Gödel's Incompleteness Theorems

Lawrence C. Paulson

17. März 2025

Zusammenfassung

Gödel's two incompleteness theorems [2] are formalised, following a careful presentation by Świerczkowski [3], in the theory of hereditarily finite sets. This represents the first ever machine-assisted proof of the second incompleteness theorem. Compared with traditional formalisations using Peano arithmetic [1], coding is simpler, with no need to formalise the notion of multiplication (let alone that of a prime number) in the formalised calculus upon which the theorem is based. However, other technical problems had to be solved in order to complete the argument.

Inhaltsverzeichnis

Kapitel 1

Syntax of Terms and Formulas using Nominal Logic

theory *SyntaxN*
imports *Nominal2.Nominal2 HereditarilyFinite.OrdArith*
begin

1.1 Terms and Formulas

1.1.1 Hf is a pure permutation type

instantiation *hf* :: *pt*
begin
  definition $p \cdot (s::hf) = s$
  instance
    ⟨*proof*⟩
end

instance *hf* :: *pure*
  ⟨*proof*⟩

atom-decl *name*

declare *fresh-set-empty* [*simp*]

lemma *supp-name* [*simp*]: **fixes** *i*::*name* **shows** *supp i* = {*atom i*}
  ⟨*proof*⟩

1.1.2 The datatypes

nominal-datatype *tm* = *Zero* | *Var name* | *Eats tm tm*

nominal-datatype *fm* =
  Mem tm tm (**infixr** ‹*IN*› *150*)
| *Eq tm tm* (**infixr** ‹*EQ*› *150*)
| *Disj fm fm* (**infixr** ‹*OR*› *130*)
| *Neg fm*
| *Ex x*::*name f*::*fm* **binds** *x* **in** *f*

Mem, Eq are atomic formulas; Disj, Neg, Ex are non-atomic

declare *tm.supp* [*simp*] *fm.supp* [*simp*]

1.1.3 Substitution

nominal-function *subst* :: *name* ⇒ *tm* ⇒ *tm* ⇒ *tm*
  where
  subst i x Zero = *Zero*
| *subst i x* (*Var k*) = (*if i*=*k then x else Var k*)
| *subst i x* (*Eats t u*) = *Eats* (*subst i x t*) (*subst i x u*)
⟨*proof*⟩

nominal-termination (*eqvt*)
  ⟨*proof*⟩

lemma *fresh-subst-if* [*simp*]:
  j ♯ *subst i x t* ⟷ (*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
  ⟨*proof*⟩

lemma *forget-subst-tm* [*simp*]: *atom a* ♯ *tm* ⟹ *subst a x tm* = *tm*
  ⟨*proof*⟩

lemma *subst-tm-id* [*simp*]: *subst a* (*Var a*) *tm* = *tm*
  ⟨*proof*⟩

lemma *subst-tm-commute* [*simp*]:
  atom j ♯ *tm* ⟹ *subst j u* (*subst i t tm*) = *subst i* (*subst j u t*) *tm*
  ⟨*proof*⟩

lemma *subst-tm-commute2* [*simp*]:
  atom j ♯ *t* ⟹ *atom i* ♯ *u* ⟹ *i* ≠ *j* ⟹ *subst j u* (*subst i t tm*) = *subst i t* (*subst j u tm*)
  ⟨*proof*⟩

lemma *repeat-subst-tm* [*simp*]: *subst i u* (*subst i t tm*) = *subst i* (*subst i u t*) *tm*
  ⟨*proof*⟩

nominal-function *subst-fm* :: *fm* ⇒ *name* ⇒ *tm* ⇒ *fm* (‹-'(-::=-')› [*1000*, *0*, *0*] *200*)
  where
  Mem: (*Mem t u*)(*i*::=*x*) = *Mem* (*subst i x t*) (*subst i x u*)
| *Eq*: (*Eq t u*)(*i*::=*x*) = *Eq* (*subst i x t*) (*subst i x u*)
| *Disj*: (*Disj A B*)(*i*::=*x*) = *Disj* (*A*(*i*::=*x*)) (*B*(*i*::=*x*))

| *Neg*: $(Neg\ A)(i::=x) \quad = Neg\ \ (A(i::=x))$
| *Ex*: $atom\ j \sharp (i, x) \Longrightarrow (Ex\ j\ A)(i::=x) = Ex\ j\ (A(i::=x))$
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *size-subst-fm* [*simp*]: $size\ (A(i::=x)) = size\ A$
⟨*proof*⟩

lemma *forget-subst-fm* [*simp*]: $atom\ a \sharp A \Longrightarrow A(a::=x) = A$
⟨*proof*⟩

lemma *subst-fm-id* [*simp*]: $A(a::=Var\ a) = A$
⟨*proof*⟩

lemma *fresh-subst-fm-if* [*simp*]:
$j \sharp (A(i::=x)) \longleftrightarrow (atom\ i \sharp A \wedge j \sharp A) \vee (j \sharp x \wedge (j \sharp A \vee j = atom\ i))$
⟨*proof*⟩

lemma *subst-fm-commute* [*simp*]:
$atom\ j \sharp A \Longrightarrow (A(i::=t))(j::=u) = A(i ::= subst\ j\ u\ t)$
⟨*proof*⟩

lemma *repeat-subst-fm* [*simp*]: $(A(i::=t))(i::=u) = A(i ::= subst\ i\ u\ t)$
⟨*proof*⟩

lemma *subst-fm-Ex-with-renaming*:
$atom\ i' \sharp (A, i, j, t) \Longrightarrow (Ex\ i\ A)(j ::= t) = Ex\ i'\ (((i \leftrightarrow i') \cdot A)(j ::= t))$
⟨*proof*⟩

the simplifier cannot apply the rule above, because it introduces a new variable at the right hand side.

⟨*ML*⟩

1.1.4 Semantics

definition *e0* :: (*name*, *hf*) *finfun* — the null environment
where $e0 \equiv finfun\text{-}const\ 0$

nominal-function *eval-tm* :: (*name*, *hf*) *finfun* ⇒ *tm* ⇒ *hf*
where
eval-tm e Zero = *0*
| $eval\text{-}tm\ e\ (Var\ k) = finfun\text{-}apply\ e\ k$
| $eval\text{-}tm\ e\ (Eats\ t\ u) = eval\text{-}tm\ e\ t \triangleleft eval\text{-}tm\ e\ u$
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

syntax
-EvalTm :: *tm* ⇒ (*name*, *hf*) *finfun* ⇒ *hf* (‹⟦-⟧› [0,1000]1000)

syntax-consts
-EvalTm == *eval-tm*

translations
⟦*tm*⟧*e* == *CONST eval-tm e tm*

nominal-function *eval-fm* :: (*name*, *hf*) *finfun* ⇒ *fm* ⇒ *bool*
where
eval-fm e (*t IN u*) ⟷ ⟦*t*⟧*e* ∈ ⟦*u*⟧*e*
| *eval-fm e* (*t EQ u*) ⟷ ⟦*t*⟧*e* = ⟦*u*⟧*e*
| *eval-fm e* (*A OR B*) ⟷ *eval-fm e A* ∨ *eval-fm e B*
| *eval-fm e* (*Neg A*) ⟷ (~ *eval-fm e A*)
| *atom k* ♯ *e* ⟹ *eval-fm e* (*Ex k A*) ⟷ (∃ *x*. *eval-fm* (*finfun-update e k x*) *A*)
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *eval-tm-rename*:
assumes *atom k′* ♯ *t*
shows ⟦*t*⟧(*finfun-update e k x*) = ⟦(*k′* ↔ *k*) ∙ *t*⟧(*finfun-update e k′ x*)
⟨*proof*⟩

lemma *eval-fm-rename*:
assumes *atom k′* ♯ *A*
shows *eval-fm* (*finfun-update e k x*) *A* = *eval-fm* (*finfun-update e k′ x*) ((*k′* ↔ *k*) ∙ *A*)
⟨*proof*⟩

lemma *better-ex-eval-fm*[*simp*]:
eval-fm e (*Ex k A*) ⟷ (∃ *x*. *eval-fm* (*finfun-update e k x*) *A*)
⟨*proof*⟩

lemma *forget-eval-tm* [*simp*]: *atom i* ♯ *t* ⟹ ⟦*t*⟧(*finfun-update e i x*) = ⟦*t*⟧*e*
⟨*proof*⟩

lemma *forget-eval-fm* [*simp*]:
atom k ♯ *A* ⟹ *eval-fm* (*finfun-update e k x*) *A* = *eval-fm e A*
⟨*proof*⟩

lemma *eval-subst-tm*: ⟦*subst i t u*⟧*e* = ⟦*u*⟧(*finfun-update e i* ⟦*t*⟧*e*)
⟨*proof*⟩

lemma *eval-subst-fm*: *eval-fm e* (*fm*(*i*::= *t*)) = *eval-fm* (*finfun-update e i* ⟦*t*⟧*e*) *fm*
⟨*proof*⟩

1.1.5 Derived syntax

Ordered pairs

definition *HPair* :: *tm* ⇒ *tm* ⇒ *tm*
  where *HPair a b* = *Eats* (*Eats Zero* (*Eats* (*Eats Zero b*) *a*)) (*Eats* (*Eats Zero a*) *a*)

lemma *HPair-eqvt* [*eqvt*]: (*p* ∙ *HPair a b*) = *HPair* (*p* ∙ *a*) (*p* ∙ *b*)
  ⟨*proof*⟩

lemma *fresh-HPair* [*simp*]: *x* ♯ *HPair a b* ⟷ (*x* ♯ *a* ∧ *x* ♯ *b*)
  ⟨*proof*⟩

lemma *HPair-injective-iff* [*iff*]: *HPair a b* = *HPair a′ b′* ⟷ (*a* = *a′* ∧ *b* = *b′*)
  ⟨*proof*⟩

lemma *subst-tm-HPair* [*simp*]: *subst i x* (*HPair a b*) = *HPair* (*subst i x a*) (*subst i x b*)
  ⟨*proof*⟩

lemma *eval-tm-HPair* [*simp*]: ⟦*HPair a b*⟧*e* = *hpair* ⟦*a*⟧*e* ⟦*b*⟧*e*
  ⟨*proof*⟩

Ordinals

definition
  SUCC :: *tm* ⇒ *tm* **where**
    SUCC x ≡ *Eats x x*

fun *ORD-OF* :: *nat* ⇒ *tm*
  where
   ORD-OF 0 = *Zero*
| *ORD-OF* (*Suc k*) = *SUCC* (*ORD-OF k*)

lemma *eval-tm-SUCC* [*simp*]: ⟦*SUCC t*⟧*e* = *succ* ⟦*t*⟧*e*
  ⟨*proof*⟩

lemma *SUCC-fresh-iff* [*simp*]: *a* ♯ *SUCC t* ⟷ *a* ♯ *t*
  ⟨*proof*⟩

lemma *SUCC-eqvt* [*eqvt*]: (*p* ∙ *SUCC a*) = *SUCC* (*p* ∙ *a*)
  ⟨*proof*⟩

lemma *SUCC-subst* [*simp*]: *subst i t* (*SUCC k*) = *SUCC* (*subst i t k*)
  ⟨*proof*⟩

lemma *eval-tm-ORD-OF* [*simp*]: ⟦*ORD-OF n*⟧*e* = *ord-of n*
  ⟨*proof*⟩

lemma *ORD-OF-fresh* [*simp*]: $a \sharp \mathit{ORD\text{-}OF}\ n$
⟨*proof*⟩

lemma *ORD-OF-eqvt* [*eqvt*]: $(p \cdot \mathit{ORD\text{-}OF}\ n) = \mathit{ORD\text{-}OF}\ (p \cdot n)$
⟨*proof*⟩

1.1.6 Derived logical connectives

abbreviation *Imp* :: *fm* ⇒ *fm* ⇒ *fm* (**infixr** ‹*IMP*› *125*)
where $\mathit{Imp}\ A\ B \equiv \mathit{Disj}\ (\mathit{Neg}\ A)\ B$

abbreviation *All* :: *name* ⇒ *fm* ⇒ *fm*
where $\mathit{All}\ i\ A \equiv \mathit{Neg}\ (\mathit{Ex}\ i\ (\mathit{Neg}\ A))$

abbreviation *All2* :: *name* ⇒ *tm* ⇒ *fm* ⇒ *fm* — bounded universal quantifier, for Sigma formulas
where $\mathit{All2}\ i\ t\ A \equiv \mathit{All}\ i\ ((\mathit{Var}\ i\ \mathit{IN}\ t)\ \mathit{IMP}\ A)$

Conjunction

definition *Conj* :: *fm* ⇒ *fm* ⇒ *fm* (**infixr** ‹*AND*› *135*)
where $\mathit{Conj}\ A\ B \equiv \mathit{Neg}\ (\mathit{Disj}\ (\mathit{Neg}\ A)\ (\mathit{Neg}\ B))$

lemma *Conj-eqvt* [*eqvt*]: $p \cdot (A\ \mathit{AND}\ B) = (p \cdot A)\ \mathit{AND}\ (p \cdot B)$
⟨*proof*⟩

lemma *fresh-Conj* [*simp*]: $a \sharp A\ \mathit{AND}\ B \longleftrightarrow (a \sharp A \wedge a \sharp B)$
⟨*proof*⟩

lemma *supp-Conj* [*simp*]: $\mathit{supp}\ (A\ \mathit{AND}\ B) = \mathit{supp}\ A \cup \mathit{supp}\ B$
⟨*proof*⟩

lemma *size-Conj* [*simp*]: $\mathit{size}\ (A\ \mathit{AND}\ B) = \mathit{size}\ A + \mathit{size}\ B + 4$
⟨*proof*⟩

lemma *Conj-injective-iff* [*iff*]: $(A\ \mathit{AND}\ B) = (A'\ \mathit{AND}\ B') \longleftrightarrow (A = A' \wedge B = B')$
⟨*proof*⟩

lemma *subst-fm-Conj* [*simp*]: $(A\ \mathit{AND}\ B)(i::=x) = (A(i::=x))\ \mathit{AND}\ (B(i::=x))$
⟨*proof*⟩

lemma *eval-fm-Conj* [*simp*]: $\mathit{eval\text{-}fm}\ e\ (\mathit{Conj}\ A\ B) \longleftrightarrow (\mathit{eval\text{-}fm}\ e\ A \wedge \mathit{eval\text{-}fm}\ e\ B)$
⟨*proof*⟩

If and only if

definition *Iff* :: *fm* ⇒ *fm* ⇒ *fm* (**infixr** ‹*IFF*› *125*)
where $\mathit{Iff}\ A\ B = \mathit{Conj}\ (\mathit{Imp}\ A\ B)\ (\mathit{Imp}\ B\ A)$

lemma *Iff-eqvt* [*eqvt*]: $p \cdot (A\ IFF\ B) = (p \cdot A)\ IFF\ (p \cdot B)$
⟨*proof*⟩

lemma *fresh-Iff* [*simp*]: $a \sharp A\ IFF\ B \longleftrightarrow (a \sharp A \wedge a \sharp B)$
⟨*proof*⟩

lemma *size-Iff* [*simp*]: $size\ (A\ IFF\ B) = 2{*}(size\ A + size\ B) + 8$
⟨*proof*⟩

lemma *Iff-injective-iff* [*iff*]: $(A\ IFF\ B) = (A'\ IFF\ B') \longleftrightarrow (A = A' \wedge B = B')$
⟨*proof*⟩

lemma *subst-fm-Iff* [*simp*]: $(A\ IFF\ B)(i::=x) = (A(i::=x))\ IFF\ (B(i::=x))$
⟨*proof*⟩

lemma *eval-fm-Iff* [*simp*]: *eval-fm e* $(Iff\ A\ B) \longleftrightarrow$ (*eval-fm e* $A \longleftrightarrow$ *eval-fm e* B)
⟨*proof*⟩

1.2 Axioms and Theorems

1.2.1 Logical axioms

inductive-set *boolean-axioms* :: *fm set*
where
Ident: $A\ IMP\ A \in boolean\text{-}axioms$
| *DisjI1*: $A\ IMP\ (A\ OR\ B) \in boolean\text{-}axioms$
| *DisjCont*: $(A\ OR\ A)\ IMP\ A \in boolean\text{-}axioms$
| *DisjAssoc*: $(A\ OR\ (B\ OR\ C))\ IMP\ ((A\ OR\ B)\ OR\ C) \in boolean\text{-}axioms$
| *DisjConj*: $(C\ OR\ A)\ IMP\ (((Neg\ C)\ OR\ B)\ IMP\ (A\ OR\ B)) \in boolean\text{-}axioms$

lemma *boolean-axioms-hold*: $A \in boolean\text{-}axioms \Longrightarrow$ *eval-fm e A*
⟨*proof*⟩

inductive-set *special-axioms* :: *fm set* **where**
I: $A(i::=x)\ IMP\ (Ex\ i\ A) \in special\text{-}axioms$

lemma *special-axioms-hold*: $A \in special\text{-}axioms \Longrightarrow$ *eval-fm e A*
⟨*proof*⟩

inductive-set *induction-axioms* :: *fm set* **where**
ind:
atom $(j::name) \sharp (i,A)$
$\Longrightarrow A(i::=Zero)\ IMP\ ((All\ i\ (All\ j\ (A\ IMP\ (A(i::=\ Var\ j)\ IMP\ A(i::=\ Eats(Var\ i)(Var\ j))))))$
$IMP\ (All\ i\ A))$
$\in induction\text{-}axioms$

lemma *twist-forget-eval-fm* [*simp*]:
atom j ♯ (*i*, *A*)
⟹ *eval-fm* (*finfun-update* (*finfun-update* (*finfun-update e i x*) *j y*) *i z*) *A* =
eval-fm (*finfun-update e i z*) *A*
⟨*proof*⟩

lemma *induction-axioms-hold*: *A* ∈ *induction-axioms* ⟹ *eval-fm e A*
⟨*proof*⟩

1.2.2 Concrete variables

declare *Abs-name-inject*[*simp*]

abbreviation
X0 ≡ *Abs-name* (*Atom* (*Sort ''SyntaxN.name''* []) *0*)

abbreviation
X1 ≡ *Abs-name* (*Atom* (*Sort ''SyntaxN.name''* []) (*Suc 0*))
— We prefer *Suc 0* because simplification will transform 1 to that form anyway.

abbreviation
X2 ≡ *Abs-name* (*Atom* (*Sort ''SyntaxN.name''* []) *2*)

abbreviation
X3 ≡ *Abs-name* (*Atom* (*Sort ''SyntaxN.name''* []) *3*)

abbreviation
X4 ≡ *Abs-name* (*Atom* (*Sort ''SyntaxN.name''* []) *4*)

1.2.3 The HF axioms

definition *HF1* :: *fm* **where** — the axiom (*z* = *0*) = (∀ *x*. *x* ∉ *z*)
HF1 = (*Var X0 EQ Zero*) *IFF* (*All X1* (*Neg* (*Var X1 IN Var X0*)))

lemma *HF1-holds*: *eval-fm e HF1*
⟨*proof*⟩

definition *HF2* :: *fm* **where** — the axiom (*z* = *x* ◁ *y*) = (∀ *u*. (*u* ∈ *z*) = (*u* ∈ *x* ∨ *u* = *y*))
HF2 ≡ *Var X0 EQ Eats* (*Var X1*) (*Var X2*) *IFF*
All X3 (*Var X3 IN Var X0 IFF Var X3 IN Var X1 OR Var X3 EQ Var X2*)

lemma *HF2-holds*: *eval-fm e HF2*
⟨*proof*⟩

definition *HF-axioms* **where** *HF-axioms* = {*HF1*, *HF2*}

lemma *HF-axioms-hold*: *A* ∈ *HF-axioms* ⟹ *eval-fm e A*

⟨*proof*⟩

1.2.4 Equality axioms

definition *refl-ax* :: *fm* **where**
refl-ax = *Var X1 EQ Var X1*

lemma *refl-ax-holds*: *eval-fm e refl-ax*
⟨*proof*⟩

definition *eq-cong-ax* :: *fm* **where**
eq-cong-ax = ((*Var X1 EQ Var X2*) *AND* (*Var X3 EQ Var X4*)) *IMP*
((*Var X1 EQ Var X3*) *IMP* (*Var X2 EQ Var X4*))

lemma *eq-cong-ax-holds*: *eval-fm e eq-cong-ax*
⟨*proof*⟩

definition *mem-cong-ax* :: *fm* **where**
mem-cong-ax = ((*Var X1 EQ Var X2*) *AND* (*Var X3 EQ Var X4*)) *IMP*
((*Var X1 IN Var X3*) *IMP* (*Var X2 IN Var X4*))

lemma *mem-cong-ax-holds*: *eval-fm e mem-cong-ax*
⟨*proof*⟩

definition *eats-cong-ax* :: *fm* **where**
eats-cong-ax = ((*Var X1 EQ Var X2*) *AND* (*Var X3 EQ Var X4*)) *IMP*
((*Eats* (*Var X1*) (*Var X3*)) *EQ* (*Eats* (*Var X2*) (*Var X4*)))

lemma *eats-cong-ax-holds*: *eval-fm e eats-cong-ax*
⟨*proof*⟩

definition *equality-axioms* :: *fm set* **where**
equality-axioms = {*refl-ax*, *eq-cong-ax*, *mem-cong-ax*, *eats-cong-ax*}

lemma *equality-axioms-hold*: *A* ∈ *equality-axioms* ⟹ *eval-fm e A*
⟨*proof*⟩

1.2.5 The proof system

This arbitrary additional axiom generalises the statements of the incompleteness theorems and other results to any formal system stronger than the HF theory. The additional axiom could be the conjunction of any finite number of assertions. Any more general extension must be a form that can be formalised for the proof predicate.

consts *extra-axiom* :: *fm*

specification (*extra-axiom*)
extra-axiom-holds: *eval-fm e extra-axiom*
⟨*proof*⟩

inductive *hfthm* :: *fm set* ⇒ *fm* ⇒ *bool* (**infixl** ‹⊢› *55*)
where
Hyp: $A \in H \Longrightarrow H \vdash A$
| *Extra*: $H \vdash \textit{extra-axiom}$
| *Bool*: $A \in \textit{boolean-axioms} \Longrightarrow H \vdash A$
| *Eq*: $A \in \textit{equality-axioms} \Longrightarrow H \vdash A$
| *Spec*: $A \in \textit{special-axioms} \Longrightarrow H \vdash A$
| *HF*: $A \in \textit{HF-axioms} \Longrightarrow H \vdash A$
| *Ind*: $A \in \textit{induction-axioms} \Longrightarrow H \vdash A$
| *MP*: $H \vdash A\ \textit{IMP}\ B \Longrightarrow H' \vdash A \Longrightarrow H \cup H' \vdash B$
| *Exists*: $H \vdash A\ \textit{IMP}\ B \Longrightarrow \textit{atom}\ i \sharp B \Longrightarrow \forall C \in H.\ \textit{atom}\ i \sharp C \Longrightarrow H \vdash (\textit{Ex}\ i\ A)\ \textit{IMP}\ B$

Soundness theorem!

theorem *hfthm-sound*: **assumes** $H \vdash A$ **shows** $(\forall B{\in}H.\ \textit{eval-fm}\ e\ B) \Longrightarrow \textit{eval-fm}\ e\ A$
⟨*proof*⟩

1.2.6 Derived rules of inference

lemma *contraction*: $\textit{insert}\ A\ (\textit{insert}\ A\ H) \vdash B \Longrightarrow \textit{insert}\ A\ H \vdash B$
⟨*proof*⟩

lemma *thin-Un*: $H \vdash A \Longrightarrow H \cup H' \vdash A$
⟨*proof*⟩

lemma *thin*: $H \vdash A \Longrightarrow H \subseteq H' \Longrightarrow H' \vdash A$
⟨*proof*⟩

lemma *thin0*: $\{\} \vdash A \Longrightarrow H \vdash A$
⟨*proof*⟩

lemma *thin1*: $H \vdash B \Longrightarrow \textit{insert}\ A\ H \vdash B$
⟨*proof*⟩

lemma *thin2*: $\textit{insert}\ A1\ H \vdash B \Longrightarrow \textit{insert}\ A1\ (\textit{insert}\ A2\ H) \vdash B$
⟨*proof*⟩

lemma *thin3*: $\textit{insert}\ A1\ (\textit{insert}\ A2\ H) \vdash B \Longrightarrow \textit{insert}\ A1\ (\textit{insert}\ A2\ (\textit{insert}\ A3\ H)) \vdash B$
⟨*proof*⟩

lemma *thin4*:
$\textit{insert}\ A1\ (\textit{insert}\ A2\ (\textit{insert}\ A3\ H)) \vdash B$
$\Longrightarrow \textit{insert}\ A1\ (\textit{insert}\ A2\ (\textit{insert}\ A3\ (\textit{insert}\ A4\ H))) \vdash B$
⟨*proof*⟩

lemma *rotate2*: $\textit{insert}\ A2\ (\textit{insert}\ A1\ H) \vdash B \Longrightarrow \textit{insert}\ A1\ (\textit{insert}\ A2\ H) \vdash B$
⟨*proof*⟩

lemma *rotate3*: *insert A3 (insert A1 (insert A2 H)) ⊢ B ⟹ insert A1 (insert A2 (insert A3 H)) ⊢ B*
⟨*proof*⟩

lemma *rotate4*:
insert A4 (insert A1 (insert A2 (insert A3 H))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 H))) ⊢ B
⟨*proof*⟩

lemma *rotate5*:
insert A5 (insert A1 (insert A2 (insert A3 (insert A4 H)))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 H)))) ⊢ B
⟨*proof*⟩

lemma *rotate6*:
insert A6 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 H))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 H))))) ⊢ B
⟨*proof*⟩

lemma *rotate7*:
insert A7 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 H)))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 H)))))) ⊢ B
⟨*proof*⟩

lemma *rotate8*:
insert A8 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 H))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 H))))))) ⊢ B
⟨*proof*⟩

lemma *rotate9*:
insert A9 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 H)))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 H))))))))) ⊢ B
⟨*proof*⟩

lemma *rotate10*:
insert A10 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 H)))))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 H))))))))) ⊢ B
⟨*proof*⟩

lemma *rotate11*:
insert A11 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 H)))))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 H)))))))))) ⊢ B
⟨*proof*⟩

lemma *rotate12*:
insert A12 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 H)))))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 H)))))))))) ⊢ B
⟨*proof*⟩

lemma *rotate13*:
insert A13 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 H)))))))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 (insert A13 H)))))))))))) ⊢ B
⟨*proof*⟩

lemma *rotate14*:
insert A14 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 (insert A13 H)))))))))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 (insert A13 (insert A14 H)))))))))))))) ⊢ B
⟨*proof*⟩

lemma *rotate15*:
insert A15 (insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 (insert A13 (insert A14 H)))))))))))))) ⊢ B
⟹ insert A1 (insert A2 (insert A3 (insert A4 (insert A5 (insert A6 (insert A7 (insert A8 (insert A9 (insert A10 (insert A11 (insert A12 (insert A13 (insert A14 (insert A15 H)))))))))))))) ⊢ B
⟨*proof*⟩

lemma *MP-same*: *H ⊢ A IMP B ⟹ H ⊢ A ⟹ H ⊢ B*
⟨*proof*⟩

lemma *MP-thin*: *HA ⊢ A IMP B ⟹ HB ⊢ A ⟹ HA ∪ HB ⊆ H ⟹ H ⊢ B*
⟨*proof*⟩

lemma *MP-null*: *{} ⊢ A IMP B ⟹ H ⊢ A ⟹ H ⊢ B*
⟨*proof*⟩

lemma *Disj-commute*: $H \vdash B\ OR\ A \Longrightarrow H \vdash A\ OR\ B$
⟨*proof*⟩

lemma *S*: **assumes** $H \vdash A\ IMP\ (B\ IMP\ C)\ H' \vdash A\ IMP\ B$ **shows** $H \cup H' \vdash A\ IMP\ C$
⟨*proof*⟩

lemma *Assume*: $insert\ A\ H \vdash A$
⟨*proof*⟩

lemmas *AssumeH* = *Assume Assume* [*THEN rotate2*] *Assume* [*THEN rotate3*] *Assume* [*THEN rotate4*] *Assume* [*THEN rotate5*]
Assume [*THEN rotate6*] *Assume* [*THEN rotate7*] *Assume* [*THEN rotate8*] *Assume* [*THEN rotate9*] *Assume* [*THEN rotate10*]
Assume [*THEN rotate11*] *Assume* [*THEN rotate12*]
declare *AssumeH* [*intro*!]

lemma *Imp-triv-I*: $H \vdash B \Longrightarrow H \vdash A\ IMP\ B$
⟨*proof*⟩

lemma *DisjAssoc1*: $H \vdash A\ OR\ (B\ OR\ C) \Longrightarrow H \vdash (A\ OR\ B)\ OR\ C$
⟨*proof*⟩

lemma *DisjAssoc2*: $H \vdash (A\ OR\ B)\ OR\ C \Longrightarrow H \vdash A\ OR\ (B\ OR\ C)$
⟨*proof*⟩

lemma *Disj-commute-Imp*: $H \vdash (B\ OR\ A)\ IMP\ (A\ OR\ B)$
⟨*proof*⟩

lemma *Disj-Semicong-1*: $H \vdash A\ OR\ C \Longrightarrow H \vdash A\ IMP\ B \Longrightarrow H \vdash B\ OR\ C$
⟨*proof*⟩

lemma *Imp-Imp-commute*: $H \vdash B\ IMP\ (A\ IMP\ C) \Longrightarrow H \vdash A\ IMP\ (B\ IMP\ C)$
⟨*proof*⟩

