j$
⟨*proof*⟩

5.3 Nonempty sequences indexed by ordinals

definition *OrdDom* **where**
$OrdDom\ r \equiv \forall x\ y.\ \langle x,y\rangle \in r \longrightarrow Ord\ x$

lemma *OrdDom-insf*: $[\![OrdDom\ s;\ Ord\ k]\!] \Longrightarrow OrdDom\ (insf\ s\ (succ\ k)\ y)$
⟨*proof*⟩

lemma *OrdDom-hunion* [*simp*]: $OrdDom\ (s1 \sqcup s2) \longleftrightarrow OrdDom\ s1 \wedge OrdDom\ s2$
⟨*proof*⟩

lemma *OrdDom-hrestrict*: $OrdDom\ s \Longrightarrow OrdDom\ (hrestrict\ s\ A)$
⟨*proof*⟩

lemma *OrdDom-shift*: $[\![OrdDom\ s;\ Ord\ k]\!] \Longrightarrow OrdDom\ (shift\ s\ k)$
⟨*proof*⟩

A sequence of positive length ending with y

definition $LstSeq :: hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$
where $LstSeq\ s\ k\ y \equiv Seq\ s\ (succ\ k) \wedge Ord\ k \wedge \langle k,y\rangle \in s \wedge OrdDom\ s$

lemma *LstSeq-imp-Seq-succ*: $LstSeq\ s\ k\ y \Longrightarrow Seq\ s\ (succ\ k)$
⟨*proof*⟩

lemma *LstSeq-imp-Seq-same*: $LstSeq\ s\ k\ y \Longrightarrow Seq\ s\ k$
⟨*proof*⟩

lemma *LstSeq-imp-Ord*: $LstSeq\ s\ k\ y \Longrightarrow Ord\ k$
⟨*proof*⟩

lemma *LstSeq-trunc*: $LstSeq\ s\ k\ y \Longrightarrow l \in k \Longrightarrow LstSeq\ s\ l\ (app\ s\ l)$
⟨*proof*⟩

lemma *LstSeq-insf*: $LstSeq\ s\ k\ z \Longrightarrow LstSeq\ (insf\ s\ (succ\ k)\ y)\ (succ\ k)\ y$
⟨*proof*⟩

lemma *app-insf-LstSeq*: $LstSeq\ s\ k\ z \Longrightarrow app\ (insf\ s\ (succ\ k)\ y)\ (succ\ k) = y$
⟨*proof*⟩

lemma *app-insf2-LstSeq*: $LstSeq\ s\ k\ z \Longrightarrow k' \neq succ\ k \Longrightarrow app\ (insf\ s\ (succ\ k)\ y)$
$k' = app\ s\ k'$
⟨*proof*⟩

lemma *app-insf-LstSeq-if*: $LstSeq\ s\ k\ z \Longrightarrow app\ (insf\ s\ (succ\ k)\ y)\ k' = (if\ k' =$
$succ\ k\ then\ y\ else\ app\ s\ k')$
⟨*proof*⟩

lemma *LstSeq-append-app1*:
$LstSeq\ s\ k\ y \Longrightarrow l \in succ\ k \Longrightarrow app\ (seq\text{-}append\ (succ\ k)\ s\ s')\ l = app\ s\ l$
⟨*proof*⟩

lemma *LstSeq-append-app2*:
$[\![LstSeq\ s1\ k1\ y1;\ LstSeq\ s2\ k2\ y2;\ l = succ\ k1\ @+\ j]\!]$
$\Longrightarrow app\ (seq\text{-}append\ (succ\ k1)\ s1\ s2)\ l = app\ s2\ j$
⟨*proof*⟩

lemma *Seq-append-pair*:
$[\![Seq\ s1\ k1;\ Seq\ s2\ (succ\ n);\ \langle n,\ y\rangle \in s2;\ Ord\ n]\!] \Longrightarrow \langle k1\ @+\ n,\ y\rangle \in (seq\text{-}append$
$k1\ s1\ s2)$
⟨*proof*⟩

lemma *Seq-append-OrdDom*: $[\![Ord\ k;\ OrdDom\ s1;\ OrdDom\ s2]\!] \Longrightarrow OrdDom\ (seq\text{-}append$
$k\ s1\ s2)$
⟨*proof*⟩

lemma *LstSeq-append*:
$[\![LstSeq\ s1\ k1\ y1;\ LstSeq\ s2\ k2\ y2]\!] \Longrightarrow LstSeq\ (seq\text{-}append\ (succ\ k1)\ s1\ s2)\ (succ$

$(k1\ @+\ k2))\ y2$
⟨*proof*⟩

lemma *LstSeq-app* [*simp*]: $LstSeq\ s\ k\ y \Longrightarrow app\ s\ k = y$
⟨*proof*⟩