1.2.7 The Deduction Theorem

lemma *deduction-Diff*: **assumes** $H \vdash B$ **shows** $H - \{C\} \vdash C\ IMP\ B$
⟨*proof*⟩

theorem *Imp-I* [*intro*!]: $insert\ A\ H \vdash B \Longrightarrow H \vdash A\ IMP\ B$
⟨*proof*⟩

lemma *anti-deduction*: $H \vdash A\ IMP\ B \Longrightarrow insert\ A\ H \vdash B$
⟨*proof*⟩

1.2.8 Cut rules

lemma *cut*: $H \vdash A \Longrightarrow insert\ A\ H' \vdash B \Longrightarrow H \cup H' \vdash B$

⟨*proof*⟩

lemma *cut-same*: $H \vdash A \Longrightarrow \mathit{insert}\ A\ H \vdash B \Longrightarrow H \vdash B$
⟨*proof*⟩

lemma *cut-thin*: $HA \vdash A \Longrightarrow \mathit{insert}\ A\ HB \vdash B \Longrightarrow HA \cup HB \subseteq H \Longrightarrow H \vdash B$
⟨*proof*⟩

lemma *cut0*: $\{\} \vdash A \Longrightarrow \mathit{insert}\ A\ H \vdash B \Longrightarrow H \vdash B$
⟨*proof*⟩

lemma *cut1*: $\{A\} \vdash B \Longrightarrow H \vdash A \Longrightarrow H \vdash B$
⟨*proof*⟩

lemma *rcut1*: $\{A\} \vdash B \Longrightarrow \mathit{insert}\ B\ H \vdash C \Longrightarrow \mathit{insert}\ A\ H \vdash C$
⟨*proof*⟩

lemma *cut2*: $[\![\{A,B\} \vdash C;\ H \vdash A;\ H \vdash B]\!] \Longrightarrow H \vdash C$
⟨*proof*⟩

lemma *rcut2*: $\{A,B\} \vdash C \Longrightarrow \mathit{insert}\ C\ H \vdash D \Longrightarrow H \vdash B \Longrightarrow \mathit{insert}\ A\ H \vdash D$
⟨*proof*⟩

lemma *cut3*: $[\![\{A,B,C\} \vdash D;\ H \vdash A;\ H \vdash B;\ H \vdash C]\!] \Longrightarrow H \vdash D$
⟨*proof*⟩

lemma *cut4*: $[\![\{A,B,C,D\} \vdash E;\ H \vdash A;\ H \vdash B;\ H \vdash C;\ H \vdash D]\!] \Longrightarrow H \vdash E$
⟨*proof*⟩

1.3 Miscellaneous logical rules

lemma *Disj-I1*: $H \vdash A \Longrightarrow H \vdash A\ \mathit{OR}\ B$
⟨*proof*⟩

lemma *Disj-I2*: $H \vdash B \Longrightarrow H \vdash A\ \mathit{OR}\ B$
⟨*proof*⟩

lemma *Peirce*: $H \vdash (\mathit{Neg}\ A)\ \mathit{IMP}\ A \Longrightarrow H \vdash A$
⟨*proof*⟩

lemma *Contra*: $\mathit{insert}\ (\mathit{Neg}\ A)\ H \vdash A \Longrightarrow H \vdash A$
⟨*proof*⟩

lemma *Imp-Neg-I*: $H \vdash A\ \mathit{IMP}\ B \Longrightarrow H \vdash A\ \mathit{IMP}\ (\mathit{Neg}\ B) \Longrightarrow H \vdash \mathit{Neg}\ A$
⟨*proof*⟩

lemma *NegNeg-I*: $H \vdash A \Longrightarrow H \vdash \mathit{Neg}\ (\mathit{Neg}\ A)$
⟨*proof*⟩

lemma *NegNeg-D*: $H \vdash Neg\ (Neg\ A) \Longrightarrow H \vdash A$
⟨*proof*⟩

lemma *Neg-D*: $H \vdash Neg\ A \Longrightarrow H \vdash A \Longrightarrow H \vdash B$
⟨*proof*⟩

lemma *Disj-Neg-1*: $H \vdash A\ OR\ B \Longrightarrow H \vdash Neg\ B \Longrightarrow H \vdash A$
⟨*proof*⟩

lemma *Disj-Neg-2*: $H \vdash A\ OR\ B \Longrightarrow H \vdash Neg\ A \Longrightarrow H \vdash B$
⟨*proof*⟩

lemma *Neg-Disj-I*: $H \vdash Neg\ A \Longrightarrow H \vdash Neg\ B \Longrightarrow H \vdash Neg\ (A\ OR\ B)$
⟨*proof*⟩

lemma *Conj-I* [*intro*!]: $H \vdash A \Longrightarrow H \vdash B \Longrightarrow H \vdash A\ AND\ B$
⟨*proof*⟩

lemma *Conj-E1*: $H \vdash A\ AND\ B \Longrightarrow H \vdash A$
⟨*proof*⟩

lemma *Conj-E2*: $H \vdash A\ AND\ B \Longrightarrow H \vdash B$
⟨*proof*⟩

lemma *Conj-commute*: $H \vdash B\ AND\ A \Longrightarrow H \vdash A\ AND\ B$
⟨*proof*⟩

lemma *Conj-E*: **assumes** $insert\ A\ (insert\ B\ H) \vdash C$ **shows** $insert\ (A\ AND\ B)\ H \vdash C$
⟨*proof*⟩

lemmas *Conj-EH* = *Conj-E Conj-E* [*THEN rotate2*] *Conj-E* [*THEN rotate3*] *Conj-E* [*THEN rotate4*] *Conj-E* [*THEN rotate5*]
Conj-E [*THEN rotate6*] *Conj-E* [*THEN rotate7*] *Conj-E* [*THEN rotate8*] *Conj-E* [*THEN rotate9*] *Conj-E* [*THEN rotate10*]
declare *Conj-EH* [*intro*!]

lemma *Neg-I0*: **assumes** $(\bigwedge B.\ atom\ i \sharp B \Longrightarrow insert\ A\ H \vdash B)$ **shows** $H \vdash Neg\ A$
⟨*proof*⟩

lemma *Neg-mono*: $insert\ A\ H \vdash B \Longrightarrow insert\ (Neg\ B)\ H \vdash Neg\ A$
⟨*proof*⟩

lemma *Conj-mono*: $insert\ A\ H \vdash B \Longrightarrow insert\ C\ H \vdash D \Longrightarrow insert\ (A\ AND\ C)\ H \vdash B\ AND\ D$
⟨*proof*⟩

lemma *Disj-mono*:

assumes *insert A H* $\vdash$ *B insert C H* $\vdash$ *D* **shows** *insert* (*A OR C*) *H* $\vdash$ *B OR D*
⟨*proof*⟩

lemma *Disj-E*:
assumes *A*: *insert A H* $\vdash$ *C* **and** *B*: *insert B H* $\vdash$ *C* **shows** *insert* (*A OR B*) *H* $\vdash$ *C*
⟨*proof*⟩

lemmas *Disj-EH* = *Disj-E Disj-E* [*THEN rotate2*] *Disj-E* [*THEN rotate3*] *Disj-E* [*THEN rotate4*] *Disj-E* [*THEN rotate5*]
Disj-E [*THEN rotate6*] *Disj-E* [*THEN rotate7*] *Disj-E* [*THEN rotate8*] *Disj-E* [*THEN rotate9*] *Disj-E* [*THEN rotate10*]
declare *Disj-EH* [*intro*!]

lemma *Contra′*: *insert A H* $\vdash$ *Neg A* $\Longrightarrow$ *H* $\vdash$ *Neg A*
⟨*proof*⟩

lemma *NegNeg-E* [*intro*!]: *insert A H* $\vdash$ *B* $\Longrightarrow$ *insert* (*Neg* (*Neg A*)) *H* $\vdash$ *B*
⟨*proof*⟩

declare *NegNeg-E* [*THEN rotate2*, *intro*!]
declare *NegNeg-E* [*THEN rotate3*, *intro*!]
declare *NegNeg-E* [*THEN rotate4*, *intro*!]
declare *NegNeg-E* [*THEN rotate5*, *intro*!]
declare *NegNeg-E* [*THEN rotate6*, *intro*!]
declare *NegNeg-E* [*THEN rotate7*, *intro*!]
declare *NegNeg-E* [*THEN rotate8*, *intro*!]

lemma *Imp-E*:
assumes *A*: *H* $\vdash$ *A* **and** *B*: *insert B H* $\vdash$ *C* **shows** *insert* (*A IMP B*) *H* $\vdash$ *C*
⟨*proof*⟩

lemma *Imp-cut*:
assumes *insert C H* $\vdash$ *A IMP B* {*A*} $\vdash$ *C*
shows *H* $\vdash$ *A IMP B*
⟨*proof*⟩

lemma *Iff-I* [*intro*!]: *insert A H* $\vdash$ *B* $\Longrightarrow$ *insert B H* $\vdash$ *A* $\Longrightarrow$ *H* $\vdash$ *A IFF B*
⟨*proof*⟩

lemma *Iff-MP-same*: *H* $\vdash$ *A IFF B* $\Longrightarrow$ *H* $\vdash$ *A* $\Longrightarrow$ *H* $\vdash$ *B*
⟨*proof*⟩

lemma *Iff-MP2-same*: *H* $\vdash$ *A IFF B* $\Longrightarrow$ *H* $\vdash$ *B* $\Longrightarrow$ *H* $\vdash$ *A*
⟨*proof*⟩

lemma *Iff-refl* [*intro*!]: *H* $\vdash$ *A IFF A*
⟨*proof*⟩

lemma *Iff-sym*: $H \vdash A\ IFF\ B \Longrightarrow H \vdash B\ IFF\ A$
⟨*proof*⟩

lemma *Iff-trans*: $H \vdash A\ IFF\ B \Longrightarrow H \vdash B\ IFF\ C \Longrightarrow H \vdash A\ IFF\ C$
⟨*proof*⟩

lemma *Iff-E*:
$insert\ A\ (insert\ B\ H) \vdash C \Longrightarrow insert\ (Neg\ A)\ (insert\ (Neg\ B)\ H) \vdash C \Longrightarrow insert\ (A\ IFF\ B)\ H \vdash C$
⟨*proof*⟩

lemma *Iff-E1*:
assumes *A*: $H \vdash A$ **and** *B*: $insert\ B\ H \vdash C$ **shows** $insert\ (A\ IFF\ B)\ H \vdash C$
⟨*proof*⟩

lemma *Iff-E2*:
assumes *A*: $H \vdash A$ **and** *B*: $insert\ B\ H \vdash C$ **shows** $insert\ (B\ IFF\ A)\ H \vdash C$
⟨*proof*⟩

lemma *Iff-MP-left*: $H \vdash A\ IFF\ B \Longrightarrow insert\ A\ H \vdash C \Longrightarrow insert\ B\ H \vdash C$
⟨*proof*⟩

lemma *Iff-MP-left'*: $H \vdash A\ IFF\ B \Longrightarrow insert\ B\ H \vdash C \Longrightarrow insert\ A\ H \vdash C$
⟨*proof*⟩

lemma *Swap*: $insert\ (Neg\ B)\ H \vdash A \Longrightarrow insert\ (Neg\ A)\ H \vdash B$
⟨*proof*⟩

lemma *Cases*: $insert\ A\ H \vdash B \Longrightarrow insert\ (Neg\ A)\ H \vdash B \Longrightarrow H \vdash B$
⟨*proof*⟩

lemma *Neg-Conj-E*: $H \vdash B \Longrightarrow insert\ (Neg\ A)\ H \vdash C \Longrightarrow insert\ (Neg\ (A\ AND\ B))\ H \vdash C$
⟨*proof*⟩

lemma *Disj-CI*: $insert\ (Neg\ B)\ H \vdash A \Longrightarrow H \vdash A\ OR\ B$
⟨*proof*⟩

lemma *Disj-3I*: $insert\ (Neg\ A)\ (insert\ (Neg\ C)\ H) \vdash B \Longrightarrow H \vdash A\ OR\ B\ OR\ C$
⟨*proof*⟩

lemma *Contrapos1*: $H \vdash A\ IMP\ B \Longrightarrow H \vdash Neg\ B\ IMP\ Neg\ A$
⟨*proof*⟩

lemma *Contrapos2*: $H \vdash (Neg\ B)\ IMP\ (Neg\ A) \Longrightarrow H \vdash A\ IMP\ B$
⟨*proof*⟩

lemma *ContraAssumeN* [*intro*]: $B \in H \Longrightarrow insert\ (Neg\ B)\ H \vdash A$
⟨*proof*⟩

lemma *ContraAssume*: $Neg\ B \in H \Longrightarrow insert\ B\ H \vdash A$
⟨*proof*⟩

lemma *ContraProve*: $H \vdash B \Longrightarrow insert\ (Neg\ B)\ H \vdash A$
⟨*proof*⟩

lemma *Disj-IE1*: $insert\ B\ H \vdash C \Longrightarrow insert\ (A\ OR\ B)\ H \vdash A\ OR\ C$
⟨*proof*⟩

lemmas *Disj-IE1H = Disj-IE1 Disj-IE1 [THEN rotate2] Disj-IE1 [THEN rotate3] Disj-IE1 [THEN rotate4] Disj-IE1 [THEN rotate5]*
Disj-IE1 [THEN rotate6] Disj-IE1 [THEN rotate7] Disj-IE1 [THEN rotate8]
declare *Disj-IE1H [intro!]*

1.3.1 Quantifier reasoning

lemma *Ex-I*: $H \vdash A(i::=x) \Longrightarrow H \vdash Ex\ i\ A$
⟨*proof*⟩

lemma *Ex-E*:
assumes $insert\ A\ H \vdash B\ atom\ i \sharp B\ \forall C \in H.\ atom\ i \sharp C$
shows $insert\ (Ex\ i\ A)\ H \vdash B$
⟨*proof*⟩

lemma *Ex-E-with-renaming*:
assumes $insert\ ((i \leftrightarrow i') \cdot A)\ H \vdash B\ atom\ i' \sharp (A,i,B)\ \forall C \in H.\ atom\ i' \sharp C$
shows $insert\ (Ex\ i\ A)\ H \vdash B$
⟨*proof*⟩

lemmas *Ex-EH = Ex-E Ex-E [THEN rotate2] Ex-E [THEN rotate3] Ex-E [THEN rotate4] Ex-E [THEN rotate5]*
Ex-E [THEN rotate6] Ex-E [THEN rotate7] Ex-E [THEN rotate8] Ex-E [THEN rotate9] Ex-E [THEN rotate10]
declare *Ex-EH [intro!]*

lemma *Ex-mono*: $insert\ A\ H \vdash B \Longrightarrow \forall C \in H.\ atom\ i \sharp C \Longrightarrow insert\ (Ex\ i\ A)\ H \vdash (Ex\ i\ B)$
⟨*proof*⟩

lemma *All-I [intro!]*: $H \vdash A \Longrightarrow \forall C \in H.\ atom\ i \sharp C \Longrightarrow H \vdash All\ i\ A$
⟨*proof*⟩

lemma *All-D*: $H \vdash All\ i\ A \Longrightarrow H \vdash A(i::=x)$
⟨*proof*⟩

lemma *All-E*: $insert\ (A(i::=x))\ H \vdash B \Longrightarrow insert\ (All\ i\ A)\ H \vdash B$
⟨*proof*⟩

lemma *All-E'*: $H \vdash All\ i\ A \Longrightarrow insert\ (A(i::=x))\ H \vdash B \Longrightarrow H \vdash B$
⟨*proof*⟩

lemma *All2-E*: $[\![atom\ i \sharp t;\ H \vdash x\ IN\ t;\ insert\ (A(i::=x))\ H \vdash B]\!] \Longrightarrow insert\ (All2\ i\ t\ A)\ H \vdash B$
⟨*proof*⟩

lemma *All2-E'*: $[\![H \vdash All2\ i\ t\ A;\ H \vdash x\ IN\ t;\ insert\ (A(i::=x))\ H \vdash B;\ atom\ i \sharp t]\!] \Longrightarrow H \vdash B$
⟨*proof*⟩

1.3.2 Congruence rules

lemma *Neg-cong*: $H \vdash A\ IFF\ A' \Longrightarrow H \vdash Neg\ A\ IFF\ Neg\ A'$
⟨*proof*⟩

lemma *Disj-cong*: $H \vdash A\ IFF\ A' \Longrightarrow H \vdash B\ IFF\ B' \Longrightarrow H \vdash A\ OR\ B\ IFF\ A'\ OR\ B'$
⟨*proof*⟩

lemma *Conj-cong*: $H \vdash A\ IFF\ A' \Longrightarrow H \vdash B\ IFF\ B' \Longrightarrow H \vdash A\ AND\ B\ IFF\ A'\ AND\ B'$
⟨*proof*⟩

lemma *Imp-cong*: $H \vdash A\ IFF\ A' \Longrightarrow H \vdash B\ IFF\ B' \Longrightarrow H \vdash (A\ IMP\ B)\ IFF\ (A'\ IMP\ B')$
⟨*proof*⟩

lemma *Iff-cong*: $H \vdash A\ IFF\ A' \Longrightarrow H \vdash B\ IFF\ B' \Longrightarrow H \vdash (A\ IFF\ B)\ IFF\ (A'\ IFF\ B')$
⟨*proof*⟩

lemma *Ex-cong*: $H \vdash A\ IFF\ A' \Longrightarrow \forall C \in H.\ atom\ i \sharp C \Longrightarrow H \vdash (Ex\ i\ A)\ IFF\ (Ex\ i\ A')$
⟨*proof*⟩

lemma *All-cong*: $H \vdash A\ IFF\ A' \Longrightarrow \forall C \in H.\ atom\ i \sharp C \Longrightarrow H \vdash (All\ i\ A)\ IFF\ (All\ i\ A')$
⟨*proof*⟩

lemma *Subst*: $H \vdash A \Longrightarrow \forall B \in H.\ atom\ i \sharp B \Longrightarrow H \vdash A\ (i::=x)$
⟨*proof*⟩

1.4 Equality reasoning

1.4.1 The congruence property for (EQ), and other basic properties of equality

lemma *Eq-cong1*: $\{\} \vdash (t\ EQ\ t'\ AND\ u\ EQ\ u')\ IMP\ (t\ EQ\ u\ IMP\ t'\ EQ\ u')$
⟨*proof*⟩

lemma *Refl* [*iff*]: $H \vdash t\ EQ\ t$
⟨*proof*⟩

Apparently necessary in order to prove the congruence property.

lemma *Sym*: **assumes** $H \vdash t\ EQ\ u$ **shows** $H \vdash u\ EQ\ t$
⟨*proof*⟩

lemma *Sym-L*: $insert\ (t\ EQ\ u)\ H \vdash A \Longrightarrow insert\ (u\ EQ\ t)\ H \vdash A$
⟨*proof*⟩

lemma *Trans*: **assumes** $H \vdash x\ EQ\ y\ H \vdash y\ EQ\ z$ **shows** $H \vdash x\ EQ\ z$
⟨*proof*⟩

lemma *Eq-cong*:
assumes $H \vdash t\ EQ\ t'\ H \vdash u\ EQ\ u'$ **shows** $H \vdash t\ EQ\ u\ IFF\ t'\ EQ\ u'$
⟨*proof*⟩

lemma *Eq-Trans-E*: $H \vdash x\ EQ\ u \Longrightarrow insert\ (t\ EQ\ u)\ H \vdash A \Longrightarrow insert\ (x\ EQ\ t)\ H \vdash A$
⟨*proof*⟩

1.4.2 The congruence property for (IN)

lemma *Mem-cong1*: $\{\} \vdash (t\ EQ\ t'\ AND\ u\ EQ\ u')\ IMP\ (t\ IN\ u\ IMP\ t'\ IN\ u')$
⟨*proof*⟩

lemma *Mem-cong*:
assumes $H \vdash t\ EQ\ t'\ H \vdash u\ EQ\ u'$ **shows** $H \vdash t\ IN\ u\ IFF\ t'\ IN\ u'$
⟨*proof*⟩

1.4.3 The congruence properties for *Eats* and *HPair*

lemma *Eats-cong1*: $\{\} \vdash (t\ EQ\ t'\ AND\ u\ EQ\ u')\ IMP\ (Eats\ t\ u\ EQ\ Eats\ t'\ u')$
⟨*proof*⟩

lemma *Eats-cong*: $[\![H \vdash t\ EQ\ t';\ H \vdash u\ EQ\ u']\!] \Longrightarrow H \vdash Eats\ t\ u\ EQ\ Eats\ t'\ u'$
⟨*proof*⟩

lemma *HPair-cong*: $[\![H \vdash t\ EQ\ t';\ H \vdash u\ EQ\ u']\!] \Longrightarrow H \vdash HPair\ t\ u\ EQ\ HPair\ t'\ u'$
⟨*proof*⟩

lemma *SUCC-cong*: $H \vdash t\ EQ\ t' \Longrightarrow H \vdash SUCC\ t\ EQ\ SUCC\ t'$
⟨*proof*⟩

1.4.4 Substitution for Equalities

lemma *Eq-subst-tm-Iff*: $\{t\ EQ\ u\} \vdash subst\ i\ t\ tm\ EQ\ subst\ i\ u\ tm$
⟨*proof*⟩

lemma *Eq-subst-fm-Iff*: $insert\ (t\ EQ\ u)\ H \vdash A(i::=t)\ IFF\ A(i::=u)$
⟨*proof*⟩

lemma *Var-Eq-subst-Iff*: $insert\ (Var\ i\ EQ\ t)\ H \vdash A(i::=t)\ IFF\ A$
⟨*proof*⟩

lemma *Var-Eq-imp-subst-Iff*: $H \vdash Var\ i\ EQ\ t \Longrightarrow H \vdash A(i::=t)\ IFF\ A$
⟨*proof*⟩

1.4.5 Congruence Rules for Predicates

lemma *P1-cong*:
fixes *tms* :: *tm list*
assumes $\bigwedge i\ t\ x.\ atom\ i \sharp tms \Longrightarrow (P\ t)(i::=x) = P\ (subst\ i\ x\ t)$ **and** $H \vdash x\ EQ\ x'$
shows $H \vdash P\ x\ IFF\ P\ x'$
⟨*proof*⟩

lemma *P2-cong*:
fixes *tms* :: *tm list*
assumes *sub*: $\bigwedge i\ t\ u\ x.\ atom\ i \sharp tms \Longrightarrow (P\ t\ u)(i::=x) = P\ (subst\ i\ x\ t)\ (subst\ i\ x\ u)$
and *eq*: $H \vdash x\ EQ\ x'\ H \vdash y\ EQ\ y'$
shows $H \vdash P\ x\ y\ IFF\ P\ x'\ y'$
⟨*proof*⟩

lemma *P3-cong*:
fixes *tms* :: *tm list*
assumes *sub*: $\bigwedge i\ t\ u\ v\ x.\ atom\ i \sharp tms \Longrightarrow (P\ t\ u\ v)(i::=x) = P\ (subst\ i\ x\ t)\ (subst\ i\ x\ u)\ (subst\ i\ x\ v)$
and *eq*: $H \vdash x\ EQ\ x'\ H \vdash y\ EQ\ y'\ H \vdash z\ EQ\ z'$
shows $H \vdash P\ x\ y\ z\ IFF\ P\ x'\ y'\ z'$
⟨*proof*⟩

lemma *P4-cong*:
fixes *tms* :: *tm list*
assumes *sub*: $\bigwedge i\ t1\ t2\ t3\ t4\ x.\ atom\ i \sharp tms \Longrightarrow (P\ t1\ t2\ t3\ t4)(i::=x) = P\ (subst\ i\ x\ t1)\ (subst\ i\ x\ t2)\ (subst\ i\ x\ t3)\ (subst\ i\ x\ t4)$
and *eq*: $H \vdash x1\ EQ\ x1'\ H \vdash x2\ EQ\ x2'\ H \vdash x3\ EQ\ x3'\ H \vdash x4\ EQ\ x4'$
shows $H \vdash P\ x1\ x2\ x3\ x4\ IFF\ P\ x1'\ x2'\ x3'\ x4'$
⟨*proof*⟩

1.5 Zero and Falsity

lemma *Mem-Zero-iff*:
assumes *atom i* ♯ *t* **shows** $H \vdash (t\ EQ\ Zero)\ IFF\ (All\ i\ (Neg\ ((Var\ i)\ IN\ t)))$
⟨*proof*⟩

lemma *Mem-Zero-E* [*intro*!]: $insert\ (x\ IN\ Zero)\ H \vdash A$
⟨*proof*⟩

declare *Mem-Zero-E* [*THEN rotate2*, *intro*!]
declare *Mem-Zero-E* [*THEN rotate3*, *intro*!]
declare *Mem-Zero-E* [*THEN rotate4*, *intro*!]
declare *Mem-Zero-E* [*THEN rotate5*, *intro*!]
declare *Mem-Zero-E* [*THEN rotate6*, *intro*!]
declare *Mem-Zero-E* [*THEN rotate7*, *intro*!]
declare *Mem-Zero-E* [*THEN rotate8*, *intro*!]

1.5.1 The Formula *Fls*; Consistency of the Calculus

definition *Fls* **where** $Fls \equiv Zero\ IN\ Zero$

lemma *Fls-eqvt* [*eqvt*]: $(p \cdot Fls) = Fls$
⟨*proof*⟩

lemma *Fls-fresh* [*simp*]: *a* ♯ *Fls*
⟨*proof*⟩

lemma *Neg-I* [*intro*!]: $insert\ A\ H \vdash Fls \Longrightarrow H \vdash Neg\ A$
⟨*proof*⟩

lemma *Neg-E* [*intro*!]: $H \vdash A \Longrightarrow insert\ (Neg\ A)\ H \vdash Fls$
⟨*proof*⟩

declare *Neg-E* [*THEN rotate2*, *intro*!]
declare *Neg-E* [*THEN rotate3*, *intro*!]
declare *Neg-E* [*THEN rotate4*, *intro*!]
declare *Neg-E* [*THEN rotate5*, *intro*!]
declare *Neg-E* [*THEN rotate6*, *intro*!]
declare *Neg-E* [*THEN rotate7*, *intro*!]
declare *Neg-E* [*THEN rotate8*, *intro*!]

We need these because Neg (A IMP B) doesn't have to be syntactically a conjunction.

lemma *Neg-Imp-I* [*intro*!]: $H \vdash A \Longrightarrow insert\ B\ H \vdash Fls \Longrightarrow H \vdash Neg\ (A\ IMP\ B)$
⟨*proof*⟩

lemma *Neg-Imp-E* [*intro*!]: $insert\ (Neg\ B)\ (insert\ A\ H) \vdash C \Longrightarrow insert\ (Neg\ (A\ IMP\ B))\ H \vdash C$
⟨*proof*⟩

declare *Neg-Imp-E* [*THEN rotate2*, *intro*!]
declare *Neg-Imp-E* [*THEN rotate3*, *intro*!]
declare *Neg-Imp-E* [*THEN rotate4*, *intro*!]
declare *Neg-Imp-E* [*THEN rotate5*, *intro*!]
declare *Neg-Imp-E* [*THEN rotate6*, *intro*!]
declare *Neg-Imp-E* [*THEN rotate7*, *intro*!]
declare *Neg-Imp-E* [*THEN rotate8*, *intro*!]

lemma *Fls-E* [*intro*!]: *insert Fls H* ⊢ *A*
⟨*proof*⟩

declare *Fls-E* [*THEN rotate2*, *intro*!]
declare *Fls-E* [*THEN rotate3*, *intro*!]
declare *Fls-E* [*THEN rotate4*, *intro*!]
declare *Fls-E* [*THEN rotate5*, *intro*!]
declare *Fls-E* [*THEN rotate6*, *intro*!]
declare *Fls-E* [*THEN rotate7*, *intro*!]
declare *Fls-E* [*THEN rotate8*, *intro*!]

lemma *truth-provable*: *H* ⊢ (*Neg Fls*)
⟨*proof*⟩

lemma *ExFalso*: *H* ⊢ *Fls* ⟹ *H* ⊢ *A*
⟨*proof*⟩

Thanks to Andrei Popescu for pointing out that consistency was provable here.

proposition *consistent*: ¬ {} ⊢ *Fls*
⟨*proof*⟩

1.5.2 More properties of *Zero*

lemma *Eq-Zero-D*:
assumes *H* ⊢ *t EQ Zero H* ⊢ *u IN t* **shows** *H* ⊢ *A*
⟨*proof*⟩

lemma *Eq-Zero-thm*:
assumes *atom i* ♯ *t* **shows** {*All i* (*Neg* ((*Var i*) *IN t*))} ⊢ *t EQ Zero*
⟨*proof*⟩

lemma *Eq-Zero-I*:
assumes *insi*: *insert* ((*Var i*) *IN t*) *H* ⊢ *Fls* **and** *i1*: *atom i* ♯ *t* **and** *i2*: ∀ *B* ∈ *H*. *atom i* ♯ *B*
shows *H* ⊢ *t EQ Zero*
⟨*proof*⟩

1.5.3 Basic properties of *Eats*

lemma *Eq-Eats-iff*:
assumes *atom i* ♯ (*z*,*t*,*u*)

shows $H \vdash (z\ EQ\ Eats\ t\ u)\ IFF\ (All\ i\ (Var\ i\ IN\ z\ IFF\ Var\ i\ IN\ t\ OR\ Var\ i\ EQ\ u))$
⟨*proof*⟩

lemma *Eq-Eats-I*:
$H \vdash All\ i\ (Var\ i\ IN\ z\ IFF\ Var\ i\ IN\ t\ OR\ Var\ i\ EQ\ u) \Longrightarrow atom\ i \sharp (z,t,u) \Longrightarrow H \vdash z\ EQ\ Eats\ t\ u$
⟨*proof*⟩

lemma *Mem-Eats-Iff*:
$H \vdash x\ IN\ (Eats\ t\ u)\ IFF\ x\ IN\ t\ OR\ x\ EQ\ u$
⟨*proof*⟩

lemma *Mem-Eats-I1*: $H \vdash u\ IN\ t \Longrightarrow H \vdash u\ IN\ Eats\ t\ z$
⟨*proof*⟩

lemma *Mem-Eats-I2*: $H \vdash u\ EQ\ z \Longrightarrow H \vdash u\ IN\ Eats\ t\ z$
⟨*proof*⟩

lemma *Mem-Eats-E*:
assumes *A*: $insert\ (u\ IN\ t)\ H \vdash C$ **and** *B*: $insert\ (u\ EQ\ z)\ H \vdash C$
shows $insert\ (u\ IN\ Eats\ t\ z)\ H \vdash C$
⟨*proof*⟩

lemmas *Mem-Eats-EH = Mem-Eats-E Mem-Eats-E* [*THEN rotate2*] *Mem-Eats-E* [*THEN rotate3*] *Mem-Eats-E* [*THEN rotate4*] *Mem-Eats-E* [*THEN rotate5*] *Mem-Eats-E* [*THEN rotate6*] *Mem-Eats-E* [*THEN rotate7*] *Mem-Eats-E* [*THEN rotate8*]
declare *Mem-Eats-EH* [*intro*!]

lemma *Mem-SUCC-I1*: $H \vdash u\ IN\ t \Longrightarrow H \vdash u\ IN\ SUCC\ t$
⟨*proof*⟩

lemma *Mem-SUCC-I2*: $H \vdash u\ EQ\ t \Longrightarrow H \vdash u\ IN\ SUCC\ t$
⟨*proof*⟩

lemma *Mem-SUCC-Refl* [*simp*]: $H \vdash k\ IN\ SUCC\ k$
⟨*proof*⟩

lemma *Mem-SUCC-E*:
assumes $insert\ (u\ IN\ t)\ H \vdash C\ insert\ (u\ EQ\ t)\ H \vdash C$ **shows** $insert\ (u\ IN\ SUCC\ t)\ H \vdash C$
⟨*proof*⟩

lemmas *Mem-SUCC-EH = Mem-SUCC-E Mem-SUCC-E* [*THEN rotate2*] *Mem-SUCC-E* [*THEN rotate3*] *Mem-SUCC-E* [*THEN rotate4*] *Mem-SUCC-E* [*THEN rotate5*] *Mem-SUCC-E* [*THEN rotate6*] *Mem-SUCC-E* [*THEN rotate7*] *Mem-SUCC-E* [*THEN rotate8*]

lemma *Eats-EQ-Zero-E*: *insert* (*Eats t u EQ Zero*) $H \vdash A$
⟨*proof*⟩

lemmas *Eats-EQ-Zero-EH* = *Eats-EQ-Zero-E Eats-EQ-Zero-E* [*THEN rotate2*] *Eats-EQ-Zero-E* [*THEN rotate3*] *Eats-EQ-Zero-E* [*THEN rotate4*] *Eats-EQ-Zero-E* [*THEN rotate5*]
Eats-EQ-Zero-E [*THEN rotate6*] *Eats-EQ-Zero-E* [*THEN rotate7*] *Eats-EQ-Zero-E* [*THEN rotate8*]
declare *Eats-EQ-Zero-EH* [*intro*!]

lemma *Eats-EQ-Zero-E2*: *insert* (*Zero EQ Eats t u*) $H \vdash A$
⟨*proof*⟩

lemmas *Eats-EQ-Zero-E2H* = *Eats-EQ-Zero-E2 Eats-EQ-Zero-E2* [*THEN rotate2*] *Eats-EQ-Zero-E2* [*THEN rotate3*] *Eats-EQ-Zero-E2* [*THEN rotate4*] *Eats-EQ-Zero-E2* [*THEN rotate5*]
Eats-EQ-Zero-E2 [*THEN rotate6*] *Eats-EQ-Zero-E2* [*THEN rotate7*] *Eats-EQ-Zero-E2* [*THEN rotate8*]
declare *Eats-EQ-Zero-E2H* [*intro*!]

1.6 Bounded Quantification involving *Eats*

lemma *All2-cong*: $H \vdash t\ EQ\ t' \Longrightarrow H \vdash A\ IFF\ A' \Longrightarrow \forall C \in H.\ atom\ i \sharp C \Longrightarrow H \vdash (All2\ i\ t\ A)\ IFF\ (All2\ i\ t'\ A')$
⟨*proof*⟩

lemma *All2-Zero-E* [*intro*!]: $H \vdash B \Longrightarrow insert\ (All2\ i\ Zero\ A)\ H \vdash B$
⟨*proof*⟩

lemma *All2-Eats-I-D*:
$atom\ i \sharp (t,u) \Longrightarrow \{\ All2\ i\ t\ A,\ A(i::=u)\} \vdash (All2\ i\ (Eats\ t\ u)\ A)$
⟨*proof*⟩

lemma *All2-Eats-I*:
$[\![atom\ i \sharp (t,u);\ H \vdash All2\ i\ t\ A;\ H \vdash A(i::=u)]\!] \Longrightarrow H \vdash (All2\ i\ (Eats\ t\ u)\ A)$
⟨*proof*⟩

lemma *All2-Eats-E1*:
$[\![atom\ i \sharp (t,u);\ \forall C \in H.\ atom\ i \sharp C]\!] \Longrightarrow insert\ (All2\ i\ (Eats\ t\ u)\ A)\ H \vdash All2\ i\ t\ A$
⟨*proof*⟩

lemma *All2-Eats-E2*:
$[\![atom\ i \sharp (t,u);\ \forall C \in H.\ atom\ i \sharp C]\!] \Longrightarrow insert\ (All2\ i\ (Eats\ t\ u)\ A)\ H \vdash A(i::=u)$
⟨*proof*⟩

lemma *All2-Eats-E*:
assumes *i*: $atom\ i \sharp (t,u)$

and B: *insert* (*All2 i t A*) (*insert* (*A*(*i*::=*u*)) *H*) ⊢ *B*
shows *insert* (*All2 i* (*Eats t u*) *A*) *H* ⊢ *B*
⟨*proof*⟩

lemma *All2-SUCC-I*:
atom i ♯ *t* ⟹ *H* ⊢ *All2 i t A* ⟹ *H* ⊢ *A*(*i*::=*t*) ⟹ *H* ⊢ (*All2 i* (*SUCC t*) *A*)
⟨*proof*⟩

lemma *All2-SUCC-E*:
assumes *atom i* ♯ *t*
and *insert* (*All2 i t A*) (*insert* (*A*(*i*::=*t*)) *H*) ⊢ *B*
shows *insert* (*All2 i* (*SUCC t*) *A*) *H* ⊢ *B*
⟨*proof*⟩

lemma *All2-SUCC-E′*:
assumes *H* ⊢ *u EQ SUCC t*
and *atom i* ♯ *t* ∀ *C* ∈ *H*. *atom i* ♯ *C*
and *insert* (*All2 i t A*) (*insert* (*A*(*i*::=*t*)) *H*) ⊢ *B*
shows *insert* (*All2 i u A*) *H* ⊢ *B*
⟨*proof*⟩

1.7 Induction

lemma *Ind*:
assumes *j*: *atom* (*j*::*name*) ♯ (*i*,*A*)
and *prems*: *H* ⊢ *A*(*i*::=*Zero*) *H* ⊢ *All i* (*All j* (*A IMP* (*A*(*i*::= *Var j*) *IMP A*(*i*::= *Eats*(*Var i*)(*Var j*)))))
shows *H* ⊢ *A*
⟨*proof*⟩

end

Kapitel 2

De Bruijn Syntax, Quotations, Codes, V-Codes

theory *Coding*
imports *SyntaxN*
begin

declare *fresh-Nil* [*iff*]

2.1 de Bruijn Indices (locally-nameless version)

nominal-datatype *dbtm* = *DBZero* | *DBVar name* | *DBInd nat* | *DBEats dbtm dbtm*

nominal-datatype *dbfm* =
  DBMem dbtm dbtm
| *DBEq dbtm dbtm*
| *DBDisj dbfm dbfm*
| *DBNeg dbfm*
| *DBEx dbfm*

declare *dbtm.supp* [*simp*]
declare *dbfm.supp* [*simp*]

fun *lookup* :: *name list* ⇒ *nat* ⇒ *name* ⇒ *dbtm*
  where
  lookup [] *n x* = *DBVar x*
| *lookup* (*y* # *ys*) *n x* = (*if x* = *y then DBInd n else* (*lookup ys* (*Suc n*) *x*))

lemma *fresh-imp-notin-env*: *atom name* ♯ *e* ⟹ *name* ∉ *set e*
  ⟨*proof*⟩

lemma *lookup-notin*: *x* ∉ *set e* ⟹ *lookup e n x* = *DBVar x*
  ⟨*proof*⟩

lemma *lookup-in*:
$x \in set\ e \Longrightarrow \exists k.\ lookup\ e\ n\ x = DBInd\ k \wedge n \leq k \wedge k < n + length\ e$
⟨*proof*⟩

lemma *lookup-fresh*: $x \sharp lookup\ e\ n\ y \longleftrightarrow y \in set\ e \vee x \neq atom\ y$
⟨*proof*⟩

lemma *lookup-eqvt*[*eqvt*]: $(p \cdot lookup\ xs\ n\ x) = lookup\ (p \cdot xs)\ (p \cdot n)\ (p \cdot x)$
⟨*proof*⟩

lemma *lookup-inject* [*iff*]: $(lookup\ e\ n\ x = lookup\ e\ n\ y) \longleftrightarrow x = y$
⟨*proof*⟩

nominal-function *trans-tm* :: *name list* ⇒ *tm* ⇒ *dbtm*
where
trans-tm e Zero = *DBZero*
| *trans-tm e* (*Var k*) = *lookup e 0 k*
| *trans-tm e* (*Eats t u*) = *DBEats* (*trans-tm e t*) (*trans-tm e u*)
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *fresh-trans-tm-iff* [*simp*]: $i \sharp trans\text{-}tm\ e\ t \longleftrightarrow i \sharp t \vee i \in atom\ `\ set\ e$
⟨*proof*⟩

lemma *trans-tm-forget*: $atom\ i \sharp t \Longrightarrow trans\text{-}tm\ [i]\ t = trans\text{-}tm\ []\ t$
⟨*proof*⟩

nominal-function (*invariant* $\lambda(xs, \text{-})\ y.\ atom\ `\ set\ xs \sharp* y$)
trans-fm :: *name list* ⇒ *fm* ⇒ *dbfm*
where
trans-fm e (*Mem t u*) = *DBMem* (*trans-tm e t*) (*trans-tm e u*)
| *trans-fm e* (*Eq t u*) = *DBEq* (*trans-tm e t*) (*trans-tm e u*)
| *trans-fm e* (*Disj A B*) = *DBDisj* (*trans-fm e A*) (*trans-fm e B*)
| *trans-fm e* (*Neg A*) = *DBNeg* (*trans-fm e A*)
| $atom\ k \sharp e \Longrightarrow$ *trans-fm e* (*Ex k A*) = *DBEx* (*trans-fm* (*k#e*) *A*)
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *fresh-trans-fm* [*simp*]: $i \sharp trans\text{-}fm\ e\ A \longleftrightarrow i \sharp A \vee i \in atom\ `\ set\ e$
⟨*proof*⟩

abbreviation *DBConj* :: *dbfm* ⇒ *dbfm* ⇒ *dbfm*
where *DBConj t u* ≡ *DBNeg* (*DBDisj* (*DBNeg t*) (*DBNeg u*))

lemma *trans-fm-Conj* [*simp*]: *trans-fm e* (*Conj A B*) = *DBConj* (*trans-fm e A*) (*trans-fm e B*)
⟨*proof*⟩

lemma *trans-tm-inject* [*iff*]: (*trans-tm e t* = *trans-tm e u*) ⟷ *t* = *u*
⟨*proof*⟩

lemma *trans-fm-inject* [*iff*]: (*trans-fm e A* = *trans-fm e B*) ⟷ *A* = *B*
⟨*proof*⟩

lemma *trans-fm-perm*:
assumes *c*: *atom c* ♯ (*i*,*j*,*A*,*B*)
and *t*: *trans-fm* [*i*] *A* = *trans-fm* [*j*] *B*
shows (*i* ↔ *c*) ∙ *A* = (*j* ↔ *c*) ∙ *B*
⟨*proof*⟩

2.2 Abstraction and Substitution on de Bruijn Formulas

nominal-function *abst-dbtm* :: *name* ⇒ *nat* ⇒ *dbtm* ⇒ *dbtm*
where
abst-dbtm name i DBZero = *DBZero*
| *abst-dbtm name i* (*DBVar name*′) = (*if name* = *name*′ *then DBInd i else DBVar name*′)
| *abst-dbtm name i* (*DBInd j*) = *DBInd j*
| *abst-dbtm name i* (*DBEats t1 t2*) = *DBEats* (*abst-dbtm name i t1*) (*abst-dbtm name i t2*)
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

nominal-function *subst-dbtm* :: *dbtm* ⇒ *name* ⇒ *dbtm* ⇒ *dbtm*
where
subst-dbtm u x DBZero = *DBZero*
| *subst-dbtm u x* (*DBVar name*) = (*if x* = *name then u else DBVar name*)
| *subst-dbtm u x* (*DBInd j*) = *DBInd j*
| *subst-dbtm u x* (*DBEats t1 t2*) = *DBEats* (*subst-dbtm u x t1*) (*subst-dbtm u x t2*)
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *fresh-iff-non-subst-dbtm*: *subst-dbtm DBZero i t* = *t* ⟷ *atom i* ♯ *t*
⟨*proof*⟩

lemma *lookup-append*: *lookup* (*e* @ [*i*]) *n j* = *abst-dbtm i* (*length e* + *n*) (*lookup e*

n j)
⟨*proof*⟩

lemma *trans-tm-abs*: *trans-tm* (*e*@[*name*]) *t* = *abst-dbtm name* (*length e*) (*trans-tm e t*)
⟨*proof*⟩

2.2.1 Well-Formed Formulas

nominal-function *abst-dbfm* :: *name* ⇒ *nat* ⇒ *dbfm* ⇒ *dbfm*
where
abst-dbfm name i (*DBMem t1 t2*) = *DBMem* (*abst-dbtm name i t1*) (*abst-dbtm name i t2*)
| *abst-dbfm name i* (*DBEq t1 t2*) = *DBEq* (*abst-dbtm name i t1*) (*abst-dbtm name i t2*)
| *abst-dbfm name i* (*DBDisj A1 A2*) = *DBDisj* (*abst-dbfm name i A1*) (*abst-dbfm name i A2*)
| *abst-dbfm name i* (*DBNeg A*) = *DBNeg* (*abst-dbfm name i A*)
| *abst-dbfm name i* (*DBEx A*) = *DBEx* (*abst-dbfm name* (*i*+*1*) *A*)
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

nominal-function *subst-dbfm* :: *dbtm* ⇒ *name* ⇒ *dbfm* ⇒ *dbfm*
where
subst-dbfm u x (*DBMem t1 t2*) = *DBMem* (*subst-dbtm u x t1*) (*subst-dbtm u x t2*)
| *subst-dbfm u x* (*DBEq t1 t2*) = *DBEq* (*subst-dbtm u x t1*) (*subst-dbtm u x t2*)
| *subst-dbfm u x* (*DBDisj A1 A2*) = *DBDisj* (*subst-dbfm u x A1*) (*subst-dbfm u x A2*)
| *subst-dbfm u x* (*DBNeg A*) = *DBNeg* (*subst-dbfm u x A*)
| *subst-dbfm u x* (*DBEx A*) = *DBEx* (*subst-dbfm u x A*)
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *fresh-iff-non-subst-dbfm*: *subst-dbfm DBZero i t* = *t* ⟷ *atom i* ♯ *t*
⟨*proof*⟩

2.3 Well formed terms and formulas (de Bruijn representation)

2.3.1 Well-Formed Terms

inductive *wf-dbtm* :: *dbtm* ⇒ *bool*
where
Zero: *wf-dbtm DBZero*

| *Var*: *wf-dbtm* (*DBVar name*)
| *Eats*: *wf-dbtm t1* $\Longrightarrow$ *wf-dbtm t2* $\Longrightarrow$ *wf-dbtm* (*DBEats t1 t2*)

equivariance *wf-dbtm*

inductive-cases *Zero-wf-dbtm* [*elim*!]: *wf-dbtm DBZero*
inductive-cases *Var-wf-dbtm* [*elim*!]: *wf-dbtm* (*DBVar name*)
inductive-cases *Ind-wf-dbtm* [*elim*!]: *wf-dbtm* (*DBInd i*)
inductive-cases *Eats-wf-dbtm* [*elim*!]: *wf-dbtm* (*DBEats t1 t2*)

declare *wf-dbtm.intros* [*intro*]

lemma *wf-dbtm-imp-is-tm*:
assumes *wf-dbtm x*
shows $\exists$ *t*::*tm*. *x* = *trans-tm* [] *t*
⟨*proof*⟩

lemma *wf-dbtm-trans-tm*: *wf-dbtm* (*trans-tm* [] *t*)
⟨*proof*⟩

theorem *wf-dbtm-iff-is-tm*: *wf-dbtm x* $\longleftrightarrow$ ($\exists$ *t*::*tm*. *x* = *trans-tm* [] *t*)
⟨*proof*⟩

2.3.2 Well-Formed Formulas

inductive *wf-dbfm* :: *dbfm* $\Rightarrow$ *bool*
where
Mem: *wf-dbtm t1* $\Longrightarrow$ *wf-dbtm t2* $\Longrightarrow$ *wf-dbfm* (*DBMem t1 t2*)
| *Eq*: *wf-dbtm t1* $\Longrightarrow$ *wf-dbtm t2* $\Longrightarrow$ *wf-dbfm* (*DBEq t1 t2*)
| *Disj*: *wf-dbfm A1* $\Longrightarrow$ *wf-dbfm A2* $\Longrightarrow$ *wf-dbfm* (*DBDisj A1 A2*)
| *Neg*: *wf-dbfm A* $\Longrightarrow$ *wf-dbfm* (*DBNeg A*)
| *Ex*: *wf-dbfm A* $\Longrightarrow$ *wf-dbfm* (*DBEx* (*abst-dbfm name 0 A*))

equivariance *wf-dbfm*

lemma *atom-fresh-abst-dbtm* [*simp*]: *atom i* $\sharp$ *abst-dbtm i n t*
⟨*proof*⟩

lemma *atom-fresh-abst-dbfm* [*simp*]: *atom i* $\sharp$ *abst-dbfm i n A*
⟨*proof*⟩

Setting up strong induction: ävoiding"for name. Necessary to allow some proofs to go through

nominal-inductive *wf-dbfm*
avoids *Ex*: *name*
⟨*proof*⟩

inductive-cases *Mem-wf-dbfm* [*elim*!]: *wf-dbfm* (*DBMem t1 t2*)
inductive-cases *Eq-wf-dbfm* [*elim*!]: *wf-dbfm* (*DBEq t1 t2*)

inductive-cases *Disj-wf-dbfm* [*elim*!]: *wf-dbfm* (*DBDisj A1 A2*)
inductive-cases *Neg-wf-dbfm* [*elim*!]: *wf-dbfm* (*DBNeg A*)
inductive-cases *Ex-wf-dbfm* [*elim*!]: *wf-dbfm* (*DBEx z*)

declare *wf-dbfm.intros* [*intro*]

lemma *trans-fm-abs*: *trans-fm* (*e*@[*name*]) *A* = *abst-dbfm name* (*length e*) (*trans-fm e A*)
⟨*proof*⟩

lemma *abst-trans-fm*: *abst-dbfm name 0* (*trans-fm* [] *A*) = *trans-fm* [*name*] *A*
⟨*proof*⟩

lemma *abst-trans-fm2*: *i* ≠ *j* ⟹ *abst-dbfm i* (*Suc 0*) (*trans-fm* [*j*] *A*) = *trans-fm* [*j*,*i*] *A*
⟨*proof*⟩

lemma *wf-dbfm-imp-is-fm*:
assumes *wf-dbfm x* **shows** ∃ *A*::*fm*. *x* = *trans-fm* [] *A*
⟨*proof*⟩

lemma *wf-dbfm-trans-fm*: *wf-dbfm* (*trans-fm* [] *A*)
⟨*proof*⟩

lemma *wf-dbfm-iff-is-fm*: *wf-dbfm x* ⟷ (∃ *A*::*fm*. *x* = *trans-fm* [] *A*)
⟨*proof*⟩

lemma *dbtm-abst-ignore* [*simp*]:
abst-dbtm name i (*abst-dbtm name j t*) = *abst-dbtm name j t*
⟨*proof*⟩

lemma *abst-dbtm-fresh-ignore* [*simp*]: *atom name* ♯ *u* ⟹ *abst-dbtm name j u* = *u*
⟨*proof*⟩

lemma *dbtm-subst-ignore* [*simp*]:
subst-dbtm u name (*abst-dbtm name j t*) = *abst-dbtm name j t*
⟨*proof*⟩

lemma *dbtm-abst-swap-subst*:
name ≠ *name*′ ⟹ *atom name*′ ♯ *u* ⟹
subst-dbtm u name (*abst-dbtm name*′ *j t*) = *abst-dbtm name*′ *j* (*subst-dbtm u name t*)
⟨*proof*⟩

lemma *dbfm-abst-swap-subst*:
name ≠ *name*′ ⟹ *atom name*′ ♯ *u* ⟹
subst-dbfm u name (*abst-dbfm name*′ *j A*) = *abst-dbfm name*′ *j* (*subst-dbfm u name A*)
⟨*proof*⟩

lemma *subst-trans-commute* [*simp*]:
atom i ♯ *e* ⟹ *subst-dbtm* (*trans-tm e u*) *i* (*trans-tm e t*) = *trans-tm e* (*subst i u t*)
⟨*proof*⟩

lemma *subst-fm-trans-commute* [*simp*]:
subst-dbfm (*trans-tm* [] *u*) *name* (*trans-fm* [] *A*) = *trans-fm* [] (*A* (*name*::= *u*))
⟨*proof*⟩

lemma *subst-fm-trans-commute-eq*:
du = *trans-tm* [] *u* ⟹ *subst-dbfm du i* (*trans-fm* [] *A*) = *trans-fm* [] (*A*(*i*::=*u*))
⟨*proof*⟩

2.4 Quotations

fun *htuple* :: *nat* ⇒ *hf* **where**
htuple 0 = ⟨*0*,*0*⟩
| *htuple* (*Suc k*) = ⟨*0*, *htuple k*⟩

fun *HTuple* :: *nat* ⇒ *tm* **where**
HTuple 0 = *HPair Zero Zero*
| *HTuple* (*Suc k*) = *HPair Zero* (*HTuple k*)

lemma *eval-tm-HTuple* [*simp*]: ⟦*HTuple n*⟧*e* = *htuple n*
⟨*proof*⟩

lemma *fresh-HTuple* [*simp*]: *x* ♯ *HTuple n*
⟨*proof*⟩

lemma *HTuple-eqvt*[*eqvt*]: (*p* ∙ *HTuple n*) = *HTuple* (*p* ∙ *n*)
⟨*proof*⟩

lemma *htuple-nonzero* [*simp*]: *htuple k* ≠ *0*
⟨*proof*⟩

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

2.4.1 Quotations of de Bruijn terms

definition *nat-of-name* :: *name* ⇒ *nat*
where *nat-of-name x* = *nat-of* (*atom x*)

lemma *nat-of-name-inject* [*simp*]: *nat-of-name n1* = *nat-of-name n2* ⟷ *n1* = *n2*
⟨*proof*⟩

definition *name-of-nat* :: *nat* ⇒ *name*

where *name-of-nat n* ≡ *Abs-name* (*Atom* (*Sort ''SyntaxN.name''* []) *n*)

lemma *nat-of-name-Abs-eq* [*simp*]: *nat-of-name* (*Abs-name* (*Atom* (*Sort ''SyntaxN.name''* []) *n*)) = *n*
⟨*proof*⟩

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

lemma *name-of-nat-nat-of-name* [*simp*]: *name-of-nat* (*nat-of-name i*) = *i*
⟨*proof*⟩

lemma *HPair-neq-ORD-OF* [*simp*]: *HPair x y* ≠ *ORD-OF i*
⟨*proof*⟩

Infinite support, so we cannot use nominal primrec.

function *quot-dbtm* :: *dbtm* ⇒ *tm*
where
quot-dbtm DBZero = *Zero*
| *quot-dbtm* (*DBVar name*) = *ORD-OF* (*Suc* (*nat-of-name name*))
| *quot-dbtm* (*DBInd k*) = *HPair* (*HTuple 6*) (*ORD-OF k*)
| *quot-dbtm* (*DBEats t u*) = *HPair* (*HTuple 1*) (*HPair* (*quot-dbtm t*) (*quot-dbtm u*))
⟨*proof*⟩

termination
⟨*proof*⟩

lemma *quot-dbtm-inject-lemma* [*simp*]: ⟦*quot-dbtm t*⟧*e* = ⟦*quot-dbtm u*⟧*e* ⟷ *t*=*u*
⟨*proof*⟩

lemma *quot-dbtm-inject* [*iff*]: *quot-dbtm t* = *quot-dbtm u* ⟷ *t*=*u*
⟨*proof*⟩

2.4.2 Quotations of de Bruijn formulas

Infinite support, so we cannot use nominal primrec.

function *quot-dbfm* :: *dbfm* ⇒ *tm*
where
quot-dbfm (*DBMem t u*) = *HPair* (*HTuple 0*) (*HPair* (*quot-dbtm t*) (*quot-dbtm u*))
| *quot-dbfm* (*DBEq t u*) = *HPair* (*HTuple 2*) (*HPair* (*quot-dbtm t*) (*quot-dbtm u*))
| *quot-dbfm* (*DBDisj A B*) = *HPair* (*HTuple 3*) (*HPair* (*quot-dbfm A*) (*quot-dbfm B*))
| *quot-dbfm* (*DBNeg A*) = *HPair* (*HTuple 4*) (*quot-dbfm A*)
| *quot-dbfm* (*DBEx A*) = *HPair* (*HTuple 5*) (*quot-dbfm A*)
⟨*proof*⟩

termination

⟨*proof*⟩

lemma *htuple-minus-1*: $n > 0 \Longrightarrow htuple\ n = \langle 0, htuple\ (n - 1)\rangle$
⟨*proof*⟩

lemma *HTuple-minus-1*: $n > 0 \Longrightarrow HTuple\ n = HPair\ Zero\ (HTuple\ (n - 1))$
⟨*proof*⟩

lemmas *HTS = HTuple-minus-1 HTuple.simps* — for freeness reasoning on codes

lemma *quot-dbfm-inject-lemma* [*simp*]: $[\![quot\text{-}dbfm\ A]\!]e = [\![quot\text{-}dbfm\ B]\!]e \longleftrightarrow A{=}B$
⟨*proof*⟩

class *quot* =
fixes *quot* :: $'a \Rightarrow tm$ (‹«-»›)

instantiation *tm* :: *quot*
begin
definition *quot-tm* :: $tm \Rightarrow tm$
where *quot-tm t = quot-dbtm* (*trans-tm* [] *t*)

instance ⟨*proof*⟩
end

lemma *quot-dbtm-fresh* [*simp*]: $s \sharp (quot\text{-}dbtm\ t)$
⟨*proof*⟩

lemma *quot-tm-fresh* [*simp*]: **fixes** *t*::*tm* **shows** $s \sharp «t»$
⟨*proof*⟩

lemma *quot-Zero* [*simp*]: «*Zero*» = *Zero*
⟨*proof*⟩

lemma *quot-Var*: «*Var x*» = *SUCC* (*ORD-OF* (*nat-of-name x*))
⟨*proof*⟩

lemma *quot-Eats*: «*Eats x y*» = *HPair* (*HTuple 1*) (*HPair* «*x*» «*y*»)
⟨*proof*⟩

irrelevance of the environment for quotations, because they are ground terms

lemma *eval-quot-dbtm-ignore*:
$[\![quot\text{-}dbtm\ t]\!]e = [\![quot\text{-}dbtm\ t]\!]e'$
⟨*proof*⟩

lemma *eval-quot-dbfm-ignore*:
$[\![quot\text{-}dbfm\ A]\!]e = [\![quot\text{-}dbfm\ A]\!]e'$

⟨*proof*⟩

instantiation *fm* :: *quot*
begin
definition *quot-fm* :: *fm* ⇒ *tm*
where *quot-fm A* = *quot-dbfm* (*trans-fm* [] *A*)

instance ⟨*proof*⟩
end

lemma *quot-dbfm-fresh* [*simp*]: *s* ♯ (*quot-dbfm A*)
⟨*proof*⟩

lemma *quot-fm-fresh* [*simp*]: **fixes** *A*::*fm* **shows** *s* ♯ «*A*»
⟨*proof*⟩

lemma *quot-fm-permute* [*simp*]: **fixes** *A*:: *fm* **shows** *p* ∙ «*A*» = «*A*»
⟨*proof*⟩

lemma *quot-Mem*: «*x IN y*» = *HPair* (*HTuple 0*) (*HPair* («*x*») («*y*»))
⟨*proof*⟩

lemma *quot-Eq*: «*x EQ y*» = *HPair* (*HTuple 2*) (*HPair* («*x*») («*y*»))
⟨*proof*⟩

lemma *quot-Disj*: «*A OR B*» = *HPair* (*HTuple 3*) (*HPair* («*A*») («*B*»))
⟨*proof*⟩

lemma *quot-Neg*: «*Neg A*» = *HPair* (*HTuple 4*) («*A*»)
⟨*proof*⟩

lemma *quot-Ex*: «*Ex i A*» = *HPair* (*HTuple 5*) (*quot-dbfm* (*trans-fm* [*i*] *A*))
⟨*proof*⟩

lemma *eval-quot-fm-ignore*: **fixes** *A*:: *fm* **shows** ⟦«*A*»⟧*e* = ⟦«*A*»⟧*e*′
⟨*proof*⟩

lemmas *quot-simps* = *quot-Var quot-Eats quot-Eq quot-Mem quot-Disj quot-Neg quot-Ex*