5.3.1 Sequence-building operators

definition *Builds* :: $(hf \Rightarrow bool) \Rightarrow (hf \Rightarrow hf \Rightarrow hf \Rightarrow bool) \Rightarrow hf \Rightarrow hf \Rightarrow bool$
where $Builds\ B\ C\ s\ l \equiv B\ (app\ s\ l) \vee (\exists\, m \in l.\ \exists\, n \in l.\ C\ (app\ s\ l)\ (app\ s\ m)\ (app\ s\ n))$

definition *BuildSeq* :: $(hf \Rightarrow bool) \Rightarrow (hf \Rightarrow hf \Rightarrow hf \Rightarrow bool) \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$
where $BuildSeq\ B\ C\ s\ k\ y \equiv LstSeq\ s\ k\ y \wedge (\forall\, l \in succ\ k.\ Builds\ B\ C\ s\ l)$

lemma *BuildSeqI*: $LstSeq\ s\ k\ y \Longrightarrow (\bigwedge l.\ l \in succ\ k \Longrightarrow Builds\ B\ C\ s\ l) \Longrightarrow BuildSeq\ B\ C\ s\ k\ y$
⟨*proof*⟩

lemma *BuildSeq-imp-LstSeq*: $BuildSeq\ B\ C\ s\ k\ y \Longrightarrow LstSeq\ s\ k\ y$
⟨*proof*⟩

lemma *BuildSeq-imp-Seq*: $BuildSeq\ B\ C\ s\ k\ y \Longrightarrow Seq\ s\ (succ\ k)$
⟨*proof*⟩

lemma *BuildSeq-conj-distrib*:
$BuildSeq\ (\lambda x.\ B\ x \wedge P\ x)\ (\lambda x\ y\ z.\ C\ x\ y\ z \wedge P\ x)\ s\ k\ y \longleftrightarrow$
$BuildSeq\ B\ C\ s\ k\ y \wedge (\forall\, l \in succ\ k.\ P\ (app\ s\ l))$
⟨*proof*⟩

lemma *BuildSeq-mono*:
assumes y: $BuildSeq\ B\ C\ s\ k\ y$
and B: $\bigwedge x.\ B\ x \Longrightarrow B'\ x$ **and** C: $\bigwedge x\ y\ z.\ C\ x\ y\ z \Longrightarrow C'\ x\ y\ z$
shows $BuildSeq\ B'\ C'\ s\ k\ y$
⟨*proof*⟩

lemma *BuildSeq-trunc*:
assumes b: $BuildSeq\ B\ C\ s\ k\ y$
and l: $l \in k$
shows $BuildSeq\ B\ C\ s\ l\ (app\ s\ l)$
⟨*proof*⟩

5.3.2 Showing that Sequences can be Constructed

lemma *Builds-insf*: $Builds\ B\ C\ s\ l \Longrightarrow LstSeq\ s\ k\ z \Longrightarrow l \in succ\ k \Longrightarrow Builds\ B\ C\ (insf\ s\ (succ\ k)\ y)\ l$
⟨*proof*⟩

lemma *BuildSeq-insf*:

assumes b: $BuildSeq\ B\ C\ s\ k\ z$
and m: $m \in succ\ k$
and n: $n \in succ\ k$
and y: $B\ y \vee C\ y\ (app\ s\ m)\ (app\ s\ n)$
shows $BuildSeq\ B\ C\ (insf\ s\ (succ\ k)\ y)\ (succ\ k)\ y$
$\langle proof \rangle$

lemma *BuildSeq-insf1*:
assumes b: $BuildSeq\ B\ C\ s\ k\ z$
and y: $B\ y$
shows $BuildSeq\ B\ C\ (insf\ s\ (succ\ k)\ y)\ (succ\ k)\ y$
$\langle proof \rangle$

lemma *BuildSeq-insf2*:
assumes b: $BuildSeq\ B\ C\ s\ k\ z$
and m: $m \in k$
and n: $n \in k$
and y: $C\ y\ (app\ s\ m)\ (app\ s\ n)$
shows $BuildSeq\ B\ C\ (insf\ s\ (succ\ k)\ y)\ (succ\ k)\ y$
$\langle proof \rangle$

lemma *BuildSeq-append*:
assumes $s1$: $BuildSeq\ B\ C\ s1\ k1\ y1$ **and** $s2$: $BuildSeq\ B\ C\ s2\ k2\ y2$
shows $BuildSeq\ B\ C\ (seq\text{-}append\ (succ\ k1)\ s1\ s2)\ (succ\ (k1\ @{+}\ k2))\ y2$
$\langle proof \rangle$