2.5 Definitions Involving Coding

definition *q-Var* :: *name* ⇒ *hf*
where *q-Var i* ≡ *succ* (*ord-of* (*nat-of-name i*))

definition *q-Ind* :: *hf* ⇒ *hf*
where *q-Ind k* ≡ ⟨*htuple 6*, *k*⟩

abbreviation *Q-Eats* :: *tm* ⇒ *tm* ⇒ *tm*
  where *Q-Eats t u* ≡ *HPair* (*HTuple* (*Suc 0*)) (*HPair t u*)

definition *q-Eats* :: *hf* ⇒ *hf* ⇒ *hf*
  where *q-Eats x y* ≡ ⟨*htuple 1*, *x*, *y*⟩

abbreviation *Q-Succ* :: *tm* ⇒ *tm*
  where *Q-Succ t* ≡ *Q-Eats t t*

definition *q-Succ* :: *hf* ⇒ *hf*
  where *q-Succ x* ≡ *q-Eats x x*

lemma *quot-Succ*: «*SUCC x*» = *Q-Succ* «*x*»
  ⟨*proof*⟩

abbreviation *Q-HPair* :: *tm* ⇒ *tm* ⇒ *tm*
  where *Q-HPair t u* ≡
        Q-Eats (*Q-Eats Zero* (*Q-Eats* (*Q-Eats Zero u*) *t*))
            (*Q-Eats* (*Q-Eats Zero t*) *t*)

definition *q-HPair* :: *hf* ⇒ *hf* ⇒ *hf*
  where *q-HPair x y* ≡
        q-Eats (*q-Eats 0* (*q-Eats* (*q-Eats 0 y*) *x*))
            (*q-Eats* (*q-Eats 0 x*) *x*)

abbreviation *Q-Mem* :: *tm* ⇒ *tm* ⇒ *tm*
  where *Q-Mem t u* ≡ *HPair* (*HTuple 0*) (*HPair t u*)

definition *q-Mem* :: *hf* ⇒ *hf* ⇒ *hf*
  where *q-Mem x y* ≡ ⟨*htuple 0*, *x*, *y*⟩

abbreviation *Q-Eq* :: *tm* ⇒ *tm* ⇒ *tm*
  where *Q-Eq t u* ≡ *HPair* (*HTuple 2*) (*HPair t u*)

definition *q-Eq* :: *hf* ⇒ *hf* ⇒ *hf*
  where *q-Eq x y* ≡ ⟨*htuple 2*, *x*, *y*⟩

abbreviation *Q-Disj* :: *tm* ⇒ *tm* ⇒ *tm*
  where *Q-Disj t u* ≡ *HPair* (*HTuple 3*) (*HPair t u*)

definition *q-Disj* :: *hf* ⇒ *hf* ⇒ *hf*
  where *q-Disj x y* ≡ ⟨*htuple 3*, *x*, *y*⟩

abbreviation *Q-Neg* :: *tm* ⇒ *tm*
  where *Q-Neg t* ≡ *HPair* (*HTuple 4*) *t*

definition *q-Neg* :: *hf* ⇒ *hf*
  where *q-Neg x* ≡ ⟨*htuple 4*, *x*⟩

abbreviation *Q-Conj* :: *tm* ⇒ *tm* ⇒ *tm*
  where *Q-Conj t u* ≡ *Q-Neg* (*Q-Disj* (*Q-Neg t*) (*Q-Neg u*))

definition *q-Conj* :: *hf* ⇒ *hf* ⇒ *hf*
  where *q-Conj t u* ≡ *q-Neg* (*q-Disj* (*q-Neg t*) (*q-Neg u*))

abbreviation *Q-Imp* :: *tm* ⇒ *tm* ⇒ *tm*
  where *Q-Imp t u* ≡ *Q-Disj* (*Q-Neg t*) *u*

definition *q-Imp* :: *hf* ⇒ *hf* ⇒ *hf*
  where *q-Imp t u* ≡ *q-Disj* (*q-Neg t*) *u*

abbreviation *Q-Ex* :: *tm* ⇒ *tm*
  where *Q-Ex t* ≡ *HPair* (*HTuple 5*) *t*

definition *q-Ex* :: *hf* ⇒ *hf*
  where *q-Ex x* ≡ ⟨*htuple 5*, *x*⟩

abbreviation *Q-All* :: *tm* ⇒ *tm*
  where *Q-All t* ≡ *Q-Neg* (*Q-Ex* (*Q-Neg t*))

definition *q-All* :: *hf* ⇒ *hf*
  where *q-All x* ≡ *q-Neg* (*q-Ex* (*q-Neg x*))

lemmas *q-defs* = *q-Var-def q-Ind-def q-Eats-def q-HPair-def q-Eq-def q-Mem-def*
q-Disj-def q-Neg-def q-Conj-def q-Imp-def q-Ex-def q-All-def

lemma *q-Eats-iff* [*iff*]: *q-Eats x y* = *q-Eats x′ y′* ⟷ *x*=*x′* ∧ *y*=*y′*
⟨*proof*⟩

lemma *quot-subst-eq*: «*A*(*i*::=*t*)» = *quot-dbfm* (*subst-dbfm* (*trans-tm* [] *t*) *i* (*trans-fm* [] *A*))
⟨*proof*⟩

lemma *Q-Succ-cong*: *H* ⊢ *x EQ x′* ⟹ *H* ⊢ *Q-Succ x EQ Q-Succ x′*
⟨*proof*⟩

2.6 Quotations are Injective

2.6.1 Terms

lemma *eval-tm-inject* [*simp*]: **fixes** *t*::*tm* **shows** ⟦«*t*»⟧ *e* = ⟦«*u*»⟧ *e* ⟷ *t*=*u*
⟨*proof*⟩

2.6.2 Formulas

lemma *eval-fm-inject* [*simp*]: **fixes** *A*::*fm* **shows** ⟦«*A*»⟧ *e* = ⟦«*B*»⟧ *e* ⟷ *A*=*B*
⟨*proof*⟩

2.6.3 The set Γ of Definition 1.1, constant terms used for coding

inductive *coding-tm* :: *tm* ⇒ *bool*
  where
    Ord: ∃ *i*. *x* = *ORD-OF i* ⟹ *coding-tm x*
  | *HPair*: *coding-tm x* ⟹ *coding-tm y* ⟹ *coding-tm* (*HPair x y*)

declare *coding-tm.intros* [*intro*]

lemma *coding-tm-Zero* [*intro*]: *coding-tm Zero*
  ⟨*proof*⟩

lemma *coding-tm-HTuple* [*intro*]: *coding-tm* (*HTuple k*)
  ⟨*proof*⟩

inductive-simps *coding-tm-HPair* [*simp*]: *coding-tm* (*HPair x y*)

lemma *quot-dbtm-coding* [*simp*]: *coding-tm* (*quot-dbtm t*)
  ⟨*proof*⟩

lemma *quot-dbfm-coding* [*simp*]: *coding-tm* (*quot-dbfm fm*)
  ⟨*proof*⟩

lemma *quot-fm-coding*: **fixes** *A*::*fm* **shows** *coding-tm* «*A*»
  ⟨*proof*⟩

inductive *coding-hf* :: *hf* ⇒ *bool*
  where
    Ord: ∃ *i*. *x* = *ord-of i* ⟹ *coding-hf x*
  | *HPair*: *coding-hf x* ⟹ *coding-hf y* ⟹ *coding-hf* (⟨*x*,*y*⟩)

declare *coding-hf.intros* [*intro*]

lemma *coding-hf-0* [*intro*]: *coding-hf 0*
  ⟨*proof*⟩

inductive-simps *coding-hf-hpair* [*simp*]: *coding-hf* (⟨*x*,*y*⟩)

lemma *coding-tm-hf* [*simp*]: *coding-tm t* ⟹ *coding-hf* ⟦*t*⟧*e*
  ⟨*proof*⟩

2.7 V-Coding for terms and formulas, for the Second Theorem

Infinite support, so we cannot use nominal primrec.

function *vquot-dbtm* :: *name set* ⇒ *dbtm* ⇒ *tm*
  where

vquot-dbtm V DBZero = Zero
| *vquot-dbtm V (DBVar name) = (if name ∈ V then Var name*
else ORD-OF (Suc (nat-of-name name)))
| *vquot-dbtm V (DBInd k) = HPair (HTuple 6) (ORD-OF k)*
| *vquot-dbtm V (DBEats t u) = HPair (HTuple 1) (HPair (vquot-dbtm V t)*
(vquot-dbtm V u))
⟨*proof*⟩

termination
⟨*proof*⟩

lemma *fresh-vquot-dbtm* [*simp*]: *i ♯ vquot-dbtm V tm ⟷ i ♯ tm ∨ i ∉ atom ‘ V*
⟨*proof*⟩

Infinite support, so we cannot use nominal primrec.

function *vquot-dbfm* :: *name set ⇒ dbfm ⇒ tm*
where
vquot-dbfm V (DBMem t u) = HPair (HTuple 0) (HPair (vquot-dbtm V t)
(vquot-dbtm V u))
| *vquot-dbfm V (DBEq t u) = HPair (HTuple 2) (HPair (vquot-dbtm V t) (vquot-dbtm*
V u))
| *vquot-dbfm V (DBDisj A B) = HPair (HTuple 3) (HPair (vquot-dbfm V A)*
(vquot-dbfm V B))
| *vquot-dbfm V (DBNeg A) = HPair (HTuple 4) (vquot-dbfm V A)*
| *vquot-dbfm V (DBEx A) = HPair (HTuple 5) (vquot-dbfm V A)*
⟨*proof*⟩

termination
⟨*proof*⟩

lemma *fresh-vquot-dbfm* [*simp*]: *i ♯ vquot-dbfm V fm ⟷ i ♯ fm ∨ i ∉ atom ‘ V*
⟨*proof*⟩

class *vquot* =
fixes *vquot* :: *'a ⇒ name set ⇒ tm* (‹⌊-⌋-› [*0,1000*]*1000*)

instantiation *tm* :: *vquot*
begin
definition *vquot-tm* :: *tm ⇒ name set ⇒ tm*
where *vquot-tm t V = vquot-dbtm V (trans-tm [] t)*
instance ⟨*proof*⟩
end

lemma *vquot-dbtm-empty* [*simp*]: *vquot-dbtm {} t = quot-dbtm t*
⟨*proof*⟩

lemma *vquot-tm-empty* [*simp*]: **fixes** *t*::*tm* **shows** ⌊*t*⌋{} = «*t*»
⟨*proof*⟩

lemma *vquot-dbtm-eq*: *atom ‘ V ∩ supp t = atom ‘ W ∩ supp t ⟹ vquot-dbtm V t = vquot-dbtm W t*
⟨*proof*⟩

instantiation *fm* :: *vquot*
begin
definition *vquot-fm* :: *fm ⇒ name set ⇒ tm*
where *vquot-fm A V = vquot-dbfm V* (*trans-fm* [] *A*)
instance ⟨*proof*⟩
end

lemma *vquot-fm-fresh* [*simp*]: **fixes** *A*::*fm* **shows** *i ♯ ⌊A⌋ V ⟷ i ♯ A ∨ i ∉ atom ‘ V*
⟨*proof*⟩

lemma *vquot-dbfm-empty* [*simp*]: *vquot-dbfm* {} *A = quot-dbfm A*
⟨*proof*⟩

lemma *vquot-fm-empty* [*simp*]: **fixes** *A*::*fm* **shows** *⌊A⌋*{} = «*A*»
⟨*proof*⟩

lemma *vquot-dbfm-eq*: *atom ‘ V ∩ supp A = atom ‘ W ∩ supp A ⟹ vquot-dbfm V A = vquot-dbfm W A*
⟨*proof*⟩

lemma *vquot-fm-insert*:
fixes *A*::*fm* **shows** *atom i ∉ supp A ⟹ ⌊A⌋*(*insert i V*) = *⌊A⌋ V*
⟨*proof*⟩

declare *HTuple.simps* [*simp del*]

end

Kapitel 3

Basic Predicates

theory *Predicates*
imports *SyntaxN*
begin

3.1 The Subset Relation

nominal-function *Subset* :: $tm \Rightarrow tm \Rightarrow fm$ (**infixr** ‹*SUBS*› *150*)
where $atom\ z \sharp (t,\ u) \Longrightarrow t\ SUBS\ u = All2\ z\ t\ ((Var\ z)\ IN\ u)$
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

declare *Subset.simps* [*simp del*]

lemma *Subset-fresh-iff* [*simp*]: $a \sharp t\ SUBS\ u \longleftrightarrow a \sharp t \wedge a \sharp u$
⟨*proof*⟩

lemma *eval-fm-Subset* [*simp*]: $eval\text{-}fm\ e\ (Subset\ t\ u) \longleftrightarrow (\llbracket t \rrbracket e \leq \llbracket u \rrbracket e)$
⟨*proof*⟩

lemma *subst-fm-Subset* [*simp*]: $(t\ SUBS\ u)(i::=x) = (subst\ i\ x\ t)\ SUBS\ (subst\ i\ x\ u)$
⟨*proof*⟩

lemma *Subset-I*:
assumes $insert\ ((Var\ i)\ IN\ t)\ H \vdash (Var\ i)\ IN\ u\ atom\ i \sharp (t,u)\ \forall B \in H.\ atom\ i \sharp B$
shows $H \vdash t\ SUBS\ u$
⟨*proof*⟩

lemma *Subset-D*:
assumes *major*: $H \vdash t\ SUBS\ u$ **and** *minor*: $H \vdash a\ IN\ t$ **shows** $H \vdash a\ IN\ u$
⟨*proof*⟩

lemma *Subset-E*: $H \vdash t\ SUBS\ u \Longrightarrow H \vdash a\ IN\ t \Longrightarrow insert\ (a\ IN\ u)\ H \vdash A \Longrightarrow H \vdash A$
⟨*proof*⟩

lemma *Subset-cong*: $H \vdash t\ EQ\ t' \Longrightarrow H \vdash u\ EQ\ u' \Longrightarrow H \vdash t\ SUBS\ u\ IFF\ t'\ SUBS\ u'$
⟨*proof*⟩

lemma *Set-MP*: $x\ SUBS\ y \in H \Longrightarrow z\ IN\ x \in H \Longrightarrow insert\ (z\ IN\ y)\ H \vdash A \Longrightarrow H \vdash A$
⟨*proof*⟩

lemma *Zero-Subset-I* [*intro*!]: $H \vdash Zero\ SUBS\ t$
⟨*proof*⟩

lemma *Zero-SubsetE*: $H \vdash A \Longrightarrow insert\ (Zero\ SUBS\ X)\ H \vdash A$
⟨*proof*⟩

lemma *Subset-Zero-D*:
assumes $H \vdash t\ SUBS\ Zero$ **shows** $H \vdash t\ EQ\ Zero$
⟨*proof*⟩

lemma *Subset-refl*: $H \vdash t\ SUBS\ t$
⟨*proof*⟩

lemma *Eats-Subset-Iff*: $H \vdash Eats\ x\ y\ SUBS\ z\ IFF\ (x\ SUBS\ z)\ AND\ (y\ IN\ z)$
⟨*proof*⟩

lemma *Eats-Subset-I* [*intro*!]: $H \vdash x\ SUBS\ z \Longrightarrow H \vdash y\ IN\ z \Longrightarrow H \vdash Eats\ x\ y\ SUBS\ z$
⟨*proof*⟩

lemma *Eats-Subset-E* [*intro*!]:
$insert\ (x\ SUBS\ z)\ (insert\ (y\ IN\ z)\ H) \vdash C \Longrightarrow insert\ (Eats\ x\ y\ SUBS\ z)\ H \vdash C$
⟨*proof*⟩

A surprising proof: a consequence of $?H \vdash Eats\ ?x\ ?y\ SUBS\ ?z\ IFF\ ?x\ SUBS\ ?z\ AND\ ?y\ IN\ ?z$ and reflexivity!

lemma *Subset-Eats-I* [*intro*!]: $H \vdash x\ SUBS\ Eats\ x\ y$
⟨*proof*⟩

lemma *SUCC-Subset-I* [*intro*!]: $H \vdash x\ SUBS\ z \Longrightarrow H \vdash x\ IN\ z \Longrightarrow H \vdash SUCC\ x\ SUBS\ z$
⟨*proof*⟩

lemma *SUCC-Subset-E* [*intro*!]:
$insert\ (x\ SUBS\ z)\ (insert\ (x\ IN\ z)\ H) \vdash C \Longrightarrow insert\ (SUCC\ x\ SUBS\ z)\ H \vdash C$
⟨*proof*⟩

lemma *Subset-trans0*: { *a SUBS b*, *b SUBS c* } ⊢ *a SUBS c*
⟨*proof*⟩

lemma *Subset-trans*: *H* ⊢ *a SUBS b* ⟹ *H* ⊢ *b SUBS c* ⟹ *H* ⊢ *a SUBS c*
⟨*proof*⟩

lemma *Subset-SUCC*: *H* ⊢ *a SUBS* (*SUCC a*)
⟨*proof*⟩

lemma *All2-Subset-lemma*: *atom l* ♯ (*k'*,*k*) ⟹ {*P*} ⊢ *P'* ⟹ {*All2 l k P*, *k' SUBS k*} ⊢ *All2 l k' P'*
⟨*proof*⟩

lemma *All2-Subset*: ⟦*H* ⊢ *All2 l k P*; *H* ⊢ *k' SUBS k*; {*P*} ⊢ *P'*; *atom l* ♯ (*k'*, *k*)⟧ ⟹ *H* ⊢ *All2 l k' P'*
⟨*proof*⟩

3.2 Extensionality

lemma *Extensionality*: *H* ⊢ *x EQ y IFF x SUBS y AND y SUBS x*
⟨*proof*⟩

lemma *Equality-I*: *H* ⊢ *y SUBS x* ⟹ *H* ⊢ *x SUBS y* ⟹ *H* ⊢ *x EQ y*
⟨*proof*⟩

lemma *EQ-imp-SUBS*: *insert* (*t EQ u*) *H* ⊢ (*t SUBS u*)
⟨*proof*⟩

lemma *EQ-imp-SUBS2*: *insert* (*u EQ t*) *H* ⊢ (*t SUBS u*)
⟨*proof*⟩

lemma *Equality-E*: *insert* (*t SUBS u*) (*insert* (*u SUBS t*) *H*) ⊢ *A* ⟹ *insert* (*t EQ u*) *H* ⊢ *A*
⟨*proof*⟩

3.3 The Disjointness Relation

The following predicate is defined in order to prove Lemma 2.3, Foundation

nominal-function *Disjoint* :: *tm* ⇒ *tm* ⇒ *fm*
where *atom z* ♯ (*t*, *u*) ⟹ *Disjoint t u* = *All2 z t* (*Neg* ((*Var z*) *IN u*))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

declare *Disjoint.simps* [*simp del*]

lemma *Disjoint-fresh-iff* [*simp*]: $a \sharp \mathit{Disjoint}\ t\ u \longleftrightarrow a \sharp t \wedge a \sharp u$
⟨*proof*⟩

lemma *subst-fm-Disjoint* [*simp*]:
$(\mathit{Disjoint}\ t\ u)(i::=x) = \mathit{Disjoint}\ (\mathit{subst}\ i\ x\ t)\ (\mathit{subst}\ i\ x\ u)$
⟨*proof*⟩

lemma *Disjoint-cong*: $H \vdash t\ \mathit{EQ}\ t' \Longrightarrow H \vdash u\ \mathit{EQ}\ u' \Longrightarrow H \vdash \mathit{Disjoint}\ t\ u\ \mathit{IFF}\ \mathit{Disjoint}\ t'\ u'$
⟨*proof*⟩

lemma *Disjoint-I*:
assumes $\mathit{insert}\ ((\mathit{Var}\ i)\ \mathit{IN}\ t)\ (\mathit{insert}\ ((\mathit{Var}\ i)\ \mathit{IN}\ u)\ H) \vdash \mathit{Fls}$
$\mathit{atom}\ i \sharp (t,u)\ \forall B \in H.\ \mathit{atom}\ i \sharp B$
shows $H \vdash \mathit{Disjoint}\ t\ u$
⟨*proof*⟩

lemma *Disjoint-E*:
assumes *major*: $H \vdash \mathit{Disjoint}\ t\ u$ **and** *minor*: $H \vdash a\ \mathit{IN}\ t\ H \vdash a\ \mathit{IN}\ u$ **shows** $H \vdash A$
⟨*proof*⟩

lemma *Disjoint-commute*: $\{\ \mathit{Disjoint}\ t\ u\ \} \vdash \mathit{Disjoint}\ u\ t$
⟨*proof*⟩

lemma *Disjoint-commute-I*: $H \vdash \mathit{Disjoint}\ t\ u \Longrightarrow H \vdash \mathit{Disjoint}\ u\ t$
⟨*proof*⟩

lemma *Disjoint-commute-D*: $\mathit{insert}\ (\mathit{Disjoint}\ t\ u)\ H \vdash A \Longrightarrow \mathit{insert}\ (\mathit{Disjoint}\ u\ t)\ H \vdash A$
⟨*proof*⟩

lemma *Zero-Disjoint-I1* [*iff*]: $H \vdash \mathit{Disjoint}\ \mathit{Zero}\ t$
⟨*proof*⟩

lemma *Zero-Disjoint-I2* [*iff*]: $H \vdash \mathit{Disjoint}\ t\ \mathit{Zero}$
⟨*proof*⟩

lemma *Disjoint-Eats-D1*: $\{\ \mathit{Disjoint}\ (\mathit{Eats}\ x\ y)\ z\ \} \vdash \mathit{Disjoint}\ x\ z$
⟨*proof*⟩

lemma *Disjoint-Eats-D2*: $\{\ \mathit{Disjoint}\ (\mathit{Eats}\ x\ y)\ z\ \} \vdash \mathit{Neg}(y\ \mathit{IN}\ z)$
⟨*proof*⟩

lemma *Disjoint-Eats-E*:
$\mathit{insert}\ (\mathit{Disjoint}\ x\ z)\ (\mathit{insert}\ (\mathit{Neg}(y\ \mathit{IN}\ z))\ H) \vdash A \Longrightarrow \mathit{insert}\ (\mathit{Disjoint}\ (\mathit{Eats}\ x\ y)\ z)\ H \vdash A$
⟨*proof*⟩

lemma *Disjoint-Eats-E2*:
insert (*Disjoint z x*) (*insert* (*Neg*(*y IN z*)) *H*) ⊢ *A* ⟹ *insert* (*Disjoint z* (*Eats x y*)) *H* ⊢ *A*
⟨*proof*⟩

lemma *Disjoint-Eats-Imp*: { *Disjoint x z*, *Neg*(*y IN z*) } ⊢ *Disjoint* (*Eats x y*) *z*
⟨*proof*⟩

lemma *Disjoint-Eats-I* [*intro*!]: *H* ⊢ *Disjoint x z* ⟹ *insert* (*y IN z*) *H* ⊢ *Fls* ⟹ *H* ⊢ *Disjoint* (*Eats x y*) *z*
⟨*proof*⟩

lemma *Disjoint-Eats-I2* [*intro*!]: *H* ⊢ *Disjoint z x* ⟹ *insert* (*y IN z*) *H* ⊢ *Fls* ⟹ *H* ⊢ *Disjoint z* (*Eats x y*)
⟨*proof*⟩

3.4 The Foundation Theorem

lemma *Foundation-lemma*:
assumes *i*: *atom i* ♯ *z*
shows { *All2 i z* (*Neg* (*Disjoint* (*Var i*) *z*)) } ⊢ *Neg* (*Var i IN z*) *AND Disjoint* (*Var i*) *z*
⟨*proof*⟩

theorem *Foundation*: *atom i* ♯ *z* ⟹ {} ⊢ *All2 i z* (*Neg* (*Disjoint* (*Var i*) *z*)) *IMP z EQ Zero*
⟨*proof*⟩

lemma *Mem-Neg-refl*: {} ⊢ *Neg* (*x IN x*)
⟨*proof*⟩

lemma *Mem-refl-E* [*intro*!]: *insert* (*x IN x*) *H* ⊢ *A*
⟨*proof*⟩

lemma *Mem-non-refl*: **assumes** *H* ⊢ *x IN x* **shows** *H* ⊢ *A*
⟨*proof*⟩

lemma *Mem-Neg-sym*: { *x IN y*, *y IN x* } ⊢ *Fls*
⟨*proof*⟩

lemma *Mem-not-sym*: *insert* (*x IN y*) (*insert* (*y IN x*) *H*) ⊢ *A*
⟨*proof*⟩

3.5 The Ordinal Property

nominal-function *OrdP* :: *tm* ⇒ *fm*
where ⟦*atom y* ♯ (*x*, *z*); *atom z* ♯ *x*⟧ ⟹
OrdP x = *All2 y x* ((*Var y*) *SUBS x AND All2 z* (*Var y*) ((*Var z*) *SUBS* (*Var*

y)))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *OrdP-fresh-iff* [*simp*]: $a \sharp OrdP\ x \longleftrightarrow a \sharp x$ (**is** *?thesis1*)
and *eval-fm-OrdP* [*simp*]: $eval\text{-}fm\ e\ (OrdP\ x) \longleftrightarrow Ord\ [\![x]\!]e$ (**is** *?thesis2*)
⟨*proof*⟩

lemma *subst-fm-OrdP* [*simp*]: $(OrdP\ t)(i::=x) = OrdP\ (subst\ i\ x\ t)$
⟨*proof*⟩

lemma *OrdP-cong*: $H \vdash x\ EQ\ x' \Longrightarrow H \vdash OrdP\ x\ IFF\ OrdP\ x'$
⟨*proof*⟩

lemma *OrdP-Mem-lemma*:
assumes *z*: $atom\ z \sharp (k,l)$ **and** *l*: $insert\ (OrdP\ k)\ H \vdash l\ IN\ k$
shows $insert\ (OrdP\ k)\ H \vdash l\ SUBS\ k\ AND\ All2\ z\ l\ (Var\ z\ SUBS\ l)$
⟨*proof*⟩

lemma *OrdP-Mem-E*:
assumes $atom\ z \sharp (k,l)$
$insert\ (OrdP\ k)\ H \vdash l\ IN\ k$
$insert\ (l\ SUBS\ k)\ (insert\ (All2\ z\ l\ (Var\ z\ SUBS\ l))\ H) \vdash A$
shows $insert\ (OrdP\ k)\ H \vdash A$
⟨*proof*⟩

lemma *OrdP-Mem-imp-Subset*:
assumes *k*: $H \vdash k\ IN\ l$ **and** *l*: $H \vdash OrdP\ l$ **shows** $H \vdash k\ SUBS\ l$
⟨*proof*⟩

lemma *SUCC-Subset-Ord-lemma*: $\{\ k'\ IN\ k,\ OrdP\ k\ \} \vdash SUCC\ k'\ SUBS\ k$
⟨*proof*⟩

lemma *SUCC-Subset-Ord*: $H \vdash k'\ IN\ k \Longrightarrow H \vdash OrdP\ k \Longrightarrow H \vdash SUCC\ k'\ SUBS\ k$
⟨*proof*⟩

lemma *OrdP-Trans-lemma*: $\{\ OrdP\ k,\ i\ IN\ j,\ j\ IN\ k\ \} \vdash i\ IN\ k$
⟨*proof*⟩

lemma *OrdP-Trans*: $H \vdash\ OrdP\ k \Longrightarrow H \vdash i\ IN\ j \Longrightarrow H \vdash j\ IN\ k \Longrightarrow H\ \vdash i\ IN\ k$
⟨*proof*⟩

lemma *Ord-IN-Ord0*:
assumes *l*: $H \vdash l\ IN\ k$

shows *insert* (*OrdP k*) *H* ⊢ *OrdP l*
⟨*proof*⟩

lemma *Ord-IN-Ord*: *H* ⊢ *l IN k* ⟹ *H* ⊢ *OrdP k* ⟹ *H* ⊢ *OrdP l*
⟨*proof*⟩

lemma *OrdP-I*:
assumes *insert* (*Var y IN x*) *H* ⊢ (*Var y*) *SUBS x*
and *insert* (*Var z IN Var y*) (*insert* (*Var y IN x*) *H*) ⊢ (*Var z*) *SUBS* (*Var y*)
and *atom y* ♯ (*x*, *z*) ∀ *B* ∈ *H*. *atom y* ♯ *B* *atom z* ♯ *x* ∀ *B* ∈ *H*. *atom z* ♯ *B*
shows *H* ⊢ *OrdP x*
⟨*proof*⟩

lemma *OrdP-Zero* [*simp*]: *H* ⊢ *OrdP Zero*
⟨*proof*⟩

lemma *OrdP-SUCC-I0*: { *OrdP k* } ⊢ *OrdP* (*SUCC k*)
⟨*proof*⟩

lemma *OrdP-SUCC-I*: *H* ⊢ *OrdP k* ⟹ *H* ⊢ *OrdP* (*SUCC k*)
⟨*proof*⟩

lemma *Zero-In-OrdP*: { *OrdP x* } ⊢ *x EQ Zero OR Zero IN x*
⟨*proof*⟩

lemma *OrdP-HPairE*: *insert* (*OrdP* (*HPair x y*)) *H* ⊢ *A*
⟨*proof*⟩

lemmas *OrdP-HPairEH* = *OrdP-HPairE OrdP-HPairE* [*THEN rotate2*] *OrdP-HPairE* [*THEN rotate3*] *OrdP-HPairE* [*THEN rotate4*] *OrdP-HPairE* [*THEN rotate5*] *OrdP-HPairE* [*THEN rotate6*] *OrdP-HPairE* [*THEN rotate7*] *OrdP-HPairE* [*THEN rotate8*] *OrdP-HPairE* [*THEN rotate9*] *OrdP-HPairE* [*THEN rotate10*]
declare *OrdP-HPairEH* [*intro*!]

lemma *Zero-Eq-HPairE*: *insert* (*Zero EQ HPair x y*) *H* ⊢ *A*
⟨*proof*⟩

lemmas *Zero-Eq-HPairEH* = *Zero-Eq-HPairE Zero-Eq-HPairE* [*THEN rotate2*] *Zero-Eq-HPairE* [*THEN rotate3*] *Zero-Eq-HPairE* [*THEN rotate4*] *Zero-Eq-HPairE* [*THEN rotate5*] *Zero-Eq-HPairE* [*THEN rotate6*] *Zero-Eq-HPairE* [*THEN rotate7*] *Zero-Eq-HPairE* [*THEN rotate8*] *Zero-Eq-HPairE* [*THEN rotate9*] *Zero-Eq-HPairE* [*THEN rotate10*]
declare *Zero-Eq-HPairEH* [*intro*!]

lemma *HPair-Eq-ZeroE*: *insert* (*HPair x y EQ Zero*) *H* ⊢ *A*
⟨*proof*⟩

lemmas *HPair-Eq-ZeroEH = HPair-Eq-ZeroE HPair-Eq-ZeroE [THEN rotate2] HPair-Eq-ZeroE [THEN rotate3] HPair-Eq-ZeroE [THEN rotate4] HPair-Eq-ZeroE [THEN rotate5]*
HPair-Eq-ZeroE [THEN rotate6] HPair-Eq-ZeroE [THEN rotate7] HPair-Eq-ZeroE [THEN rotate8] HPair-Eq-ZeroE [THEN rotate9] HPair-Eq-ZeroE [THEN rotate10]
declare *HPair-Eq-ZeroEH [intro!]*

3.6 Induction on Ordinals

lemma *OrdInd-lemma*:
assumes *j*: *atom* (*j*::*name*) ♯ (*i*,*A*)
shows { *OrdP* (*Var i*) } ⊢ (*All i* (*OrdP* (*Var i*) *IMP* ((*All2 j* (*Var i*) (*A*(*i*::= *Var j*))) *IMP A*))) *IMP A*
⟨*proof*⟩

lemma *OrdInd*:
assumes *j*: *atom* (*j*::*name*) ♯ (*i*,*A*)
and *x*: *H* ⊢ *OrdP* (*Var i*) **and** *step*: *H* ⊢ *All i* (*OrdP* (*Var i*) *IMP* (*All2 j* (*Var i*) (*A*(*i*::= *Var j*)) *IMP A*))
shows *H* ⊢ *A*
⟨*proof*⟩

lemma *OrdIndH*:
assumes *atom* (*j*::*name*) ♯ (*i*,*A*)
and *H* ⊢ *All i* (*OrdP* (*Var i*) *IMP* (*All2 j* (*Var i*) (*A*(*i*::= *Var j*)) *IMP A*))
shows *insert* (*OrdP* (*Var i*)) *H* ⊢ *A*
⟨*proof*⟩

3.7 Linearity of Ordinals

lemma *OrdP-linear-lemma*:
assumes *j*: *atom j* ♯ *i*
shows { *OrdP* (*Var i*) } ⊢ *All j* (*OrdP* (*Var j*) *IMP* (*Var i IN Var j OR Var i EQ Var j OR Var j IN Var i*))
(**is** - ⊢ *?scheme*)
⟨*proof*⟩

lemma *OrdP-linear-imp*: {} ⊢ *OrdP x IMP OrdP y IMP x IN y OR x EQ y OR y IN x*
⟨*proof*⟩

lemma *OrdP-linear*:
assumes *H* ⊢ *OrdP x H* ⊢ *OrdP y*
insert (*x IN y*) *H* ⊢ *A insert* (*x EQ y*) *H* ⊢ *A insert* (*y IN x*) *H* ⊢ *A*
shows *H* ⊢ *A*
⟨*proof*⟩

lemma *Zero-In-SUCC*: $\{OrdP\ k\} \vdash Zero\ IN\ SUCC\ k$
⟨*proof*⟩

3.8 The predicate *OrdNotEqP*

nominal-function *OrdNotEqP* :: $tm \Rightarrow tm \Rightarrow fm$ (**infixr** ‹*NEQ*› *150*)
where $OrdNotEqP\ x\ y = OrdP\ x\ AND\ OrdP\ y\ AND\ (x\ IN\ y\ OR\ y\ IN\ x)$
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *OrdNotEqP-fresh-iff* [*simp*]: $a \sharp OrdNotEqP\ x\ y \longleftrightarrow a \sharp x \wedge a \sharp y$
⟨*proof*⟩

lemma *eval-fm-OrdNotEqP* [*simp*]: $eval\text{-}fm\ e\ (OrdNotEqP\ x\ y) \longleftrightarrow Ord\ [\![x]\!]e \wedge Ord\ [\![y]\!]e \wedge [\![x]\!]e \neq [\![y]\!]e$
⟨*proof*⟩

lemma *OrdNotEqP-subst* [*simp*]: $(OrdNotEqP\ x\ y)(i::=t) = OrdNotEqP\ (subst\ i\ t\ x)\ (subst\ i\ t\ y)$
⟨*proof*⟩

lemma *OrdNotEqP-cong*: $H \vdash x\ EQ\ x' \Longrightarrow H \vdash y\ EQ\ y' \Longrightarrow H \vdash OrdNotEqP\ x\ y\ IFF\ OrdNotEqP\ x'\ y'$
⟨*proof*⟩

lemma *OrdNotEqP-self-contra*: $\{x\ NEQ\ x\} \vdash Fls$
⟨*proof*⟩

lemma *OrdNotEqP-OrdP-E*: $insert\ (OrdP\ x)\ (insert\ (OrdP\ y)\ H) \vdash A \Longrightarrow insert\ (x\ NEQ\ y)\ H \vdash A$
⟨*proof*⟩

lemma *OrdNotEqP-I*: $insert\ (x\ EQ\ y)\ H \vdash Fls \Longrightarrow H \vdash OrdP\ x \Longrightarrow H \vdash OrdP\ y \Longrightarrow H \vdash x\ NEQ\ y$
⟨*proof*⟩

declare *OrdNotEqP.simps* [*simp del*]

lemma *OrdNotEqP-imp-Neg-Eq*: $\{x\ NEQ\ y\} \vdash Neg\ (x\ EQ\ y)$
⟨*proof*⟩

lemma *OrdNotEqP-E*: $H \vdash x\ EQ\ y \Longrightarrow insert\ (x\ NEQ\ y)\ H \vdash A$
⟨*proof*⟩

3.9 Predecessor of an Ordinal

lemma *OrdP-set-max-lemma*:
assumes *j*: *atom* (*j*::*name*) ♯ *i* **and** *k*: *atom* (*k*::*name*) ♯ (*i*,*j*)
shows {} ⊢ (*Neg* (*Var i EQ Zero*) *AND* (*All2 j* (*Var i*) (*OrdP* (*Var j*)))) *IMP*
(*Ex j* (*Var j IN Var i AND* (*All2 k* (*Var i*) (*Var k SUBS Var j*))))
⟨*proof*⟩

lemma *OrdP-max-imp*:
assumes *j*: *atom j* ♯ (*x*) **and** *k*: *atom k* ♯ (*x*,*j*)
shows { *OrdP x*, *Neg* (*x EQ Zero*) } ⊢ *Ex j* (*Var j IN x AND* (*All2 k x* (*Var k SUBS Var j*)))
⟨*proof*⟩

declare *OrdP.simps* [*simp del*]

3.10 Case Analysis and Zero/SUCC Induction

lemma *OrdP-cases-lemma*:
assumes *p*: *atom p* ♯ *x*
shows { *OrdP x*, *Neg* (*x EQ Zero*) } ⊢ *Ex p* (*OrdP* (*Var p*) *AND x EQ SUCC* (*Var p*))
⟨*proof*⟩

lemma *OrdP-cases-disj*:
assumes *p*: *atom p* ♯ *x*
shows *insert* (*OrdP x*) *H* ⊢ *x EQ Zero OR Ex p* (*OrdP* (*Var p*) *AND x EQ SUCC* (*Var p*))
⟨*proof*⟩

lemma *OrdP-cases-E*:
⟦*insert* (*x EQ Zero*) *H* ⊢ *A*;
insert (*x EQ SUCC* (*Var k*)) (*insert* (*OrdP* (*Var k*)) *H*) ⊢ *A*;
atom k ♯ (*x*,*A*); ∀ *C* ∈ *H*. *atom k* ♯ *C*⟧
⟹ *insert* (*OrdP x*) *H* ⊢ *A*
⟨*proof*⟩

lemma *OrdInd2-lemma*:
{ *OrdP* (*Var i*), *A*(*i*::= *Zero*), (*All i* (*OrdP* (*Var i*) *IMP A IMP* (*A*(*i*::= *SUCC* (*Var i*))))) } ⊢ *A*
⟨*proof*⟩

lemma *OrdInd2*:
assumes *H* ⊢ *OrdP* (*Var i*)
and *H* ⊢ *A*(*i*::= *Zero*)
and *H* ⊢ *All i* (*OrdP* (*Var i*) *IMP A IMP* (*A*(*i*::= *SUCC* (*Var i*))))
shows *H* ⊢ *A*
⟨*proof*⟩

lemma *OrdInd2H*:
assumes $H \vdash A(i::= Zero)$
and $H \vdash All\ i\ (OrdP\ (Var\ i))\ IMP\ A\ IMP\ (A(i::= SUCC\ (Var\ i))))$
shows $insert\ (OrdP\ (Var\ i))\ H \vdash A$
⟨*proof*⟩

3.11 The predicate *HFun-Sigma*

To characterise the concept of a function using only bounded universal quantifiers.

See the note after the proof of Lemma 2.3.

definition *hfun-sigma* **where**
$hfun\text{-}sigma\ r \equiv \forall z \in r.\ \forall z' \in r.\ \exists x\ y\ x'\ y'.\ z = \langle x,y\rangle \wedge z' = \langle x',y'\rangle \wedge (x{=}x' \longrightarrow y{=}y')$

definition *hfun-sigma-ord* **where**
$hfun\text{-}sigma\text{-}ord\ r \equiv \forall z \in r.\ \forall z' \in r.\ \exists x\ y\ x'\ y'.\ z = \langle x,y\rangle \wedge z' = \langle x',y'\rangle \wedge Ord\ x \wedge Ord\ x' \wedge (x{=}x' \longrightarrow y{=}y')$

nominal-function *HFun-Sigma* :: $tm \Rightarrow fm$
where $[\![atom\ z \sharp (r,z',x,y,x',y');\ atom\ z' \sharp (r,x,y,x',y');$
$atom\ x \sharp (r,y,x',y');\ atom\ y \sharp (r,x',y');\ atom\ x' \sharp (r,y');\ atom\ y' \sharp (r)\ ]\!]$
$\Longrightarrow$
$HFun\text{-}Sigma\ r =$
$All2\ z\ r\ (All2\ z'\ r\ (Ex\ x\ (Ex\ y\ (Ex\ x'\ (Ex\ y'$
$(Var\ z\ EQ\ HPair\ (Var\ x)\ (Var\ y)\ AND\ Var\ z'\ EQ\ HPair\ (Var\ x')\ (Var\ y')$
$AND\ OrdP\ (Var\ x)\ AND\ OrdP\ (Var\ x')\ AND$
$((Var\ x\ EQ\ Var\ x')\ IMP\ (Var\ y\ EQ\ Var\ y'))))))))$
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *HFun-Sigma-fresh-iff* [*simp*]: $a \sharp HFun\text{-}Sigma\ r \longleftrightarrow a \sharp r$ (**is** *?thesis1*)
and *eval-fm-HFun-Sigma* [*simp*]:
$eval\text{-}fm\ e\ (HFun\text{-}Sigma\ r) \longleftrightarrow hfun\text{-}sigma\text{-}ord\ [\![r]\!]e$ (**is** *?thesis2*)
⟨*proof*⟩

lemma *HFun-Sigma-subst* [*simp*]: $(HFun\text{-}Sigma\ r)(i::=t) = HFun\text{-}Sigma\ (subst\ i\ t\ r)$
⟨*proof*⟩

lemma *HFun-Sigma-Zero*: $H \vdash HFun\text{-}Sigma\ Zero$
⟨*proof*⟩

lemma *Subset-HFun-Sigma*: $\{HFun\text{-}Sigma\ s,\ s'\ SUBS\ s\} \vdash HFun\text{-}Sigma\ s'$

⟨*proof*⟩

Captures the property of being a relation, using fewer variables than the full definition

lemma *HFun-Sigma-Mem-imp-HPair*:
assumes $H \vdash$ *HFun-Sigma r H* $\vdash$ *a IN r*
and *xy*: *atom x* ♯ (*y*,*a*,*r*) *atom y* ♯ (*a*,*r*)
shows $H \vdash$ (*Ex x* (*Ex y* (*a EQ HPair* (*Var x*) (*Var y*)))) (**is** - $\vdash$ *?concl*)
⟨*proof*⟩

3.12 The predicate *HDomain-Incl*

This is an internal version of $\forall x{\in}d.\ \exists y\ z.\ z \in r \wedge z = \langle x, y\rangle$.

nominal-function *HDomain-Incl* :: *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom x* ♯ (*r*,*d*,*y*,*z*); *atom y* ♯ (*r*,*d*,*z*); *atom z* ♯ (*r*,*d*)⟧ ⟹
HDomain-Incl r d = *All2 x d* (*Ex y* (*Ex z* (*Var z IN r AND Var z EQ HPair* (*Var x*) (*Var y*))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *HDomain-Incl-fresh-iff* [*simp*]:
a ♯ *HDomain-Incl r d* ⟷ *a* ♯ *r* ∧ *a* ♯ *d* (**is** *?thesis1*)
and *eval-fm-HDomain-Incl* [*simp*]:
eval-fm e (*HDomain-Incl r d*) ⟷ ⟦*d*⟧*e* ≤ *hdomain* ⟦*r*⟧*e* (**is** *?thesis2*)
⟨*proof*⟩

lemma *HDomain-Incl-subst* [*simp*]:
(*HDomain-Incl r d*)(*i*::=*t*) = *HDomain-Incl* (*subst i t r*) (*subst i t d*)
⟨*proof*⟩

lemma *HDomain-Incl-Subset-lemma*: { *HDomain-Incl r k*, *k′ SUBS k* } $\vdash$ *HDomain-Incl r k′*
⟨*proof*⟩

lemma *HDomain-Incl-Subset*: $H \vdash$ *HDomain-Incl r k* ⟹ $H \vdash$ *k′ SUBS k* ⟹ $H \vdash$ *HDomain-Incl r k′*
⟨*proof*⟩

lemma *HDomain-Incl-Mem-Ord*: $H \vdash$ *HDomain-Incl r k* ⟹ $H \vdash$ *k′ IN k* ⟹ $H \vdash$ *OrdP k* ⟹ $H \vdash$ *HDomain-Incl r k′*
⟨*proof*⟩

lemma *HDomain-Incl-Zero* [*simp*]: $H \vdash$ *HDomain-Incl r Zero*
⟨*proof*⟩

lemma *HDomain-Incl-Eats*: { *HDomain-Incl r d* } ⊢ *HDomain-Incl* (*Eats r* (*HPair d d'*)) (*SUCC d*)
⟨*proof*⟩

lemma *HDomain-Incl-Eats-I*: *H* ⊢ *HDomain-Incl r d* ⟹ *H* ⊢ *HDomain-Incl* (*Eats r* (*HPair d d'*)) (*SUCC d*)
⟨*proof*⟩

3.13 *HPair* is Provably Injective

lemma *Doubleton-E*:
assumes *insert* (*a EQ c*) (*insert* (*b EQ d*) *H*) ⊢ *A*
insert (*a EQ d*) (*insert* (*b EQ c*) *H*) ⊢ *A*
shows *insert* ((*Eats* (*Eats Zero b*) *a*) *EQ* (*Eats* (*Eats Zero d*) *c*)) *H* ⊢ *A*
⟨*proof*⟩

lemma *HFST*: {*HPair a b EQ HPair c d*} ⊢ *a EQ c*
⟨*proof*⟩

lemma *b-EQ-d-1*: {*a EQ c*, *a EQ d*, *b EQ c*} ⊢ *b EQ d*
⟨*proof*⟩

lemma *HSND*: {*HPair a b EQ HPair c d*} ⊢ *b EQ d*
⟨*proof*⟩

lemma *HPair-E* [*intro*!]:
assumes *insert* (*a EQ c*) (*insert* (*b EQ d*) *H*) ⊢ *A*
shows *insert* (*HPair a b EQ HPair c d*) *H* ⊢ *A*
⟨*proof*⟩

declare *HPair-E* [*THEN rotate2*, *intro*!]
declare *HPair-E* [*THEN rotate3*, *intro*!]
declare *HPair-E* [*THEN rotate4*, *intro*!]
declare *HPair-E* [*THEN rotate5*, *intro*!]
declare *HPair-E* [*THEN rotate6*, *intro*!]
declare *HPair-E* [*THEN rotate7*, *intro*!]
declare *HPair-E* [*THEN rotate8*, *intro*!]

lemma *HFun-Sigma-E*:
assumes *r*: *H* ⊢ *HFun-Sigma r*
and *b*: *H* ⊢ *HPair a b IN r*
and *b'*: *H* ⊢ *HPair a b' IN r*
shows *H* ⊢ *b EQ b'*
⟨*proof*⟩

3.14 *SUCC* is Provably Injective

lemma *SUCC-SUBS-lemma*: {*SUCC x SUBS SUCC y*} ⊢ *x SUBS y*

⟨*proof*⟩

lemma *SUCC-SUBS*: *insert* (*SUCC x SUBS SUCC y*) *H* ⊢ *x SUBS y*
⟨*proof*⟩

lemma *SUCC-inject*: *insert* (*SUCC x EQ SUCC y*) *H* ⊢ *x EQ y*
⟨*proof*⟩

lemma *SUCC-inject-E* [*intro*!]: *insert* (*x EQ y*) *H* ⊢ *A* ⟹ *insert* (*SUCC x EQ SUCC y*) *H* ⊢ *A*
⟨*proof*⟩

declare *SUCC-inject-E* [*THEN rotate2*, *intro*!]
declare *SUCC-inject-E* [*THEN rotate3*, *intro*!]
declare *SUCC-inject-E* [*THEN rotate4*, *intro*!]
declare *SUCC-inject-E* [*THEN rotate5*, *intro*!]
declare *SUCC-inject-E* [*THEN rotate6*, *intro*!]
declare *SUCC-inject-E* [*THEN rotate7*, *intro*!]
declare *SUCC-inject-E* [*THEN rotate8*, *intro*!]

lemma *OrdP-IN-SUCC-lemma*: {*OrdP x*, *y IN x*} ⊢ *SUCC y IN SUCC x*
⟨*proof*⟩

lemma *OrdP-IN-SUCC*: *H* ⊢ *OrdP x* ⟹ *H* ⊢ *y IN x* ⟹ *H* ⊢ *SUCC y IN SUCC x*
⟨*proof*⟩

lemma *OrdP-IN-SUCC-D-lemma*: {*OrdP x*, *SUCC y IN SUCC x*} ⊢ *y IN x*
⟨*proof*⟩

lemma *OrdP-IN-SUCC-D*: *H* ⊢ *OrdP x* ⟹ *H* ⊢ *SUCC y IN SUCC x* ⟹ *H* ⊢ *y IN x*
⟨*proof*⟩

lemma *OrdP-IN-SUCC-Iff*: *H* ⊢ *OrdP y* ⟹ *H* ⊢ *SUCC x IN SUCC y IFF x IN y*
⟨*proof*⟩

3.15 The predicate *LstSeqP*

lemma *hfun-sigma-ord-iff*: *hfun-sigma-ord s* ⟷ *OrdDom s* ∧ *hfun-sigma s*
⟨*proof*⟩

lemma *hfun-sigma-iff*: *hfun-sigma r* ⟷ *hfunction r* ∧ *hrelation r*
⟨*proof*⟩

lemma *Seq-iff*: *Seq r d* ⟷ *d* ≤ *hdomain r* ∧ *hfun-sigma r*
⟨*proof*⟩

lemma *LstSeq-iff*: *LstSeq s k y* $\longleftrightarrow$ *succ k* $\leq$ *hdomain s* $\wedge$ $\langle$*k*,*y*$\rangle$ $\in$ *s* $\wedge$ *hfun-sigma-ord s*
⟨*proof*⟩

nominal-function *LstSeqP* :: *tm* $\Rightarrow$ *tm* $\Rightarrow$ *tm* $\Rightarrow$ *fm*
where
LstSeqP s k y = *OrdP k AND HDomain-Incl s* (*SUCC k*) *AND HFun-Sigma s AND HPair k y IN s*
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *LstSeqP-fresh-iff* [*simp*]:
a $\sharp$ *LstSeqP s k y* $\longleftrightarrow$ *a* $\sharp$ *s* $\wedge$ *a* $\sharp$ *k* $\wedge$ *a* $\sharp$ *y* (**is** *?thesis1*)
and *eval-fm-LstSeqP* [*simp*]:
eval-fm e (*LstSeqP s k y*) $\longleftrightarrow$ *LstSeq* $[\![s]\!]e$ $[\![k]\!]e$ $[\![y]\!]e$ (**is** *?thesis2*)
⟨*proof*⟩

lemma *LstSeqP-subst* [*simp*]:
(*LstSeqP s k y*)(*i*::=*t*) = *LstSeqP* (*subst i t s*) (*subst i t k*) (*subst i t y*)
⟨*proof*⟩

lemma *LstSeqP-E*:
assumes *insert* (*HDomain-Incl s* (*SUCC k*))
(*insert* (*OrdP k*) (*insert* (*HFun-Sigma s*)
(*insert* (*HPair k y IN s*) *H*))) $\vdash$ *B*
shows *insert* (*LstSeqP s k y*) *H* $\vdash$ *B*
⟨*proof*⟩

declare *LstSeqP.simps* [*simp del*]

lemma *LstSeqP-cong*:
assumes *H* $\vdash$ *s EQ s′ H* $\vdash$ *k EQ k′ H* $\vdash$ *y EQ y′*
shows *H* $\vdash$ *LstSeqP s k y IFF LstSeqP s′ k′ y′*
⟨*proof*⟩

lemma *LstSeqP-OrdP*: *H* $\vdash$ *LstSeqP r k y* $\Longrightarrow$ *H* $\vdash$ *OrdP k*
⟨*proof*⟩

lemma *LstSeqP-Mem-lemma*: { *LstSeqP r k y*, *HPair k′ z IN r*, *k′ IN k* } $\vdash$ *LstSeqP r k′ z*
⟨*proof*⟩

lemma *LstSeqP-Mem*: *H* $\vdash$ *LstSeqP r k y* $\Longrightarrow$ *H* $\vdash$ *HPair k′ z IN r* $\Longrightarrow$ *H* $\vdash$ *k′ IN k* $\Longrightarrow$ *H* $\vdash$ *LstSeqP r k′ z*
⟨*proof*⟩

lemma *LstSeqP-imp-Mem*: $H \vdash LstSeqP\ s\ k\ y \Longrightarrow H \vdash HPair\ k\ y\ IN\ s$
⟨*proof*⟩

lemma *LstSeqP-SUCC*: $H \vdash LstSeqP\ r\ (SUCC\ d)\ y \Longrightarrow H \vdash HPair\ d\ z\ IN\ r \Longrightarrow H \vdash LstSeqP\ r\ d\ z$
⟨*proof*⟩

lemma *LstSeqP-EQ*: $[\![H \vdash LstSeqP\ s\ k\ y;\ H \vdash HPair\ k\ y'\ IN\ s]\!] \Longrightarrow H \vdash y\ EQ\ y'$
⟨*proof*⟩

end

Kapitel 4

Sigma-Formulas and Theorem 2.5

theory *Sigma*
imports *Predicates*
begin

4.1 Ground Terms and Formulas

definition *ground-aux* :: *tm* ⇒ *atom set* ⇒ *bool*
  where *ground-aux t S* ≡ (*supp t* ⊆ *S*)

abbreviation *ground* :: *tm* ⇒ *bool*
  where *ground t* ≡ *ground-aux t* {}

definition *ground-fm-aux* :: *fm* ⇒ *atom set* ⇒ *bool*
  where *ground-fm-aux A S* ≡ (*supp A* ⊆ *S*)

abbreviation *ground-fm* :: *fm* ⇒ *bool*
  where *ground-fm A* ≡ *ground-fm-aux A* {}

lemma *ground-aux-simps*[*simp*]:
  ground-aux Zero S = *True*
  ground-aux (*Var k*) *S* = (*if atom k* ∈ *S then True else False*)
  ground-aux (*Eats t u*) *S* = (*ground-aux t S* ∧ *ground-aux u S*)
⟨*proof*⟩

lemma *ground-fm-aux-simps*[*simp*]:
  ground-fm-aux Fls S = *True*
  ground-fm-aux (*t IN u*) *S* = (*ground-aux t S* ∧ *ground-aux u S*)
  ground-fm-aux (*t EQ u*) *S* = (*ground-aux t S* ∧ *ground-aux u S*)
  ground-fm-aux (*A OR B*) *S* = (*ground-fm-aux A S* ∧ *ground-fm-aux B S*)
  ground-fm-aux (*A AND B*) *S* = (*ground-fm-aux A S* ∧ *ground-fm-aux B S*)
  ground-fm-aux (*A IFF B*) *S* = (*ground-fm-aux A S* ∧ *ground-fm-aux B S*)

ground-fm-aux (*Neg A*) *S* = (*ground-fm-aux A S*)
ground-fm-aux (*Ex x A*) *S* = (*ground-fm-aux A* (*S* ∪ {*atom x*}))
⟨*proof*⟩

lemma *ground-fresh*[*simp*]:
ground t ⟹ *atom i* ♯ *t*
ground-fm A ⟹ *atom i* ♯ *A*
⟨*proof*⟩

4.2 Sigma Formulas

Section 2 material

4.2.1 Strict Sigma Formulas

Definition 2.1

inductive *ss-fm* :: *fm* ⇒ *bool* **where**
MemI: *ss-fm* (*Var i IN Var j*)
| *DisjI*: *ss-fm A* ⟹ *ss-fm B* ⟹ *ss-fm* (*A OR B*)
| *ConjI*: *ss-fm A* ⟹ *ss-fm B* ⟹ *ss-fm* (*A AND B*)
| *ExI*: *ss-fm A* ⟹ *ss-fm* (*Ex i A*)
| *All2I*: *ss-fm A* ⟹ *atom j* ♯ (*i*,*A*) ⟹ *ss-fm* (*All2 i* (*Var j*) *A*)

equivariance *ss-fm*

nominal-inductive *ss-fm*
avoids *ExI*: *i* | *All2I*: *i*
⟨*proof*⟩

declare *ss-fm.intros* [*intro*]

definition *Sigma-fm* :: *fm* ⇒ *bool*
where *Sigma-fm A* ⟷ (∃ *B*. *ss-fm B* ∧ *supp B* ⊆ *supp A* ∧ {} ⊢ *A IFF B*)

lemma *Sigma-fm-Iff*: ⟦{} ⊢ *B IFF A*; *supp A* ⊆ *supp B*; *Sigma-fm A*⟧ ⟹ *Sigma-fm B*
⟨*proof*⟩

lemma *ss-fm-imp-Sigma-fm* [*intro*]: *ss-fm A* ⟹ *Sigma-fm A*
⟨*proof*⟩

lemma *Sigma-fm-Fls* [*iff*]: *Sigma-fm Fls*
⟨*proof*⟩

4.2.2 Closure properties for Sigma-formulas

lemma

assumes *Sigma-fm A Sigma-fm B*
shows *Sigma-fm-AND* [*intro*!]: *Sigma-fm* (*A AND B*)
and *Sigma-fm-OR* [*intro*!]: *Sigma-fm* (*A OR B*)
and *Sigma-fm-Ex* [*intro*!]: *Sigma-fm* (*Ex i A*)
⟨*proof*⟩

lemma *Sigma-fm-All2-Var*:
assumes *H0*: *Sigma-fm A* **and** *ij*: *atom j* ♯ (*i*,*A*)
shows *Sigma-fm* (*All2 i* (*Var j*) *A*)
⟨*proof*⟩