lemma *BuildSeq-combine*:
assumes $b1$: $BuildSeq\ B\ C\ s1\ k1\ y1$ **and** $b2$: $BuildSeq\ B\ C\ s2\ k2\ y2$
and y: $C\ y\ y1\ y2$
shows $BuildSeq\ B\ C\ (insf\ (seq\text{-}append\ (succ\ k1)\ s1\ s2)\ (succ\ (succ\ (k1\ @{+}\ k2)))$
$y)\ (succ\ (succ\ (k1\ @{+}\ k2)))\ y$
$\langle proof \rangle$

lemma *LstSeq-1*: $LstSeq\ \{\!|\langle 0,\ y\rangle|\!\}\ 0\ y$
$\langle proof \rangle$

lemma *BuildSeq-1*: $B\ y \Longrightarrow BuildSeq\ B\ C\ \{\!|\langle 0,\ y\rangle|\!\}\ 0\ y$
$\langle proof \rangle$

lemma *BuildSeq-exI*: $B\ t \Longrightarrow \exists\, s\ k.\ BuildSeq\ B\ C\ s\ k\ t$
$\langle proof \rangle$

5.3.3 Proving Properties of Given Sequences

lemma *BuildSeq-succ-E*:
assumes s: $BuildSeq\ B\ C\ s\ k\ y$
obtains $B\ y$ | $m\ n$ **where** $m \in k\ n \in k\ C\ y\ (app\ s\ m)\ (app\ s\ n)$
$\langle proof \rangle$

lemma *BuildSeq-induct* [*consumes 1*, *case-names B C*]:
assumes *major*: *BuildSeq B C s k a*
and *B*: $\bigwedge x.\ B\ x \Longrightarrow P\ x$
and *C*: $\bigwedge x\ y\ z.\ C\ x\ y\ z \Longrightarrow P\ y \Longrightarrow P\ z \Longrightarrow P\ x$
shows *P a*
⟨*proof*⟩

definition *BuildSeq2* :: $[[hf,hf] \Rightarrow bool,\ [hf,hf,hf,hf,hf,hf] \Rightarrow bool,\ hf,\ hf,\ hf,\ hf] \Rightarrow bool$
where $BuildSeq2\ B\ C\ s\ k\ y\ y' \equiv$
$BuildSeq\ (\lambda p.\ \exists x\ x'.\ p = \langle x,x'\rangle \wedge B\ x\ x')$
$(\lambda p\ q\ r.\ \exists x\ x'\ y\ y'\ z\ z'.\ p = \langle x,x'\rangle \wedge q = \langle y,y'\rangle \wedge r = \langle z,z'\rangle \wedge C\ x\ x'\ y\ y'\ z\ z')$
$s\ k\ \langle y,y'\rangle$

lemma *BuildSeq2-combine*:
assumes *b1*: $BuildSeq2\ B\ C\ s1\ k1\ y1\ y1'$ **and** *b2*: $BuildSeq2\ B\ C\ s2\ k2\ y2\ y2'$
and *y*: $C\ y\ y'\ y1\ y1'\ y2\ y2'$
shows $BuildSeq2\ B\ C\ (insf\ (seq\text{-}append\ (succ\ k1)\ s1\ s2)\ (succ\ (succ\ (k1\ @+\ k2)))\ \langle y,\ y'\rangle)$
$(succ\ (succ\ (k1\ @+\ k2)))\ y\ y'$
⟨*proof*⟩

lemma *BuildSeq2-1*: $B\ y\ y' \Longrightarrow BuildSeq2\ B\ C\ \{\!\vert\langle 0,\ y,\ y'\rangle\vert\!\}\ 0\ y\ y'$
⟨*proof*⟩

lemma *BuildSeq2-exI*: $B\ t\ t' \Longrightarrow \exists s\ k.\ BuildSeq2\ B\ C\ s\ k\ t\ t'$
⟨*proof*⟩

lemma *BuildSeq2-induct* [*consumes 1*, *case-names B C*]:
assumes $BuildSeq2\ B\ C\ s\ k\ a\ a'$
and *B*: $\bigwedge x\ x'.\ B\ x\ x' \Longrightarrow P\ x\ x'$
and *C*: $\bigwedge x\ x'\ y\ y'\ z\ z'.\ C\ x\ x'\ y\ y'\ z\ z' \Longrightarrow P\ y\ y' \Longrightarrow P\ z\ z' \Longrightarrow P\ x\ x'$
shows $P\ a\ a'$
⟨*proof*⟩