4.3 Lemma 2.2: Atomic formulas are Sigma-formulas

lemma *Eq-Eats-Iff*:
assumes [*unfolded fresh-Pair*, *simp*]: *atom i* ♯ (*z*,*x*,*y*)
shows {} ⊢ *z EQ Eats x y IFF* (*All2 i z* (*Var i IN x OR Var i EQ y*)) *AND x SUBS z AND y IN z*
⟨*proof*⟩

lemma *Subset-Zero-sf*: *Sigma-fm* (*Var i SUBS Zero*)
⟨*proof*⟩

lemma *Eq-Zero-sf*: *Sigma-fm* (*Var i EQ Zero*)
⟨*proof*⟩

lemma *theorem-sf*: **assumes** {} ⊢ *A* **shows** *Sigma-fm A*
⟨*proof*⟩

The subset relation

lemma *Var-Subset-sf*: *Sigma-fm* (*Var i SUBS Var j*)
⟨*proof*⟩

lemma *Zero-Mem-sf*: *Sigma-fm* (*Zero IN Var i*)
⟨*proof*⟩

lemma *ijk*: *i* + *k* < *Suc* (*i* + *j* + *k*)
⟨*proof*⟩

lemma *All2-term-Iff-fresh*: *i*≠*j* ⟹ *atom j′* ♯ (*i*,*j*,*A*) ⟹
{} ⊢ (*All2 i* (*Var j*) *A*) *IFF Ex j′* (*Var j EQ Var j′ AND All2 i* (*Var j′*) *A*)
⟨*proof*⟩

lemma *Sigma-fm-All2-fresh*:
assumes *Sigma-fm A i*≠*j*
shows *Sigma-fm* (*All2 i* (*Var j*) *A*)
⟨*proof*⟩

lemma *Subset-Eats-sf*:
assumes ⋀*j*::*name*. *Sigma-fm* (*Var j IN t*)

and $\bigwedge k$::*name*. *Sigma-fm* (*Var k EQ u*)
shows *Sigma-fm* (*Var i SUBS Eats t u*)
⟨*proof*⟩

lemma *Eq-Eats-sf*:
assumes $\bigwedge j$::*name*. *Sigma-fm* (*Var j EQ t*)
and $\bigwedge k$::*name*. *Sigma-fm* (*Var k EQ u*)
shows *Sigma-fm* (*Var i EQ Eats t u*)
⟨*proof*⟩

lemma *Eats-Mem-sf*:
assumes $\bigwedge j$::*name*. *Sigma-fm* (*Var j EQ t*)
and $\bigwedge k$::*name*. *Sigma-fm* (*Var k EQ u*)
shows *Sigma-fm* (*Eats t u IN Var i*)
⟨*proof*⟩

lemma *Subset-Mem-sf-lemma*:
size t + *size u* < *n* ⟹ *Sigma-fm* (*t SUBS u*) ∧ *Sigma-fm* (*t IN u*)
⟨*proof*⟩

lemma *Subset-sf* [*iff*]: *Sigma-fm* (*t SUBS u*)
⟨*proof*⟩

lemma *Mem-sf* [*iff*]: *Sigma-fm* (*t IN u*)
⟨*proof*⟩

The equality relation is a Sigma-Formula

lemma *Equality-sf* [*iff*]: *Sigma-fm* (*t EQ u*)
⟨*proof*⟩

4.4 Universal Quantification Bounded by an Arbitrary Term

lemma *All2-term-Iff*: *atom i* ♯ *t* ⟹ *atom j* ♯ (*i*,*t*,*A*) ⟹
{} ⊢ (*All2 i t A*) *IFF Ex j* (*Var j EQ t AND All2 i* (*Var j*) *A*)
⟨*proof*⟩

lemma *Sigma-fm-All2* [*intro*!]:
assumes *Sigma-fm A atom i* ♯ *t*
shows *Sigma-fm* (*All2 i t A*)
⟨*proof*⟩

4.5 Lemma 2.3: Sequence-related concepts are Sigma-formulas

lemma *OrdP-sf* [*iff*]: *Sigma-fm* (*OrdP t*)
⟨*proof*⟩

lemma *OrdNotEqP-sf* [*iff*]: *Sigma-fm* (*OrdNotEqP t u*)
⟨*proof*⟩

lemma *HDomain-Incl-sf* [*iff*]: *Sigma-fm* (*HDomain-Incl t u*)
⟨*proof*⟩

lemma *HFun-Sigma-Iff*:
assumes *atom z* ♯ (*r*,*z*′,*x*,*y*,*x*′,*y*′) *atom z*′ ♯ (*r*,*x*,*y*,*x*′,*y*′)
atom x ♯ (*r*,*y*,*x*′,*y*′) *atom y* ♯ (*r*,*x*′,*y*′)
atom x′ ♯ (*r*,*y*′) *atom y*′ ♯ (*r*)
shows
{} ⊢*HFun-Sigma r IFF*
All2 z r (*All2 z*′ *r* (*Ex x* (*Ex y* (*Ex x*′ (*Ex y*′
(*Var z EQ HPair* (*Var x*) (*Var y*) *AND Var z*′ *EQ HPair* (*Var x*′) (*Var y*′)
AND OrdP (*Var x*) *AND OrdP* (*Var x*′) *AND*
((*Var x NEQ Var x*′) *OR* (*Var y EQ Var y*′))))))))
⟨*proof*⟩

lemma *HFun-Sigma-sf* [*iff*]: *Sigma-fm* (*HFun-Sigma t*)
⟨*proof*⟩

lemma *LstSeqP-sf* [*iff*]: *Sigma-fm* (*LstSeqP t u v*)
⟨*proof*⟩

4.6 A Key Result: Theorem 2.5

4.6.1 Preparation

To begin, we require some facts connecting quantification and ground terms.

lemma *obtain-const-tm*: **obtains** *t* **where** ⟦*t*⟧*e* = *x ground t*
⟨*proof*⟩

lemma *ex-eval-fm-iff-exists-tm*:
eval-fm e (*Ex k A*) ⟷ (∃ *t*. *eval-fm e* (*A*(*k*::=*t*)) ∧ *ground t*)
⟨*proof*⟩

In a negative context, the formulation above is actually weaker than this one.

lemma *ex-eval-fm-iff-exists-tm*′:
eval-fm e (*Ex k A*) ⟷ (∃ *t*. *eval-fm e* (*A*(*k*::=*t*)))
⟨*proof*⟩

A ground term defines a finite set of ground terms, its elements.

nominal-function *elts* :: *tm* ⇒ *tm set* **where**
elts Zero = {}
| *elts* (*Var k*) = {}

| *elts* (*Eats t u*) = *insert u* (*elts t*)
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *eval-fm-All2-Eats*:
atom i ♯ (*t*,*u*) ⟹
eval-fm e (*All2 i* (*Eats t u*) *A*) ⟷ *eval-fm e* (*A*(*i*::=*u*)) ∧ *eval-fm e* (*All2 i t A*)
⟨*proof*⟩

The term *t* must be ground, since *elts* doesn't handle variables.

lemma *eval-fm-All2-Iff-elts*:
ground t ⟹ *eval-fm e* (*All2 i t A*) ⟷ (∀ *u* ∈ *elts t*. *eval-fm e* (*A*(*i*::=*u*)))
⟨*proof*⟩

lemma *prove-elts-imp-prove-All2*:
ground t ⟹ (⋀*u*. *u* ∈ *elts t* ⟹ {} ⊢ *A*(*i*::=*u*)) ⟹ {} ⊢ *All2 i t A*
⟨*proof*⟩

4.6.2 The base cases: ground atomic formulas

lemma *ground-prove*:
⟦*size t* + *size u* < *n*; *ground t*; *ground u*⟧
⟹ (⟦*t*⟧*e* ≤ ⟦*u*⟧*e* ⟶ {} ⊢ *t SUBS u*) ∧ (⟦*t*⟧*e* ∈ ⟦*u*⟧*e* ⟶ {} ⊢ *t IN u*)
⟨*proof*⟩

lemma
assumes *ground t ground u*
shows *ground-prove-SUBS*: ⟦*t*⟧*e* ≤ ⟦*u*⟧*e* ⟹ {} ⊢ *t SUBS u*
and *ground-prove-IN*: ⟦*t*⟧*e* ∈ ⟦*u*⟧*e* ⟹ {} ⊢ *t IN u*
and *ground-prove-EQ*: ⟦*t*⟧*e* = ⟦*u*⟧*e* ⟹ {} ⊢ *t EQ u*
⟨*proof*⟩

lemma *ground-subst*:
ground-aux tm (*insert* (*atom i*) *S*) ⟹ *ground t* ⟹ *ground-aux* (*subst i t tm*) *S*
⟨*proof*⟩

lemma *ground-subst-fm*:
ground-fm-aux A (*insert* (*atom i*) *S*) ⟹ *ground t* ⟹ *ground-fm-aux* (*A*(*i*::=*t*)) *S*
⟨*proof*⟩

lemma *elts-imp-ground*: *u* ∈ *elts t* ⟹ *ground-aux t S* ⟹ *ground-aux u S*
⟨*proof*⟩

4.6.3 Sigma-Eats Formulas

inductive *se-fm* :: *fm* ⇒ *bool* **where**

MemI: *se-fm* (*t IN u*)
| *DisjI*: *se-fm A* $\Longrightarrow$ *se-fm B* $\Longrightarrow$ *se-fm* (*A OR B*)
| *ConjI*: *se-fm A* $\Longrightarrow$ *se-fm B* $\Longrightarrow$ *se-fm* (*A AND B*)
| *ExI*: *se-fm A* $\Longrightarrow$ *se-fm* (*Ex i A*)
| *All2I*: *se-fm A* $\Longrightarrow$ *atom i* $\sharp$ *t* $\Longrightarrow$ *se-fm* (*All2 i t A*)

equivariance *se-fm*

nominal-inductive *se-fm*
avoids *ExI*: *i* | *All2I*: *i*
⟨*proof*⟩

declare *se-fm.intros* [*intro*]

lemma *subst-fm-in-se-fm*: *se-fm A* $\Longrightarrow$ *se-fm* (*A*(*k*::=*x*))
⟨*proof*⟩
lemma *ground-se-fm-induction*:
ground-fm α $\Longrightarrow$ *size* α < *n* $\Longrightarrow$ *se-fm* α $\Longrightarrow$ *eval-fm e* α $\Longrightarrow$ {} $\vdash$ α
⟨*proof*⟩

lemma *ss-imp-se-fm*: *ss-fm A* $\Longrightarrow$ *se-fm A*
⟨*proof*⟩

lemma *se-fm-imp-thm*: ⟦*se-fm A*; *ground-fm A*; *eval-fm e A*⟧ $\Longrightarrow$ {} $\vdash$ *A*
⟨*proof*⟩

Theorem 2.5

theorem *Sigma-fm-imp-thm*: ⟦*Sigma-fm A*; *ground-fm A*; *eval-fm e0 A*⟧ $\Longrightarrow$ {} $\vdash$ *A*
⟨*proof*⟩

end

Kapitel 5

Predicates for Terms, Formulas and Substitution

theory *Coding-Predicates*
imports *Coding Sigma*
begin

declare *succ-iff* [*simp del*]

This material comes from Section 3, greatly modified for de Bruijn syntax.

5.1 Predicates for atomic terms

5.1.1 Free Variables

definition *is-Var* :: *hf* ⇒ *bool* **where** *is-Var x* ≡ *Ord x* ∧ *0* ∈ *x*

definition *VarP* :: *tm* ⇒ *fm* **where** *VarP x* ≡ *OrdP x AND Zero IN x*

lemma *VarP-eqvt* [*eqvt*]: (*p* ∙ *VarP x*) = *VarP* (*p* ∙ *x*)
⟨*proof*⟩

lemma *VarP-fresh-iff* [*simp*]: *a* ♯ *VarP x* ⟷ *a* ♯ *x*
⟨*proof*⟩

lemma *eval-fm-VarP* [*simp*]: *eval-fm e* (*VarP x*) ⟷ *is-Var* ⟦*x*⟧*e*
⟨*proof*⟩

lemma *VarP-sf* [*iff*]: *Sigma-fm* (*VarP x*)
⟨*proof*⟩

lemma *VarP-subst* [*simp*]: (*VarP x*)(*i*::=*t*) = *VarP* (*subst i t x*)
⟨*proof*⟩

lemma *VarP-cong*: $H \vdash x\ EQ\ x' \Longrightarrow H \vdash VarP\ x\ IFF\ VarP\ x'$
⟨*proof*⟩

lemma *VarP-HPairE* [*intro*!]: $insert\ (VarP\ (HPair\ x\ y))\ H \vdash A$
⟨*proof*⟩

lemma *is-Var-succ-iff* [*simp*]: $is\text{-}Var\ (succ\ x) = Ord\ x$
⟨*proof*⟩

lemma *is-Var-q-Var* [*iff*]: $is\text{-}Var\ (q\text{-}Var\ i)$
⟨*proof*⟩

definition *decode-Var* :: $hf \Rightarrow name$
where $decode\text{-}Var\ x \equiv name\text{-}of\text{-}nat\ (nat\text{-}of\text{-}ord\ (pred\ x))$

lemma *decode-Var-q-Var* [*simp*]: $decode\text{-}Var\ (q\text{-}Var\ i) = i$
⟨*proof*⟩

lemma *is-Var-imp-decode-Var*: $is\text{-}Var\ x \Longrightarrow x = [\![«Var\ (decode\text{-}Var\ x)»]\!]\ e$
⟨*proof*⟩

lemma *is-Var-iff*: $is\text{-}Var\ v \longleftrightarrow v = succ\ (ord\text{-}of\ (nat\text{-}of\text{-}name\ (decode\text{-}Var\ v)))$
⟨*proof*⟩

lemma *decode-Var-inject* [*simp*]: $is\text{-}Var\ v \Longrightarrow is\text{-}Var\ v' \Longrightarrow decode\text{-}Var\ v = decode\text{-}Var\ v' \longleftrightarrow v{=}v'$
⟨*proof*⟩

5.1.2 De Bruijn Indexes

definition *is-Ind* :: $hf \Rightarrow bool$
where $is\text{-}Ind\ x \equiv (\exists\, m.\ Ord\ m \wedge x = \langle htuple\ 6,\ m\rangle)$

abbreviation *Q-Ind* :: $tm \Rightarrow tm$
where $Q\text{-}Ind\ k \equiv HPair\ (HTuple\ 6)\ k$

nominal-function *IndP* :: $tm \Rightarrow fm$
where $atom\ m \sharp x \Longrightarrow$
$IndP\ x = Ex\ m\ (OrdP\ (Var\ m)\ AND\ x\ EQ\ HPair\ (HTuple\ 6)\ (Var\ m))$
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *IndP-fresh-iff* [*simp*]: $a \sharp IndP\ x \longleftrightarrow a \sharp x$ (**is** *?thesis1*)
and *eval-fm-IndP* [*simp*]: $eval\text{-}fm\ e\ (IndP\ x) \longleftrightarrow is\text{-}Ind\ [\![x]\!]e$ (**is** *?thesis2*)
and *IndP-sf* [*iff*]: $Sigma\text{-}fm\ (IndP\ x)$ (**is** *?thsf*)
and *OrdP-IndP-Q-Ind*: $\{OrdP\ x\} \vdash IndP\ (Q\text{-}Ind\ x)$ (**is** *?thqind*)

⟨*proof*⟩

lemma *IndP-Q-Ind*: $H \vdash OrdP\ x \Longrightarrow H \vdash IndP\ (Q\text{-}Ind\ x)$
⟨*proof*⟩

lemma *subst-fm-IndP* [*simp*]: $(IndP\ t)(i::=x) = IndP\ (subst\ i\ x\ t)$
⟨*proof*⟩

lemma *IndP-cong*: $H \vdash x\ EQ\ x' \Longrightarrow H \vdash IndP\ x\ IFF\ IndP\ x'$
⟨*proof*⟩

definition *decode-Ind* :: *hf* ⇒ *nat*
where *decode-Ind x* ≡ *nat-of-ord* (*hsnd x*)

lemma *is-Ind-pair-iff* [*simp*]: *is-Ind* ⟨*x*, *y*⟩ ⟷ *x* = *htuple 6* ∧ *Ord y*
⟨*proof*⟩

5.1.3 Various syntactic lemmas

lemma *eval-Var-q*: ⟦« *Var i*»⟧ *e* = *q-Var i*
⟨*proof*⟩

lemma *is-Var-eval-Var* [*simp*]: *is-Var* ⟦« *Var i*»⟧*e*
⟨*proof*⟩

5.2 The predicate *SeqCTermP*, for Terms and Constants

definition *SeqCTerm* :: *bool* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SeqCTerm vf s k t* ≡ *BuildSeq* (λ*u*. *u=0* ∨ *vf* ∧ *is-Var u*) (λ*u v w*. *u* = *q-Eats v w*) *s k t*

nominal-function *SeqCTermP* :: *bool* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom l* ♯ (*s*,*k*,*sl*,*m*,*n*,*sm*,*sn*); *atom sl* ♯ (*s*,*m*,*n*,*sm*,*sn*);
atom m ♯ (*s*,*n*,*sm*,*sn*); *atom n* ♯ (*s*,*sm*,*sn*);
atom sm ♯ (*s*,*sn*); *atom sn* ♯ (*s*)⟧ ⟹
SeqCTermP vf s k t =
LstSeqP s k t AND
All2 l (*SUCC k*) (*Ex sl* (*HPair* (*Var l*) (*Var sl*) *IN s AND*
(*Var sl EQ Zero OR* (*if vf then VarP* (*Var sl*) *else Fls*) *OR*
Ex m (*Ex n* (*Ex sm* (*Ex sn* (*Var m IN Var l AND Var n IN Var l AND*
HPair (*Var m*) (*Var sm*) *IN s AND HPair* (*Var n*) (*Var sn*) *IN s AND*
Var sl EQ Q-Eats (*Var sm*) (*Var sn*))))))))
⟨*proof*⟩

nominal-termination (*eqvt*)

⟨*proof*⟩

lemma
shows *SeqCTermP-fresh-iff* [*simp*]:
$a \sharp SeqCTermP\ vf\ s\ k\ t \longleftrightarrow a \sharp s \wedge a \sharp k \wedge a \sharp t$ (**is** *?thesis1*)
and *eval-fm-SeqCTermP* [*simp*]:
eval-fm e (*SeqCTermP vf s k t*) $\longleftrightarrow$ *SeqCTerm vf* $[\![s]\!]e\ [\![k]\!]e\ [\![t]\!]e$ (**is** *?thesis2*)
and *SeqCTermP-sf* [*iff*]:
Sigma-fm (*SeqCTermP vf s k t*) (**is** *?thsf*)
and *SeqCTermP-imp-LstSeqP*:
{ *SeqCTermP vf s k t* } ⊢ *LstSeqP s k t* (**is** *?thlstseq*)
and *SeqCTermP-imp-OrdP* [*simp*]:
{ *SeqCTermP vf s k t* } ⊢ *OrdP k* (**is** *?thord*)
⟨*proof*⟩

lemma *SeqCTermP-subst* [*simp*]:
(*SeqCTermP vf s k t*)(*j*::=*w*) = *SeqCTermP vf* (*subst j w s*) (*subst j w k*) (*subst j w t*)
⟨*proof*⟩

declare *SeqCTermP.simps* [*simp del*]

abbreviation *SeqTerm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SeqTerm* ≡ *SeqCTerm True*

abbreviation *SeqTermP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where *SeqTermP* ≡ *SeqCTermP True*

abbreviation *SeqConst* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SeqConst* ≡ *SeqCTerm False*

abbreviation *SeqConstP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where *SeqConstP* ≡ *SeqCTermP False*

lemma *SeqConst-imp-SeqTerm*: *SeqConst s k x* ⟹ *SeqTerm s k x*
⟨*proof*⟩

lemma *SeqConstP-imp-SeqTermP*: {*SeqConstP s k t*} ⊢ *SeqTermP s k t*
⟨*proof*⟩

5.3 The predicates *TermP* and *ConstP*

5.3.1 Definition

definition *CTerm* :: *bool* ⇒ *hf* ⇒ *bool*
where *CTerm vf t* ≡ (∃ *s k*. *SeqCTerm vf s k t*)

nominal-function *CTermP* :: *bool* ⇒ *tm* ⇒ *fm*
where ⟦*atom k* ♯ (*s*,*t*); *atom s* ♯ *t*⟧ ⟹

CTermP vf t = Ex s (Ex k (SeqCTermP vf (Var s) (Var k) t))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *CTermP-fresh-iff* [*simp*]: *a ♯ CTermP vf t ⟷ a ♯ t* (**is** *?thesis1*)
and *eval-fm-CTermP* [*simp*] :*eval-fm e (CTermP vf t) ⟷ CTerm vf ⟦t⟧e* (**is** *?thesis2*)
and *CTermP-sf* [*iff*]: *Sigma-fm (CTermP vf t)* (**is** *?thsf*)
⟨*proof*⟩

lemma *CTermP-subst* [*simp*]: *(CTermP vf i)(j::=w) = CTermP vf (subst j w i)*
⟨*proof*⟩

abbreviation *Term* :: *hf ⇒ bool*
where *Term ≡ CTerm True*

abbreviation *TermP* :: *tm ⇒ fm*
where *TermP ≡ CTermP True*

abbreviation *Const* :: *hf ⇒ bool*
where *Const ≡ CTerm False*

abbreviation *ConstP* :: *tm ⇒ fm*
where *ConstP ≡ CTermP False*

5.3.2 Correctness: It Corresponds to Quotations of Real Terms

lemma *wf-Term-quot-dbtm* [*simp*]: *wf-dbtm u ⟹ Term ⟦quot-dbtm u⟧e*
⟨*proof*⟩

corollary *Term-quot-tm* [*iff*]: **fixes** *t* :: *tm* **shows** *Term ⟦«t»⟧e*
⟨*proof*⟩

lemma *SeqCTerm-imp-wf-dbtm*:
assumes *SeqCTerm vf s k x*
shows *∃ t::dbtm. wf-dbtm t ∧ x = ⟦quot-dbtm t⟧e*
⟨*proof*⟩

corollary *Term-imp-wf-dbtm*:
assumes *Term x* **obtains** *t* **where** *wf-dbtm t x = ⟦quot-dbtm t⟧e*
⟨*proof*⟩

corollary *Term-imp-is-tm*: **assumes** *Term x* **obtains** *t::tm* **where** *x = ⟦«t»⟧ e*
⟨*proof*⟩

lemma *Term-Var*: *Term (q-Var i)*

⟨*proof*⟩

lemma *Term-Eats*: **assumes** *x*: *Term x* **and** *y*: *Term y* **shows** *Term* (*q-Eats x y*)
⟨*proof*⟩

5.3.3 Correctness properties for constants

lemma *Const-imp-Term*: *Const x* ⟹ *Term x*
⟨*proof*⟩

lemma *Const-0*: *Const 0*
⟨*proof*⟩

lemma *ConstP-imp-TermP*: {*ConstP t*} ⊢ *TermP t*
⟨*proof*⟩

5.4 Abstraction over terms

definition *SeqStTerm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SeqStTerm v u x x′ s k* ≡
is-Var v ∧ *BuildSeq2* (λ*y y′*. (*is-Ind y* ∨ *Ord y*) ∧ *y′* = (*if y*=*v then u else y*))
(λ*u u′ v v′ w w′*. *u* = *q-Eats v w* ∧ *u′* = *q-Eats v′ w′*) *s k x x′*

definition *AbstTerm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *AbstTerm v i x x′* ≡ *Ord i* ∧ (∃ *s k*. *SeqStTerm v* (*q-Ind i*) *x x′ s k*)

5.4.1 Defining the syntax: quantified body

nominal-function *SeqStTermP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom l* ♯ (*s*,*k*,*v*,*i*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*);
atom sl ♯ (*s*,*v*,*i*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*); *atom sl′* ♯ (*s*,*v*,*i*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*);
atom m ♯ (*s*,*n*,*sm*,*sm′*,*sn*,*sn′*); *atom n* ♯ (*s*,*sm*,*sm′*,*sn*,*sn′*);
atom sm ♯ (*s*,*sm′*,*sn*,*sn′*); *atom sm′* ♯ (*s*,*sn*,*sn′*);
atom sn ♯ (*s*,*sn′*); *atom sn′* ♯ *s*⟧ ⟹
SeqStTermP v i t u s k =
VarP v AND LstSeqP s k (*HPair t u*) *AND*
All2 l (*SUCC k*) (*Ex sl* (*Ex sl′* (*HPair* (*Var l*) (*HPair* (*Var sl*) (*Var sl′*)) *IN s AND*
(((*Var sl EQ v AND Var sl′ EQ i*) *OR*
((*IndP* (*Var sl*) *OR Var sl NEQ v*) *AND Var sl′ EQ Var sl*)) *OR*
Ex m (*Ex n* (*Ex sm* (*Ex sm′* (*Ex sn* (*Ex sn′* (*Var m IN Var l AND Var n IN Var l AND*
HPair (*Var m*) (*HPair* (*Var sm*) (*Var sm′*)) *IN s AND*
HPair (*Var n*) (*HPair* (*Var sn*) (*Var sn′*)) *IN s AND*
Var sl EQ Q-Eats (*Var sm*) (*Var sn*) *AND*
Var sl′ EQ Q-Eats (*Var sm′*) (*Var sn′*)))))))))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *SeqStTermP-fresh-iff* [*simp*]:
a ♯ *SeqStTermP v i t u s k* ⟷ *a* ♯ *v* ∧ *a* ♯ *i* ∧ *a* ♯ *t* ∧ *a* ♯ *u* ∧ *a* ♯ *s* ∧ *a* ♯ *k*
(**is** *?thesis1*)
and *eval-fm-SeqStTermP* [*simp*]:
eval-fm e (*SeqStTermP v i t u s k*) ⟷ *SeqStTerm* ⟦*v*⟧*e* ⟦*i*⟧*e* ⟦*t*⟧*e* ⟦*u*⟧*e* ⟦*s*⟧*e* ⟦*k*⟧*e* (**is** *?thesis2*)
and *SeqStTermP-sf* [*iff*]:
Sigma-fm (*SeqStTermP v i t u s k*) (**is** *?thsf*)
and *SeqStTermP-imp-OrdP*:
{ *SeqStTermP v i t u s k* } ⊢ *OrdP k* (**is** *?thord*)
and *SeqStTermP-imp-VarP*:
{ *SeqStTermP v i t u s k* } ⊢ *VarP v* (**is** *?thvar*)
and *SeqStTermP-imp-LstSeqP*:
{ *SeqStTermP v i t u s k* } ⊢ *LstSeqP s k* (*HPair t u*) (**is** *?thlstseq*)
⟨*proof*⟩

lemma *SeqStTermP-subst* [*simp*]:
(*SeqStTermP v i t u s k*)(*j*::=*w*) =
SeqStTermP (*subst j w v*) (*subst j w i*) (*subst j w t*) (*subst j w u*) (*subst j w s*) (*subst j w k*)
⟨*proof*⟩

lemma *SeqStTermP-cong*:
⟦*H* ⊢ *t EQ t′*; *H* ⊢ *u EQ u′*; *H* ⊢ *s EQ s′*; *H* ⊢ *k EQ k′*⟧
⟹ *H* ⊢ *SeqStTermP v i t u s k IFF SeqStTermP v i t′ u′ s′ k′*
⟨*proof*⟩

declare *SeqStTermP.simps* [*simp del*]

5.4.2 Defining the syntax: main predicate

nominal-function *AbstTermP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom s* ♯ (*v*,*i*,*t*,*u*,*k*); *atom k* ♯ (*v*,*i*,*t*,*u*)⟧ ⟹
AbstTermP v i t u =
OrdP i AND Ex s (*Ex k* (*SeqStTermP v* (*Q-Ind i*) *t u* (*Var s*) (*Var k*)))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *AbstTermP-fresh-iff* [*simp*]:
a ♯ *AbstTermP v i t u* ⟷ *a* ♯ *v* ∧ *a* ♯ *i* ∧ *a* ♯ *t* ∧ *a* ♯ *u* (**is** *?thesis1*)
and *eval-fm-AbstTermP* [*simp*]:
eval-fm e (*AbstTermP v i t u*) ⟷ *AbstTerm* ⟦*v*⟧*e* ⟦*i*⟧*e* ⟦*t*⟧*e* ⟦*u*⟧*e* (**is** *?thesis2*)

and *AbstTermP-sf* [*iff*]:
Sigma-fm (*AbstTermP v i t u*) (**is** *?thsf*)
⟨*proof*⟩

lemma *AbstTermP-subst* [*simp*]:
(*AbstTermP v i t u*)(*j*::=*w*) = *AbstTermP* (*subst j w v*) (*subst j w i*) (*subst j w t*) (*subst j w u*)
⟨*proof*⟩

declare *AbstTermP.simps* [*simp del*]

5.4.3 Correctness: It Coincides with Abstraction over real terms

lemma *not-is-Var-is-Ind*: *is-Var v* ⟹ ¬ *is-Ind v*
⟨*proof*⟩

lemma *AbstTerm-imp-abst-dbtm*:
assumes *AbstTerm v i x x′*
shows ∃ *t*. *x* = ⟦*quot-dbtm t*⟧*e* ∧
x′ = ⟦*quot-dbtm* (*abst-dbtm* (*decode-Var v*) (*nat-of-ord i*) *t*)⟧*e*
⟨*proof*⟩

lemma *AbstTerm-abst-dbtm*:
AbstTerm (*q-Var i*) (*ord-of n*) ⟦*quot-dbtm t*⟧*e*
⟦*quot-dbtm* (*abst-dbtm i n t*)⟧*e*
⟨*proof*⟩

5.5 Substitution over terms

definition *SubstTerm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SubstTerm v u x x′* ≡ *Term u* ∧ (∃ *s k*. *SeqStTerm v u x x′ s k*)

5.5.1 Defining the syntax

nominal-function *SubstTermP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom s* ♯ (*v*,*i*,*t*,*u*,*k*); *atom k* ♯ (*v*,*i*,*t*,*u*)⟧ ⟹
SubstTermP v i t u = *TermP i AND Ex s* (*Ex k* (*SeqStTermP v i t u* (*Var s*) (*Var k*)))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *SubstTermP-fresh-iff* [*simp*]:
a ♯ *SubstTermP v i t u* ⟷ *a* ♯ *v* ∧ *a* ♯ *i* ∧ *a* ♯ *t* ∧ *a* ♯ *u* (**is** *?thesis1*)
and *eval-fm-SubstTermP* [*simp*]:

eval-fm e (SubstTermP v i t u) $\longleftrightarrow$ *SubstTerm* $[\![v]\!]e$ $[\![i]\!]e$ $[\![t]\!]e$ $[\![u]\!]e$ (**is** *?thesis2*)
and *SubstTermP-sf* [*iff*]:
Sigma-fm (SubstTermP v i t u) (**is** *?thsf*)
and *SubstTermP-imp-TermP*:
{ *SubstTermP v i t u* } $\vdash$ *TermP i* (**is** *?thterm*)
and *SubstTermP-imp-VarP*:
{ *SubstTermP v i t u* } $\vdash$ *VarP v* (**is** *?thvar*)
⟨*proof*⟩

lemma *SubstTermP-subst* [*simp*]:
(*SubstTermP v i t u*)(*j*::=*w*) = *SubstTermP* (*subst j w v*) (*subst j w i*) (*subst j w t*) (*subst j w u*)
⟨*proof*⟩

lemma *SubstTermP-cong*:
$[\![H \vdash v\ EQ\ v';\ H \vdash i\ EQ\ i';\ H \vdash t\ EQ\ t';\ H \vdash u\ EQ\ u']\!]$
$\Longrightarrow H \vdash$ *SubstTermP v i t u IFF SubstTermP v′ i′ t′ u′*
⟨*proof*⟩

declare *SubstTermP.simps* [*simp del*]

lemma *SubstTerm-imp-subst-dbtm*:
assumes *SubstTerm v* $[\![$*quot-dbtm u*$]\!]e$ *x x′*
shows $\exists$ *t*. *x* = $[\![$*quot-dbtm t*$]\!]e$ $\wedge$
x′ = $[\![$*quot-dbtm* (*subst-dbtm u* (*decode-Var v*) *t*)$]\!]e$
⟨*proof*⟩

corollary *SubstTerm-imp-subst-dbtm′*:
assumes *SubstTerm v y x x′*
obtains *t*::*dbtm* **and** *u*::*dbtm*
where *y* = $[\![$*quot-dbtm u*$]\!]e$
x = $[\![$*quot-dbtm t*$]\!]e$
x′ = $[\![$*quot-dbtm* (*subst-dbtm u* (*decode-Var v*) *t*)$]\!]e$
⟨*proof*⟩

lemma *SubstTerm-subst-dbtm*:
assumes *Term* $[\![$*quot-dbtm u*$]\!]e$
shows *SubstTerm* (*q-Var v*) $[\![$*quot-dbtm u*$]\!]e$ $[\![$*quot-dbtm t*$]\!]e$ $[\![$*quot-dbtm* (*subst-dbtm u v t*)$]\!]e$
⟨*proof*⟩

5.6 Abstraction over formulas

5.6.1 The predicate *AbstAtomicP*

definition *AbstAtomic* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *AbstAtomic v i y y′* ≡
(∃ *t u t′ u′*. *AbstTerm v i t t′* ∧ *AbstTerm v i u u′* ∧

$((y = q\text{-}Eq\ t\ u \wedge y' = q\text{-}Eq\ t'\ u') \vee (y = q\text{-}Mem\ t\ u \wedge y' = q\text{-}Mem\ t'\ u')))$

nominal-function *AbstAtomicP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
  where $[\![atom\ t \sharp (v,i,y,y',t',u,u');\ atom\ t' \sharp (v,i,y,y',u,u');$
    $atom\ u \sharp (v,i,y,y',u');\ atom\ u' \sharp (v,i,y,y')]\!] \Longrightarrow$
   $AbstAtomicP\ v\ i\ y\ y' =$
    $Ex\ t\ (Ex\ u\ (Ex\ t'\ (Ex\ u'$
      $(AbstTermP\ v\ i\ (Var\ t)\ (Var\ t')\ AND\ AbstTermP\ v\ i\ (Var\ u)\ (Var\ u')\ AND$
        $((y\ EQ\ Q\text{-}Eq\ (Var\ t)\ (Var\ u)\ AND\ y'\ EQ\ Q\text{-}Eq\ (Var\ t')\ (Var\ u'))\ OR$
          $(y\ EQ\ Q\text{-}Mem\ (Var\ t)\ (Var\ u)\ AND\ y'\ EQ\ Q\text{-}Mem\ (Var\ t')\ (Var\ u')))))))$
  ⟨*proof*⟩

nominal-termination (*eqvt*)
  ⟨*proof*⟩

lemma
  shows *AbstAtomicP-fresh-iff* [*simp*]:
    $a \sharp AbstAtomicP\ v\ i\ y\ y' \longleftrightarrow a \sharp v \wedge a \sharp i \wedge a \sharp y \wedge a \sharp y'$ (**is** *?thesis1*)
   and *eval-fm-AbstAtomicP* [*simp*]:
     $eval\text{-}fm\ e\ (AbstAtomicP\ v\ i\ y\ y') \longleftrightarrow AbstAtomic\ [\![v]\!]e\ [\![i]\!]e\ [\![y]\!]e\ [\![y']\!]e$ (**is** *?thesis2*)
   and *AbstAtomicP-sf* [*iff*]: $Sigma\text{-}fm\ (AbstAtomicP\ v\ i\ y\ y')$ (**is** *?thsf*)
⟨*proof*⟩

lemma *AbstAtomicP-subst* [*simp*]:
  $(AbstAtomicP\ v\ tm\ y\ y')(i::=w) = AbstAtomicP\ (subst\ i\ w\ v)\ (subst\ i\ w\ tm)\ (subst\ i\ w\ y)\ (subst\ i\ w\ y')$
⟨*proof*⟩

declare *AbstAtomicP.simps* [*simp del*]

5.6.2 The predicate *AbsMakeForm*

definition $AbstMakeForm :: hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$
where $AbstMakeForm\ k\ y\ y'\ i\ u\ u'\ j\ w\ w' \equiv$
  $Ord\ k \wedge$
  $((k = i \wedge k = j \wedge y = q\text{-}Disj\ u\ w \wedge y' = q\text{-}Disj\ u'\ w') \vee$
   $(k = i \wedge y = q\text{-}Neg\ u \wedge y' = q\text{-}Neg\ u') \vee$
   $(succ\ k = i \wedge y = q\text{-}Ex\ u \wedge y' = q\text{-}Ex\ u'))$

definition $SeqAbstForm :: hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$
where $SeqAbstForm\ v\ i\ x\ x'\ s\ k \equiv$
  $BuildSeq3\ (AbstAtomic\ v)\ AbstMakeForm\ s\ k\ i\ x\ x'$

nominal-function *SeqAbstFormP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom l* ♯ (*s*,*k*,*v*,*sli*,*sl*,*sl′*,*m*,*n*,*smi*,*sm*,*sm′*,*sni*,*sn*,*sn′*);
atom sli ♯ (*s*,*v*,*sl*,*sl′*,*m*,*n*,*smi*,*sm*,*sm′*,*sni*,*sn*,*sn′*);
atom sl ♯ (*s*,*v*,*sl′*,*m*,*n*,*smi*,*sm*,*sm′*,*sni*,*sn*,*sn′*);
atom sl′ ♯ (*s*,*v*,*m*,*n*,*smi*,*sm*,*sm′*,*sni*,*sn*,*sn′*);
atom m ♯ (*s*,*n*,*smi*,*sm*,*sm′*,*sni*,*sn*,*sn′*);
atom n ♯ (*s*,*smi*,*sm*,*sm′*,*sni*,*sn*,*sn′*); *atom smi* ♯ (*s*,*sm*,*sm′*,*sni*,*sn*,*sn′*);
atom sm ♯ (*s*,*sm′*,*sni*,*sn*,*sn′*); *atom sm′* ♯ (*s*,*sni*,*sn*,*sn′*);
atom sni ♯ (*s*,*sn*,*sn′*); *atom sn* ♯ (*s*,*sn′*); *atom sn′* ♯ (*s*)⟧ ⟹
SeqAbstFormP v i x x′ s k =
LstSeqP s k (*HPair i* (*HPair x x′*)) *AND*
All2 l (*SUCC k*) (*Ex sli* (*Ex sl* (*Ex sl′* (*HPair* (*Var l*) (*HPair* (*Var sli*) (*HPair* (*Var sl*) (*Var sl′*))) *IN s AND*
(*AbstAtomicP v* (*Var sli*) (*Var sl*) (*Var sl′*) *OR*
OrdP (*Var sli*) *AND*
Ex m (*Ex n* (*Ex smi* (*Ex sm* (*Ex sm′* (*Ex sni* (*Ex sn* (*Ex sn′*
(*Var m IN Var l AND Var n IN Var l AND*
HPair (*Var m*) (*HPair* (*Var smi*) (*HPair* (*Var sm*) (*Var sm′*))) *IN s AND*
HPair (*Var n*) (*HPair* (*Var sni*) (*HPair* (*Var sn*) (*Var sn′*))) *IN s AND*
((*Var sli EQ Var smi AND Var sli EQ Var sni AND*
Var sl EQ Q-Disj (*Var sm*) (*Var sn*) *AND*
Var sl′ EQ Q-Disj (*Var sm′*) (*Var sn′*)) *OR*
(*Var sli EQ Var smi AND*
Var sl EQ Q-Neg (*Var sm*) *AND Var sl′ EQ Q-Neg* (*Var sm′*)) *OR*
(*SUCC* (*Var sli*) *EQ Var smi AND*
Var sl EQ Q-Ex (*Var sm*) *AND Var sl′ EQ Q-Ex* (*Var sm′*))))))))))))))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *SeqAbstFormP-fresh-iff* [*simp*]:
a ♯ *SeqAbstFormP v i x x′ s k* ⟷ *a* ♯ *v* ∧ *a* ♯ *i* ∧ *a* ♯ *x* ∧ *a* ♯ *x′* ∧ *a* ♯ *s* ∧ *a* ♯ *k* (**is** *?thesis1*)
and *eval-fm-SeqAbstFormP* [*simp*]:
eval-fm e (*SeqAbstFormP v i x x′ s k*) ⟷ *SeqAbstForm* ⟦*v*⟧*e* ⟦*i*⟧*e* ⟦*x*⟧*e* ⟦*x′*⟧*e* ⟦*s*⟧*e* ⟦*k*⟧*e* (**is** *?thesis2*)
and *SeqAbstFormP-sf* [*iff*]:
Sigma-fm (*SeqAbstFormP v i x x′ s k*) (**is** *?thsf*)
⟨*proof*⟩

lemma *SeqAbstFormP-subst* [*simp*]:
(*SeqAbstFormP v u x x′ s k*)(*i*::=*t*) =

SeqAbstFormP (*subst i t v*) (*subst i t u*) (*subst i t x*) (*subst i t x′*) (*subst i t s*) (*subst i t k*)
⟨*proof*⟩

declare *SeqAbstFormP.simps* [*simp del*]

5.6.3 Defining the syntax: the main AbstForm predicate

definition *AbstForm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *AbstForm v i x x′* ≡ *is-Var v* ∧ *Ord i* ∧ (∃ *s k*. *SeqAbstForm v i x x′ s k*)

nominal-function *AbstFormP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom s* ♯ (*v*,*i*,*x*,*x′*,*k*);
atom k ♯ (*v*,*i*,*x*,*x′*)⟧ ⟹
AbstFormP v i x x′ = *VarP v AND OrdP i AND Ex s* (*Ex k* (*SeqAbstFormP v i x x′* (*Var s*) (*Var k*)))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *AbstFormP-fresh-iff* [*simp*]:
a ♯ *AbstFormP v i x x′* ⟷ *a* ♯ *v* ∧ *a* ♯ *i* ∧ *a* ♯ *x* ∧ *a* ♯ *x′* (**is** *?thesis1*)
and *eval-fm-AbstFormP* [*simp*]:
eval-fm e (*AbstFormP v i x x′*) ⟷ *AbstForm* ⟦*v*⟧*e* ⟦*i*⟧*e* ⟦*x*⟧*e* ⟦*x′*⟧*e* (**is** *?thesis2*)
and *AbstFormP-sf* [*iff*]:
Sigma-fm (*AbstFormP v i x x′*) (**is** *?thsf*)
⟨*proof*⟩

lemma *AbstFormP-subst* [*simp*]:
(*AbstFormP v i x x′*)(*j*::=*t*) = *AbstFormP* (*subst j t v*) (*subst j t i*) (*subst j t x*) (*subst j t x′*)
⟨*proof*⟩

declare *AbstFormP.simps* [*simp del*]

5.6.4 Correctness: It Coincides with Abstraction over real Formulas

lemma *AbstForm-imp-Ord*: *AbstForm v u x x′* ⟹ *Ord v*
⟨*proof*⟩

lemma *AbstForm-imp-abst-dbfm*:
assumes *AbstForm v i x x′*
shows ∃ *A*. *x* = ⟦*quot-dbfm A*⟧*e* ∧
x′ = ⟦*quot-dbfm* (*abst-dbfm* (*decode-Var v*) (*nat-of-ord i*) *A*)⟧*e*
⟨*proof*⟩

lemma *AbstForm-abst-dbfm*:
AbstForm (*q-Var i*) (*ord-of n*) ⟦*quot-dbfm fm*⟧*e* ⟦*quot-dbfm* (*abst-dbfm i n fm*)⟧*e*
⟨*proof*⟩

5.7 Substitution over formulas

5.7.1 The predicate *SubstAtomicP*

definition *SubstAtomic* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SubstAtomic v tm y y′* ≡
(∃ *t u t′ u′*. *SubstTerm v tm t t′* ∧ *SubstTerm v tm u u′* ∧
((*y* = *q-Eq t u* ∧ *y′* = *q-Eq t′ u′*) ∨ (*y* = *q-Mem t u* ∧ *y′* = *q-Mem t′ u′*)))

nominal-function *SubstAtomicP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom t* ♯ (*v*,*tm*,*y*,*y′*,*t′*,*u*,*u′*);
atom t′ ♯ (*v*,*tm*,*y*,*y′*,*u*,*u′*);
atom u ♯ (*v*,*tm*,*y*,*y′*,*u′*);
atom u′ ♯ (*v*,*tm*,*y*,*y′*)⟧ ⟹
SubstAtomicP v tm y y′ =
Ex t (*Ex u* (*Ex t′* (*Ex u′*
(*SubstTermP v tm* (*Var t*) (*Var t′*) *AND SubstTermP v tm* (*Var u*) (*Var u′*) *AND*
((*y EQ Q-Eq* (*Var t*) (*Var u*) *AND y′ EQ Q-Eq* (*Var t′*) (*Var u′*)) *OR*
(*y EQ Q-Mem* (*Var t*) (*Var u*) *AND y′ EQ Q-Mem* (*Var t′*) (*Var u′*)))))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *SubstAtomicP-fresh-iff* [*simp*]:
a ♯ *SubstAtomicP v tm y y′* ⟷ *a* ♯ *v* ∧ *a* ♯ *tm* ∧ *a* ♯ *y* ∧ *a* ♯ *y′* (**is** *?thesis1*)
and *eval-fm-SubstAtomicP* [*simp*]:
eval-fm e (*SubstAtomicP v tm y y′*) ⟷ *SubstAtomic* ⟦*v*⟧*e* ⟦*tm*⟧*e* ⟦*y*⟧*e* ⟦*y′*⟧*e* (**is** *?thesis2*)
and *SubstAtomicP-sf* [*iff*]: *Sigma-fm* (*SubstAtomicP v tm y y′*) (**is** *?thsf*)
⟨*proof*⟩

lemma *SubstAtomicP-subst* [*simp*]:
(*SubstAtomicP v tm y y′*)(*i*::=*w*) = *SubstAtomicP* (*subst i w v*) (*subst i w tm*) (*subst i w y*) (*subst i w y′*)
⟨*proof*⟩

lemma *SubstAtomicP-cong*:

$[\![H \vdash v\ EQ\ v';\ H \vdash tm\ EQ\ tm';\ H \vdash x\ EQ\ x';\ H \vdash y\ EQ\ y']\!]$
$\Longrightarrow H \vdash SubstAtomicP\ v\ tm\ x\ y\ IFF\ SubstAtomicP\ v'\ tm'\ x'\ y'$
⟨*proof*⟩

5.7.2 The predicate *SubstMakeForm*

definition *SubstMakeForm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SubstMakeForm y y' u u' w w'* ≡
((*y* = *q-Disj u w* ∧ *y'* = *q-Disj u' w'*) ∨
(*y* = *q-Neg u* ∧ *y'* = *q-Neg u'*) ∨
(*y* = *q-Ex u* ∧ *y'* = *q-Ex u'*))

definition *SeqSubstForm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SeqSubstForm v u x x' s k* ≡ *BuildSeq2* (*SubstAtomic v u*) *SubstMakeForm s k x x'*

nominal-function *SeqSubstFormP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where [\![*atom l* ♯ (*s*,*k*,*v*,*u*,*sl*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*);
atom sl ♯ (*s*,*v*,*u*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*);
atom sl' ♯ (*s*,*v*,*u*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*);
atom m ♯ (*s*,*n*,*sm*,*sm'*,*sn*,*sn'*); *atom n* ♯ (*s*,*sm*,*sm'*,*sn*,*sn'*);
atom sm ♯ (*s*,*sm'*,*sn*,*sn'*); *atom sm'* ♯ (*s*,*sn*,*sn'*);
atom sn ♯ (*s*,*sn'*); *atom sn'* ♯ *s*]\!] ⟹
SeqSubstFormP v u x x' s k =
LstSeqP s k (*HPair x x'*) *AND*
All2 l (*SUCC k*) (*Ex sl* (*Ex sl'* (*HPair* (*Var l*) (*HPair* (*Var sl*) (*Var sl'*)) *IN s AND*
(*SubstAtomicP v u* (*Var sl*) (*Var sl'*) *OR*
Ex m (*Ex n* (*Ex sm* (*Ex sm'* (*Ex sn* (*Ex sn'* (*Var m IN Var l AND Var n IN Var l AND*
HPair (*Var m*) (*HPair* (*Var sm*) (*Var sm'*)) *IN s AND*
HPair (*Var n*) (*HPair* (*Var sn*) (*Var sn'*)) *IN s AND*
((*Var sl EQ Q-Disj* (*Var sm*) (*Var sn*) *AND*
Var sl' EQ Q-Disj (*Var sm'*) (*Var sn'*)) *OR*
(*Var sl EQ Q-Neg* (*Var sm*) *AND Var sl' EQ Q-Neg* (*Var sm'*)) *OR*
(*Var sl EQ Q-Ex* (*Var sm*) *AND Var sl' EQ Q-Ex* (*Var sm'*))))))))))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *SeqSubstFormP-fresh-iff* [*simp*]:
a ♯ *SeqSubstFormP v u x x' s k* ⟷ *a* ♯ *v* ∧ *a* ♯ *u* ∧ *a* ♯ *x* ∧ *a* ♯ *x'* ∧ *a* ♯ *s* ∧ *a* ♯ *k* (**is** *?thesis1*)
and *eval-fm-SeqSubstFormP* [*simp*]:
eval-fm e (*SeqSubstFormP v u x x' s k*) ⟷

SeqSubstForm ⟦*v*⟧*e* ⟦*u*⟧*e* ⟦*x*⟧*e* ⟦*x'*⟧*e* ⟦*s*⟧*e* ⟦*k*⟧*e* (**is** *?thesis2*)
and *SeqSubstFormP-sf* [*iff*]:
Sigma-fm (*SeqSubstFormP v u x x' s k*) (**is** *?thsf*)
and *SeqSubstFormP-imp-OrdP*:
{ *SeqSubstFormP v u x x' s k* } ⊢ *OrdP k* (**is** *?thOrd*)
and *SeqSubstFormP-imp-LstSeqP*:
{ *SeqSubstFormP v u x x' s k* } ⊢ *LstSeqP s k* (*HPair x x'*) (**is** *?thLstSeq*)
⟨*proof*⟩

lemma *SeqSubstFormP-subst* [*simp*]:
(*SeqSubstFormP v u x x' s k*)(*i*::=*t*) =
SeqSubstFormP (*subst i t v*) (*subst i t u*) (*subst i t x*) (*subst i t x'*) (*subst i t s*) (*subst i t k*)
⟨*proof*⟩

lemma *SeqSubstFormP-cong*:
⟦*H* ⊢ *t EQ t'*; *H* ⊢ *u EQ u'*; *H* ⊢ *s EQ s'*; *H* ⊢ *k EQ k'*⟧
⟹ *H* ⊢ *SeqSubstFormP v i t u s k IFF SeqSubstFormP v i t' u' s' k'*
⟨*proof*⟩

declare *SeqSubstFormP.simps* [*simp del*]

5.7.3 Defining the syntax: the main SubstForm predicate

definition *SubstForm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SubstForm v u x x'* ≡ *is-Var v* ∧ *Term u* ∧ (∃ *s k*. *SeqSubstForm v u x x' s k*)

nominal-function *SubstFormP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom s* ♯ (*v*,*i*,*x*,*x'*,*k*); *atom k* ♯ (*v*,*i*,*x*,*x'*)⟧ ⟹
SubstFormP v i x x' =
VarP v AND TermP i AND Ex s (*Ex k* (*SeqSubstFormP v i x x'* (*Var s*) (*Var k*)))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *SubstFormP-fresh-iff* [*simp*]:
a ♯ *SubstFormP v i x x'* ⟷ *a* ♯ *v* ∧ *a* ♯ *i* ∧ *a* ♯ *x* ∧ *a* ♯ *x'* (**is** *?thesis1*)
and *eval-fm-SubstFormP* [*simp*]:
eval-fm e (*SubstFormP v i x x'*) ⟷ *SubstForm* ⟦*v*⟧*e* ⟦*i*⟧*e* ⟦*x*⟧*e* ⟦*x'*⟧*e* (**is** *?thesis2*)
and *SubstFormP-sf* [*iff*]:
Sigma-fm (*SubstFormP v i x x'*) (**is** *?thsf*)
⟨*proof*⟩

lemma *SubstFormP-subst* [*simp*]:

$(SubstFormP\ v\ i\ x\ x')(j::=t) = SubstFormP\ (subst\ j\ t\ v)\ (subst\ j\ t\ i)\ (subst\ j\ t\ x)\ (subst\ j\ t\ x')$
⟨*proof*⟩

lemma *SubstFormP-cong*:
$[\![H \vdash v\ EQ\ v';\ H \vdash i\ EQ\ i';\ H \vdash t\ EQ\ t';\ H \vdash u\ EQ\ u']\!]$
$\Longrightarrow H \vdash SubstFormP\ v\ i\ t\ u\ IFF\ SubstFormP\ v'\ i'\ t'\ u'$
⟨*proof*⟩

lemma *ground-SubstFormP* [*simp*]: $ground\text{-}fm\ (SubstFormP\ v\ y\ x\ x') \longleftrightarrow ground\ v \wedge ground\ y \wedge ground\ x \wedge ground\ x'$
⟨*proof*⟩

declare *SubstFormP.simps* [*simp del*]

5.7.4 Correctness of substitution over formulas

lemma *SubstForm-imp-subst-dbfm-lemma*:
assumes $SubstForm\ v\ [\![quot\text{-}dbtm\ u]\!]e\ x\ x'$
shows $\exists A.\ x = [\![quot\text{-}dbfm\ A]\!]e \wedge x' = [\![quot\text{-}dbfm\ (subst\text{-}dbfm\ u\ (decode\text{-}Var\ v)\ A)]\!]e$
⟨*proof*⟩

lemma *SubstForm-imp-subst-dbfm*:
assumes $SubstForm\ v\ u\ x\ x'$
obtains $t\ A$ **where** $u = [\![quot\text{-}dbtm\ t]\!]e$ $x = [\![quot\text{-}dbfm\ A]\!]e$ $x' = [\![quot\text{-}dbfm\ (subst\text{-}dbfm\ t\ (decode\text{-}Var\ v)\ A)]\!]e$
⟨*proof*⟩

lemma *SubstForm-subst-dbfm*:
assumes *u*: *wf-dbtm u*
shows $SubstForm\ (q\text{-}Var\ i)\ [\![quot\text{-}dbtm\ u]\!]e\ [\![quot\text{-}dbfm\ A]\!]e\ [\![quot\text{-}dbfm\ (subst\text{-}dbfm\ u\ i\ A)]\!]e$
⟨*proof*⟩

corollary *SubstForm-subst-dbfm-eq*:
$[\![v = q\text{-}Var\ i;\ Term\ ux;\ ux = [\![quot\text{-}dbtm\ u]\!]e;\ A' = subst\text{-}dbfm\ u\ i\ A]\!]$
$\Longrightarrow SubstForm\ v\ ux\ [\![quot\text{-}dbfm\ A]\!]e\ [\![quot\text{-}dbfm\ A']\!]e$
⟨*proof*⟩

5.8 The predicate *AtomicP*

definition $Atomic :: hf \Rightarrow bool$
where $Atomic\ y \equiv \exists t\ u.\ Term\ t \wedge Term\ u \wedge (y = q\text{-}Eq\ t\ u \vee y = q\text{-}Mem\ t\ u)$

nominal-function $AtomicP :: tm \Rightarrow fm$
where $[\![atom\ t \sharp (u,y);\ atom\ u \sharp y]\!] \Longrightarrow$
$AtomicP\ y = Ex\ t\ (Ex\ u\ (TermP\ (Var\ t)\ AND\ TermP\ (Var\ u)\ AND$

(*y EQ Q-Eq* (*Var t*) (*Var u*) *OR*
y EQ Q-Mem (*Var t*) (*Var u*))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *AtomicP-fresh-iff* [*simp*]: $a \sharp$ *AtomicP y* $\longleftrightarrow$ $a \sharp y$ (**is** *?thesis1*)
and *eval-fm-AtomicP* [*simp*]: *eval-fm e* (*AtomicP y*) $\longleftrightarrow$ *Atomic*⟦*y*⟧*e* (**is** *?thesis2*)
and *AtomicP-sf* [*iff*]: *Sigma-fm* (*AtomicP y*) (**is** *?thsf*)
⟨*proof*⟩

5.9 The predicate *MakeForm*

definition *MakeForm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *MakeForm y u w* ≡
y = *q-Disj u w* ∨ *y* = *q-Neg u* ∨
(∃ *v u'*. *AbstForm v 0 u u'* ∧ *y* = *q-Ex u'*)

nominal-function *MakeFormP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom v* $\sharp$ (*y*,*u*,*w*,*au*); *atom au* $\sharp$ (*y*,*u*,*w*)⟧ ⟹
MakeFormP y u w =
y EQ Q-Disj u w OR y EQ Q-Neg u OR
Ex v (*Ex au* (*AbstFormP* (*Var v*) *Zero u* (*Var au*) *AND y EQ Q-Ex* (*Var au*)))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *MakeFormP-fresh-iff* [*simp*]:
$a \sharp$ *MakeFormP y u w* $\longleftrightarrow$ $a \sharp y \wedge a \sharp u \wedge a \sharp w$ (**is** *?thesis1*)
and *eval-fm-MakeFormP* [*simp*]:
eval-fm e (*MakeFormP y u w*) $\longleftrightarrow$ *MakeForm* ⟦*y*⟧*e* ⟦*u*⟧*e* ⟦*w*⟧*e* (**is** *?thesis2*)
and *MakeFormP-sf* [*iff*]:
Sigma-fm (*MakeFormP y u w*) (**is** *?thsf*)
⟨*proof*⟩

declare *MakeFormP.simps* [*simp del*]

5.10 The predicate *SeqFormP*

definition *SeqForm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SeqForm s k y* ≡ *BuildSeq Atomic MakeForm s k y*

nominal-function *SeqFormP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
  where ⟦*atom l* ♯ (*s*,*k*,*t*,*sl*,*m*,*n*,*sm*,*sn*); *atom sl* ♯ (*s*,*k*,*t*,*m*,*n*,*sm*,*sn*);
        atom m ♯ (*s*,*k*,*t*,*n*,*sm*,*sn*); *atom n* ♯ (*s*,*k*,*t*,*sm*,*sn*);
        atom sm ♯ (*s*,*k*,*t*,*sn*); *atom sn* ♯ (*s*,*k*,*t*)⟧ ⟹
    SeqFormP s k t =
      LstSeqP s k t AND
      All2 n (*SUCC k*) (*Ex sn* (*HPair* (*Var n*) (*Var sn*) *IN s AND* (*AtomicP* (*Var sn*) *OR*
            Ex m (*Ex l* (*Ex sm* (*Ex sl* (*Var m IN Var n AND Var l IN Var n AND*
                HPair (*Var m*) (*Var sm*) *IN s AND HPair* (*Var l*) (*Var sl*) *IN s AND*
                MakeFormP (*Var sn*) (*Var sm*) (*Var sl*))))))))
  ⟨*proof*⟩

nominal-termination (*eqvt*)
  ⟨*proof*⟩

lemma
  shows *SeqFormP-fresh-iff* [*simp*]:
      a ♯ *SeqFormP s k t* ⟷ *a* ♯ *s* ∧ *a* ♯ *k* ∧ *a* ♯ *t* (**is** *?thesis1*)
    and *eval-fm-SeqFormP* [*simp*]:
      eval-fm e (*SeqFormP s k t*) ⟷ *SeqForm* ⟦*s*⟧*e* ⟦*k*⟧*e* ⟦*t*⟧*e* (**is** *?thesis2*)
    and *SeqFormP-sf* [*iff*]: *Sigma-fm* (*SeqFormP s k t*) (**is** *?thsf*)
⟨*proof*⟩

lemma *SeqFormP-subst* [*simp*]:
    (*SeqFormP s k t*)(*j*::=*w*) = *SeqFormP* (*subst j w s*) (*subst j w k*) (*subst j w t*)
⟨*proof*⟩