definition *BuildSeq3*
:: $[[hf,hf,hf] \Rightarrow bool,\ [hf,hf,hf,hf,hf,hf,hf,hf,hf] \Rightarrow bool,\ hf,\ hf,\ hf,\ hf,\ hf] \Rightarrow bool$
where $BuildSeq3\ B\ C\ s\ k\ y\ y'\ y'' \equiv$
$BuildSeq\ (\lambda p.\ \exists x\ x'\ x''.\ p = \langle x,x',x''\rangle \wedge B\ x\ x'\ x'')$
$(\lambda p\ q\ r.\ \exists x\ x'\ x''\ y\ y'\ y''\ z\ z'\ z''.$
$p = \langle x,x',x''\rangle \wedge q = \langle y,y',y''\rangle \wedge r = \langle z,z',z''\rangle \wedge$
$C\ x\ x'\ x''\ y\ y'\ y''\ z\ z'\ z'')$
$s\ k\ \langle y,y',y''\rangle$

lemma *BuildSeq3-combine*:
assumes *b1*: $BuildSeq3\ B\ C\ s1\ k1\ y1\ y1'\ y1''$ **and** *b2*: $BuildSeq3\ B\ C\ s2\ k2\ y2\ y2'\ y2''$

and *y*: $C\ y\ y'\ y''\ y1\ y1'\ y1''\ y2\ y2'\ y2''$
shows *BuildSeq3 B C (insf (seq-append (succ k1) s1 s2) (succ (succ (k1 @+ k2))) ⟨y, y', y''⟩)*
(succ (succ (k1 @+ k2))) y y' y''
⟨*proof*⟩

lemma *BuildSeq3-1*: $B\ y\ y'\ y'' \Longrightarrow$ *BuildSeq3 B C* $\{\!|\langle 0, y, y', y''\rangle|\!\}$ $0\ y\ y'\ y''$
⟨*proof*⟩

lemma *BuildSeq3-exI*: $B\ t\ t'\ t'' \Longrightarrow \exists s\ k.$ *BuildSeq3 B C s k t t' t''*
⟨*proof*⟩

lemma *BuildSeq3-induct* [*consumes 1*, *case-names B C*]:
assumes *BuildSeq3 B C s k a a' a''*
and *B*: $\bigwedge x\ x'\ x''.\ B\ x\ x'\ x'' \Longrightarrow P\ x\ x'\ x''$
and *C*: $\bigwedge x\ x'\ x''\ y\ y'\ y''\ z\ z'\ z''.\ C\ x\ x'\ x''\ y\ y'\ y''\ z\ z'\ z'' \Longrightarrow P\ y\ y'\ y'' \Longrightarrow P\ z\ z'\ z'' \Longrightarrow P\ x\ x'\ x''$
shows $P\ a\ a'\ a''$
⟨*proof*⟩

5.4 A Unique Predecessor for every non-empty set

lemma *Rep-hf-0* [*simp*]: *Rep-hf 0 = 0*
⟨*proof*⟩

lemma *hmem-imp-less*: $x \in y \Longrightarrow$ *Rep-hf x* $<$ *Rep-hf y*
⟨*proof*⟩

lemma *hsubset-imp-le*:
assumes $x \leq y$ **shows** *Rep-hf x* $\leq$ *Rep-hf y*
⟨*proof*⟩

lemma *diff-hmem-imp-less*: **assumes** $x \in y$ **shows** *Rep-hf* $(y - \{\!|x|\!\})$ $<$ *Rep-hf y*
⟨*proof*⟩

definition *least* :: $hf \Rightarrow hf$
where *least a* $\equiv$ (*THE x.* $x \in a \wedge (\forall y.\ y \in a \longrightarrow$ *Rep-hf x* $\leq$ *Rep-hf y*))

lemma *least-equality*:
assumes $x \in a$ **and** $\bigwedge y.\ y \in a \Longrightarrow$ *Rep-hf x* $\leq$ *Rep-hf y*
shows *least a = x*
⟨*proof*⟩

lemma *leastI2-order*:
assumes $x \in a$
and $\bigwedge y.\ y \in a \Longrightarrow$ *Rep-hf x* $\leq$ *Rep-hf y*
and $\bigwedge z.\ z \in a \Longrightarrow \forall y.\ y \in a \longrightarrow$ *Rep-hf z* $\leq$ *Rep-hf y* $\Longrightarrow Q\ z$
shows *Q (least a)*
⟨*proof*⟩

lemma *nonempty-imp-ex-least*: $a \neq 0 \Longrightarrow \exists x.\ x \in a \wedge (\forall y.\ y \in a \longrightarrow Rep\text{-}hf\ x \leq Rep\text{-}hf\ y)$
⟨*proof*⟩

lemma *least-hmem*: $a \neq 0 \Longrightarrow least\ a \in a$
⟨*proof*⟩

end

Bibliography

[1] L. Kirby. Addition and multiplication of sets. *Mathematical Logic Quarterly*, 53(1):52–65, 2007.

[2] S. Świerczkowski. Finite sets and Gödel's incompleteness theorems. *Dissertationes Mathematicae*, 422:1–58, 2003. http://journals.impan.gov.pl/dm/Inf/422-0-1.html.