5.11 The predicate *FormP*

5.11.1 Definition

definition *Form* :: *hf* ⇒ *bool*
  where *Form y* ≡ (∃ *s k*. *SeqForm s k y*)

nominal-function *FormP* :: *tm* ⇒ *fm*
  where ⟦*atom k* ♯ (*s*,*y*); *atom s* ♯ *y*⟧ ⟹
    FormP y = *Ex k* (*Ex s* (*SeqFormP* (*Var s*) (*Var k*) *y*))
  ⟨*proof*⟩

nominal-termination (*eqvt*)
  ⟨*proof*⟩

lemma
  shows *FormP-fresh-iff* [*simp*]: *a* ♯ *FormP y* ⟷ *a* ♯ *y* (**is** *?thesis1*)
   and *eval-fm-FormP* [*simp*]: *eval-fm e* (*FormP y*) ⟷ *Form* ⟦*y*⟧*e* (**is** *?thesis2*)
   and *FormP-sf* [*iff*]: *Sigma-fm* (*FormP y*) (**is** *?thsf*)
⟨*proof*⟩

lemma *FormP-subst* [*simp*]: (*FormP y*)(*j*::=*w*) = *FormP* (*subst j w y*)
⟨*proof*⟩

5.11.2 Correctness: It Corresponds to Quotations of Real Formulas

lemma *AbstForm-trans-fm*:
AbstForm (*q-Var i*) *0* ⟦«*A*»⟧*e* ⟦*quot-dbfm* (*trans-fm* [*i*] *A*)⟧*e*
⟨*proof*⟩

corollary *AbstForm-trans-fm-eq*:
⟦*x* = ⟦«*A*»⟧ *e*; *x'* = ⟦*quot-dbfm* (*trans-fm* [*i*] *A*)⟧*e*⟧ ⟹ *AbstForm* (*q-Var i*) *0 x x'*
⟨*proof*⟩

lemma *wf-Form-quot-dbfm* [*simp*]:
assumes *wf-dbfm A* **shows** *Form* ⟦*quot-dbfm A*⟧*e*
⟨*proof*⟩

lemma *Form-quot-fm* [*iff*]: **fixes** *A* :: *fm* **shows** *Form* ⟦«*A*»⟧*e*
⟨*proof*⟩

lemma *Atomic-Form-is-wf-dbfm*: *Atomic x* ⟹ ∃ *A*. *wf-dbfm A* ∧ *x* = ⟦*quot-dbfm A*⟧*e*
⟨*proof*⟩

lemma *SeqForm-imp-wf-dbfm*:
assumes *SeqForm s k x*
shows ∃ *A*. *wf-dbfm A* ∧ *x* = ⟦*quot-dbfm A*⟧*e*
⟨*proof*⟩

lemma *Form-imp-wf-dbfm*:
assumes *Form x* **obtains** *A* **where** *wf-dbfm A x* = ⟦*quot-dbfm A*⟧*e*
⟨*proof*⟩

lemma *Form-imp-is-fm*: **assumes** *Form x* **obtains** *A*::*fm* **where** *x* = ⟦«*A*»⟧ *e*
⟨*proof*⟩

lemma *SubstForm-imp-subst-fm*:
assumes *SubstForm v* ⟦«*u*»⟧*e x x' Form x*
obtains *A*::*fm* **where** *x* = ⟦«*A*»⟧ *e x'* = ⟦«*A*(*decode-Var v*::=*u*)»⟧ *e*
⟨*proof*⟩

lemma *SubstForm-unique*:
assumes *is-Var v* **and** *Term y* **and** *Form x*
shows *SubstForm v y x x'* ⟷
(∃ *t*::*tm*. *y* = ⟦«*t*»⟧*e* ∧ (∃ *A*::*fm*. *x* = ⟦«*A*»⟧*e* ∧ *x'* = ⟦«*A*(*decode-Var v*::=*t*)»⟧*e*))

⟨*proof*⟩

lemma *SubstForm-quot-unique*: *SubstForm* (*q-Var i*) ⟦«*t*»⟧*e* ⟦«*A*»⟧*e x′* ⟷ *x′* = ⟦«*A*(*i*::=*t*)»⟧ *e*
⟨*proof*⟩

lemma *SubstForm-quot*: *SubstForm* ⟦«*Var i*»⟧*e* ⟦«*t*»⟧*e* ⟦«*A*»⟧*e* ⟦«*A*(*i*::=*t*)»⟧*e*
⟨*proof*⟩

5.11.3 The predicate *VarNonOccFormP* (Derived from *SubstFormP*)

definition *VarNonOccForm* :: *hf* ⇒ *hf* ⇒ *bool*
where *VarNonOccForm v x* ≡ *Form x* ∧ *SubstForm v 0 x x*

nominal-function *VarNonOccFormP* :: *tm* ⇒ *tm* ⇒ *fm*
where *VarNonOccFormP v x* = *FormP x AND SubstFormP v Zero x x*
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *VarNonOccFormP-fresh-iff* [*simp*]: *a* ♯ *VarNonOccFormP v y* ⟷ *a* ♯ *v* ∧ *a* ♯ *y* (**is** *?thesis1*)
and *eval-fm-VarNonOccFormP* [*simp*]:
eval-fm e (*VarNonOccFormP v y*) ⟷ *VarNonOccForm* ⟦*v*⟧*e* ⟦*y*⟧*e* (**is** *?thesis2*)
and *VarNonOccFormP-sf* [*iff*]: *Sigma-fm* (*VarNonOccFormP v y*) (**is** *?thsf*)
⟨*proof*⟩

5.11.4 Correctness for Real Terms and Formulas

lemma *VarNonOccForm-imp-dbfm-fresh*:
assumes *VarNonOccForm v x*
shows ∃ *A*. *wf-dbfm A* ∧ *x* = ⟦*quot-dbfm A*⟧*e* ∧ *atom* (*decode-Var v*) ♯ *A*
⟨*proof*⟩

corollary *VarNonOccForm-imp-fresh*:
assumes *VarNonOccForm v x* **obtains** *A*::*fm* **where** *x* = ⟦«*A*»⟧*e atom* (*decode-Var v*) ♯ *A*
⟨*proof*⟩

lemma *VarNonOccForm-dbfm*:
wf-dbfm A ⟹ *atom i* ♯ *A* ⟹ *VarNonOccForm* (*q-Var i*) ⟦*quot-dbfm A*⟧*e*
⟨*proof*⟩

corollary *fresh-imp-VarNonOccForm*:
fixes *A*::*fm* **shows** *atom i* ♯ *A* ⟹ *VarNonOccForm* (*q-Var i*) ⟦«*A*»⟧*e*

⟨*proof*⟩

declare *VarNonOccFormP.simps* [*simp del*]

end

Kapitel 6

Formalizing Provability

theory *Pf-Predicates*
imports *Coding-Predicates*
begin

6.1 Section 4 Predicates (Leading up to Pf)

6.1.1 The predicate *SentP*, for the Sentiential (Boolean) Axioms

definition *Sent-axioms* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool* **where**
Sent-axioms x y z w ≡
x = *q-Imp y y* ∨
x = *q-Imp y* (*q-Disj y z*) ∨
x = *q-Imp* (*q-Disj y y*) *y* ∨
x = *q-Imp* (*q-Disj y* (*q-Disj z w*)) (*q-Disj* (*q-Disj y z*) *w*) ∨
x = *q-Imp* (*q-Disj y z*) (*q-Imp* (*q-Disj* (*q-Neg y*) *w*) (*q-Disj z w*))

definition *Sent* :: *hf set* **where**
Sent ≡ {*x*. ∃ *y z w*. *Form y* ∧ *Form z* ∧ *Form w* ∧ *Sent-axioms x y z w*}

nominal-function *SentP* :: *tm* ⇒ *fm*
where ⟦*atom y* ♯ (*z*,*w*,*x*); *atom z* ♯ (*w*,*x*); *atom w* ♯ *x*⟧ ⟹
SentP x = *Ex y* (*Ex z* (*Ex w* (*FormP* (*Var y*) *AND FormP* (*Var z*) *AND FormP* (*Var w*) *AND*
((*x EQ Q-Imp* (*Var y*) (*Var y*)) *OR*
(*x EQ Q-Imp* (*Var y*) (*Q-Disj* (*Var y*) (*Var z*)) *OR*
(*x EQ Q-Imp* (*Q-Disj* (*Var y*) (*Var y*)) (*Var y*)) *OR*
(*x EQ Q-Imp* (*Q-Disj* (*Var y*) (*Q-Disj* (*Var z*) (*Var w*)))
(*Q-Disj* (*Q-Disj* (*Var y*) (*Var z*)) (*Var w*))) *OR*
(*x EQ Q-Imp* (*Q-Disj* (*Var y*) (*Var z*))
(*Q-Imp* (*Q-Disj* (*Q-Neg* (*Var y*)) (*Var w*)) (*Q-Disj* (*Var z*) (*Var w*)))))))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *SentP-fresh-iff* [*simp*]: $a \sharp SentP\ x \longleftrightarrow a \sharp x$ (**is** *?thesis1*)
and *eval-fm-SentP* [*simp*]: $eval\text{-}fm\ e\ (SentP\ x) \longleftrightarrow [\![x]\!]e \in Sent$ (**is** *?thesis2*)
and *SentP-sf* [*iff*]: $Sigma\text{-}fm\ (SentP\ x)$ (**is** *?thsf*)
⟨*proof*⟩

6.1.2 The predicate *Equality-axP*, for the Equality Axioms

definition *Equality-ax* :: *hf set* **where**
$Equality\text{-}ax \equiv \{ [\![«refl\text{-}ax»]\!]e0, [\![«eq\text{-}cong\text{-}ax»]\!]e0, [\![«mem\text{-}cong\text{-}ax»]\!]e0, [\![«eats\text{-}cong\text{-}ax»]\!]e0 \}$

function *Equality-axP* :: *tm* ⇒ *fm*
where *Equality-axP x* =
x EQ «refl-ax» OR x EQ «eq-cong-ax» OR x EQ «mem-cong-ax» OR x EQ «eats-cong-ax»
⟨*proof*⟩

termination
⟨*proof*⟩

lemma *eval-fm-Equality-axP* [*simp*]: $eval\text{-}fm\ e\ (Equality\text{-}axP\ x) \longleftrightarrow [\![x]\!]e \in Equality\text{-}ax$
⟨*proof*⟩

6.1.3 The predicate *HF-axP*, for the HF Axioms

definition *HF-ax* :: *hf set* **where**
$HF\text{-}ax \equiv \{[\![«HF1»]\!]e0, [\![«HF2»]\!]e0\}$

function *HF-axP* :: *tm* ⇒ *fm*
where *HF-axP x = x EQ «HF1» OR x EQ «HF2»*
⟨*proof*⟩

termination
⟨*proof*⟩

lemma *eval-fm-HF-axP* [*simp*]: $eval\text{-}fm\ e\ (HF\text{-}axP\ x) \longleftrightarrow [\![x]\!]e \in HF\text{-}ax$
⟨*proof*⟩

lemma *HF-axP-sf* [*iff*]: $Sigma\text{-}fm\ (HF\text{-}axP\ t)$
⟨*proof*⟩

6.1.4 The specialisation axioms

inductive-set *Special-ax* :: *hf set* **where**
I: $[\![AbstForm\ v\ 0\ x\ ax; SubstForm\ v\ y\ x\ sx; Form\ x; is\text{-}Var\ v; Term\ y]\!]$

$\Longrightarrow$ *q-Imp sx* (*q-Ex ax*) $\in$ *Special-ax*

Defining the syntax

nominal-function *Special-axP* :: *tm* $\Rightarrow$ *fm* **where**
⟦*atom v* ♯ (*p*,*sx*,*y*,*ax*,*x*); *atom x* ♯ (*p*,*sx*,*y*,*ax*);
atom ax ♯ (*p*,*sx*,*y*); *atom y* ♯ (*p*,*sx*); *atom sx* ♯ *p*⟧ $\Longrightarrow$
Special-axP p = *Ex v* (*Ex x* (*Ex ax* (*Ex y* (*Ex sx*
(*FormP* (*Var x*) *AND VarP* (*Var v*) *AND TermP* (*Var y*) *AND*
AbstFormP (*Var v*) *Zero* (*Var x*) (*Var ax*) *AND*
SubstFormP (*Var v*) (*Var y*) (*Var x*) (*Var sx*) *AND*
p EQ Q-Imp (*Var sx*) (*Q-Ex* (*Var ax*)))))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *Special-axP-fresh-iff* [*simp*]: *a* ♯ *Special-axP p* $\longleftrightarrow$ *a* ♯ *p* (**is** *?thesis1*)
and *eval-fm-Special-axP* [*simp*]: *eval-fm e* (*Special-axP p*) $\longleftrightarrow$ ⟦*p*⟧*e* $\in$ *Special-ax*
(**is** *?thesis2*)
and *Special-axP-sf* [*iff*]: *Sigma-fm* (*Special-axP p*) (**is** *?thesis3*)
⟨*proof*⟩

Correctness (or, correspondence)

lemma *Special-ax-imp-special-axioms*:
assumes *x* $\in$ *Special-ax* **shows** $\exists$ *A*. *x* = ⟦«*A*»⟧*e* $\wedge$ *A* $\in$ *special-axioms*
⟨*proof*⟩

lemma *special-axioms-into-Special-ax*: *A* $\in$ *special-axioms* $\Longrightarrow$ ⟦«*A*»⟧*e* $\in$ *Special-ax*
⟨*proof*⟩

We have precisely captured the codes of the specialisation axioms.

corollary *Special-ax-eq-special-axioms*: *Special-ax* = ($\bigcup$ *A* $\in$ *special-axioms*. { ⟦«*A*»⟧*e* })
⟨*proof*⟩

6.1.5 The induction axioms

inductive-set *Induction-ax* :: *hf set* **where**
I: ⟦*SubstForm v 0 x x0*;
SubstForm v w x xw;
SubstForm v (*q-Eats v w*) *x xevw*;
AbstForm w 0 (*q-Imp x* (*q-Imp xw xevw*)) *allw*;
AbstForm v 0 (*q-All allw*) *allvw*;
AbstForm v 0 x ax;
v $\neq$ *w*; *VarNonOccForm w x*⟧
$\Longrightarrow$ *q-Imp x0* (*q-Imp* (*q-All allvw*) (*q-All ax*)) $\in$ *Induction-ax*

Defining the syntax

nominal-function *Induction-axP* :: *tm* ⇒ *fm* **where**
⟦*atom ax* ♯ (*p*,*v*,*w*,*x*,*x0*,*xw*,*xevw*,*allw*,*allvw*);
atom allvw ♯ (*p*,*v*,*w*,*x*,*x0*,*xw*,*xevw*,*allw*); *atom allw* ♯ (*p*,*v*,*w*,*x*,*x0*,*xw*,*xevw*);
atom xevw ♯ (*p*,*v*,*w*,*x*,*x0*,*xw*); *atom xw* ♯ (*p*,*v*,*w*,*x*,*x0*);
atom x0 ♯ (*p*,*v*,*w*,*x*); *atom x* ♯ (*p*,*v*,*w*);
atom w ♯ (*p*,*v*); *atom v* ♯ *p*⟧ ⟹
Induction-axP p = *Ex v* (*Ex w* (*Ex x* (*Ex x0* (*Ex xw* (*Ex xevw* (*Ex allw* (*Ex allvw* (*Ex ax*
((*Var v NEQ Var w*) *AND VarNonOccFormP* (*Var w*) (*Var x*) *AND*
SubstFormP (*Var v*) *Zero* (*Var x*) (*Var x0*) *AND*
SubstFormP (*Var v*) (*Var w*) (*Var x*) (*Var xw*) *AND*
SubstFormP (*Var v*) (*Q-Eats* (*Var v*) (*Var w*)) (*Var x*) (*Var xevw*) *AND*
AbstFormP (*Var w*) *Zero* (*Q-Imp* (*Var x*) (*Q-Imp* (*Var xw*) (*Var xevw*))) (*Var allw*) *AND*
AbstFormP (*Var v*) *Zero* (*Q-All* (*Var allw*)) (*Var allvw*) *AND*
AbstFormP (*Var v*) *Zero* (*Var x*) (*Var ax*) *AND*
p EQ Q-Imp (*Var x0*) (*Q-Imp* (*Q-All* (*Var allvw*)) (*Q-All* (*Var ax*))))))))))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *Induction-axP-fresh-iff* [*simp*]: *a* ♯ *Induction-axP p* ⟷ *a* ♯ *p* (**is** *?thesis1*)
and *eval-fm-Induction-axP* [*simp*]:
eval-fm e (*Induction-axP p*) ⟷ ⟦*p*⟧*e* ∈ *Induction-ax* (**is** *?thesis2*)
and *Induction-axP-sf* [*iff*]: *Sigma-fm* (*Induction-axP p*) (**is** *?thesis3*)
⟨*proof*⟩

Correctness (or, correspondence)

lemma *Induction-ax-imp-induction-axioms*:
assumes *x* ∈ *Induction-ax* **shows** ∃ *A*. *x* = ⟦«*A*»⟧*e* ∧ *A* ∈ *induction-axioms*
⟨*proof*⟩

lemma *induction-axioms-into-Induction-ax*:
A ∈ *induction-axioms* ⟹ ⟦«*A*»⟧*e* ∈ *Induction-ax*
⟨*proof*⟩

We have captured the codes of the induction axioms.

corollary *Induction-ax-eq-induction-axioms*:
Induction-ax = (⋃ *A* ∈ *induction-axioms*. {⟦«*A*»⟧*e*})
⟨*proof*⟩

6.1.6 The predicate *AxiomP*, for any Axioms

definition *Extra-ax* :: *hf set* **where**
Extra-ax ≡ {⟦«*extra-axiom*»⟧*e0*}

definition *Axiom* :: *hf set* **where**
Axiom ≡ *Extra-ax* ∪ *Sent* ∪ *Equality-ax* ∪ *HF-ax* ∪ *Special-ax* ∪ *Induction-ax*

definition *AxiomP* :: *tm* ⇒ *fm*
where *AxiomP x* ≡ *x EQ* «*extra-axiom*» *OR SentP x OR Equality-axP x OR*
HF-axP x OR Special-axP x OR Induction-axP x

lemma *AxiomP-eqvt* [*eqvt*]: (*p* ∙ *AxiomP x*) = *AxiomP* (*p* ∙ *x*)
⟨*proof*⟩

lemma *AxiomP-fresh-iff* [*simp*]: *a* ♯ *AxiomP x* ⟷ *a* ♯ *x*
⟨*proof*⟩

lemma *eval-fm-AxiomP* [*simp*]: *eval-fm e* (*AxiomP x*) ⟷ ⟦*x*⟧*e* ∈ *Axiom*
⟨*proof*⟩

lemma *AxiomP-sf* [*iff*]: *Sigma-fm* (*AxiomP t*)
⟨*proof*⟩

6.1.7 The predicate *ModPonP*, for the inference rule Modus Ponens

definition *ModPon* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool* **where**
ModPon x y z ≡ (*y* = *q-Imp x z*)

definition *ModPonP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where *ModPonP x y z* = (*y EQ Q-Imp x z*)

lemma *ModPonP-eqvt* [*eqvt*]: (*p* ∙ *ModPonP x y z*) = *ModPonP* (*p* ∙ *x*) (*p* ∙ *y*) (*p* ∙ *z*)
⟨*proof*⟩

lemma *ModPonP-fresh-iff* [*simp*]: *a* ♯ *ModPonP x y z* ⟷ *a* ♯ *x* ∧ *a* ♯ *y* ∧ *a* ♯ *z*
⟨*proof*⟩

lemma *eval-fm-ModPonP* [*simp*]: *eval-fm e* (*ModPonP x y z*) ⟷ *ModPon* ⟦*x*⟧*e* ⟦*y*⟧*e* ⟦*z*⟧*e*
⟨*proof*⟩

lemma *ModPonP-sf* [*iff*]: *Sigma-fm* (*ModPonP t u v*)
⟨*proof*⟩

lemma *ModPonP-subst* [*simp*]:
(*ModPonP t u v*)(*i*::=*w*) = *ModPonP* (*subst i w t*) (*subst i w u*) (*subst i w v*)
⟨*proof*⟩

6.1.8 The predicate *ExistsP*, for the existential rule

Definition

definition *Exists* :: *hf* ⇒ *hf* ⇒ *bool* **where**
Exists p q ≡ (∃ *x x′ y v*. *Form x* ∧ *VarNonOccForm v y* ∧ *AbstForm v 0 x x′* ∧
p = *q-Imp x y* ∧ *q* = *q-Imp* (*q-Ex x′*) *y*)

nominal-function *ExistsP* :: *tm* ⇒ *tm* ⇒ *fm* **where**
⟦*atom x* ♯ (*p*,*q*,*v*,*y*,*x′*); *atom x′* ♯ (*p*,*q*,*v*,*y*);
atom y ♯ (*p*,*q*,*v*); *atom v* ♯ (*p*,*q*)⟧ ⟹
ExistsP p q = *Ex x* (*Ex x′* (*Ex y* (*Ex v* (*FormP* (*Var x*) *AND*
VarNonOccFormP (*Var v*) (*Var y*) *AND*
AbstFormP (*Var v*) *Zero* (*Var x*) (*Var x′*) *AND*
p EQ Q-Imp (*Var x*) (*Var y*) *AND*
q EQ Q-Imp (*Q-Ex* (*Var x′*)) (*Var y*)))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *ExistsP-fresh-iff* [*simp*]: *a* ♯ *ExistsP p q* ⟷ *a* ♯ *p* ∧ *a* ♯ *q* (**is** *?thesis1*)
and *eval-fm-ExistsP* [*simp*]: *eval-fm e* (*ExistsP p q*) ⟷ *Exists* ⟦*p*⟧*e* ⟦*q*⟧*e* (**is** *?thesis2*)
and *ExistsP-sf* [*iff*]: *Sigma-fm* (*ExistsP p q*) (**is** *?thesis3*)
⟨*proof*⟩

lemma *ExistsP-subst* [*simp*]: (*ExistsP p q*)(*j*::=*w*) = *ExistsP* (*subst j w p*) (*subst j w q*)
⟨*proof*⟩

Correctness

lemma *Exists-imp-exists*:
assumes *Exists p q*
shows ∃ *A B i*. *p* = ⟦«*A IMP B*»⟧*e* ∧ *q* = ⟦«(*Ex i A*) *IMP B*»⟧*e* ∧ *atom i* ♯ *B*
⟨*proof*⟩

lemma *Exists-intro*: *atom i* ♯ *B* ⟹ *Exists* (⟦«*A IMP B*»⟧*e*) ⟦«(*Ex i A*) *IMP B*»⟧*e*
⟨*proof*⟩

Thus, we have precisely captured the codes of the specialisation axioms.

corollary *Exists-iff-exists*:
Exists p q ⟷ (∃ *A B i*. *p* = ⟦«*A IMP B*»⟧*e* ∧ *q* = ⟦«(*Ex i A*) *IMP B*»⟧*e* ∧ *atom i* ♯ *B*)
⟨*proof*⟩

6.1.9 The predicate *SubstP*, for the substitution rule

Although the substitution rule is derivable in the calculus, the derivation is too complicated to reproduce within the proof function. It is much easier to provide it as an immediate inference step, justifying its soundness in terms of other inference rules.

Definition

This is the inference $H \vdash A \Longrightarrow H \vdash A\ (i::=x)$

definition *Subst* :: *hf* ⇒ *hf* ⇒ *bool* **where**
Subst p q ≡ (∃ *v u*. *SubstForm v u p q*)

nominal-function *SubstP* :: *tm* ⇒ *tm* ⇒ *fm* **where**
⟦*atom u* ♯ (*p*,*q*,*v*); *atom v* ♯ (*p*,*q*)⟧ ⟹
SubstP p q = *Ex v* (*Ex u* (*SubstFormP* (*Var v*) (*Var u*) *p q*))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *SubstP-fresh-iff* [*simp*]: *a* ♯ *SubstP p q* ⟷ *a* ♯ *p* ∧ *a* ♯ *q* (**is** *?thesis1*)
and *eval-fm-SubstP* [*simp*]: *eval-fm e* (*SubstP p q*) ⟷ *Subst* ⟦*p*⟧*e* ⟦*q*⟧*e* (**is** *?thesis2*)
and *SubstP-sf* [*iff*]: *Sigma-fm* (*SubstP p q*) (**is** *?thesis3*)
⟨*proof*⟩

lemma *SubstP-subst* [*simp*]: (*SubstP p q*)(*j*::=*w*) = *SubstP* (*subst j w p*) (*subst j w q*)
⟨*proof*⟩

Correctness

lemma *Subst-imp-subst*:
assumes *Subst p q Form p*
shows ∃ *A*::*fm*. ∃ *i t*. *p* = ⟦«*A*»⟧*e* ∧ *q* = ⟦«*A*(*i*::=*t*)»⟧*e*
⟨*proof*⟩

6.1.10 The predicate *PrfP*

definition *Prf* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *Prf s k y* ≡ *BuildSeq* (λ*x*. *x* ∈ *Axiom*) (λ*u v w*. *ModPon v w u* ∨ *Exists v u* ∨ *Subst v u*) *s k y*

nominal-function *PrfP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom l* ♯ (*s*,*sl*,*m*,*n*,*sm*,*sn*); *atom sl* ♯ (*s*,*m*,*n*,*sm*,*sn*);
atom m ♯ (*s*,*n*,*sm*,*sn*); *atom n* ♯ (*s*,*k*,*sm*,*sn*);
atom sm ♯ (*s*,*sn*); *atom sn* ♯ (*s*)⟧ ⟹

PrfP s k t =
LstSeqP s k t AND
All2 n (SUCC k) (Ex sn (HPair (Var n) (Var sn) IN s AND (AxiomP (Var sn) OR
Ex m (Ex l (Ex sm (Ex sl (Var m IN Var n AND Var l IN Var n AND
HPair (Var m) (Var sm) IN s AND HPair (Var l) (Var sl) IN s AND
(ModPonP (Var sm) (Var sl) (Var sn) OR
ExistsP (Var sm) (Var sn) OR
SubstP (Var sm) (Var sn)))))))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *PrfP-fresh-iff* [*simp*]: $a \sharp PrfP\ s\ k\ t \longleftrightarrow a \sharp s \wedge a \sharp k \wedge a \sharp t$ (**is** *?thesis1*)
and *eval-fm-PrfP* [*simp*]: $eval\text{-}fm\ e\ (PrfP\ s\ k\ t) \longleftrightarrow Prf\ [\![s]\!]e\ [\![k]\!]e\ [\![t]\!]e$ (**is** *?thesis2*)
and *PrfP-imp-OrdP* [*simp*]: $\{PrfP\ s\ k\ t\} \vdash OrdP\ k$ (**is** *?thord*)
and *PrfP-imp-LstSeqP* [*simp*]: $\{PrfP\ s\ k\ t\} \vdash LstSeqP\ s\ k\ t$ (**is** *?thlstseq*)
and *PrfP-sf* [*iff*]: *Sigma-fm (PrfP s k t)* (**is** *?thsf*)
⟨*proof*⟩

lemma *PrfP-subst* [*simp*]:
$(PrfP\ t\ u\ v)(j::=w) = PrfP\ (subst\ j\ w\ t)\ (subst\ j\ w\ u)\ (subst\ j\ w\ v)$
⟨*proof*⟩

6.1.11 The predicate *PfP*

definition $Pf :: hf \Rightarrow bool$
where $Pf\ y \equiv (\exists s\ k.\ Prf\ s\ k\ y)$

nominal-function $PfP :: tm \Rightarrow fm$
where $[\![atom\ k \sharp (s,y);\ atom\ s \sharp y]\!] \Longrightarrow$
$PfP\ y = Ex\ k\ (Ex\ s\ (PrfP\ (Var\ s)\ (Var\ k)\ y))$
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *PfP-fresh-iff* [*simp*]: $a \sharp PfP\ y \longleftrightarrow a \sharp y$ (**is** *?thesis1*)
and *eval-fm-PfP* [*simp*]: $eval\text{-}fm\ e\ (PfP\ y) \longleftrightarrow Pf\ [\![y]\!]e$ (**is** *?thesis2*)
and *PfP-sf* [*iff*]: *Sigma-fm (PfP y)* (**is** *?thsf*)
⟨*proof*⟩

lemma *PfP-subst* [*simp*]: $(PfP\ t)(j::=w) = PfP\ (subst\ j\ w\ t)$

⟨*proof*⟩

lemma *ground-PfP* [*simp*]: *ground-fm* (*PfP y*) = *ground y*
⟨*proof*⟩

6.2 Proposition 4.4

6.2.1 Left-to-Right Proof

lemma *extra-axiom-imp-Pf*: *Pf* ⟦«*extra-axiom*»⟧*e*
⟨*proof*⟩

lemma *boolean-axioms-imp-Pf*:
assumes $\alpha \in$ *boolean-axioms* **shows** *Pf* ⟦«α»⟧*e*
⟨*proof*⟩

lemma *equality-axioms-imp-Pf*:
assumes $\alpha \in$ *equality-axioms* **shows** *Pf* ⟦«α»⟧*e*
⟨*proof*⟩

lemma *HF-axioms-imp-Pf*:
assumes $\alpha \in$ *HF-axioms* **shows** *Pf* ⟦«α»⟧*e*
⟨*proof*⟩

lemma *special-axioms-imp-Pf*:
assumes $\alpha \in$ *special-axioms* **shows** *Pf* ⟦«α»⟧*e*
⟨*proof*⟩

lemma *induction-axioms-imp-Pf*:
assumes $\alpha \in$ *induction-axioms* **shows** *Pf* ⟦«α»⟧*e*
⟨*proof*⟩

lemma *ModPon-imp-Pf*: ⟦*Pf* ⟦*Q-Imp x y*⟧*e*; *Pf* ⟦*x*⟧*e*⟧ ⟹ *Pf* ⟦*y*⟧*e*
⟨*proof*⟩

lemma *quot-ModPon-imp-Pf*: ⟦*Pf* ⟦«α *IMP* β»⟧*e*; *Pf* ⟦«α»⟧*e*⟧ ⟹ *Pf* ⟦«β»⟧*e*
⟨*proof*⟩

lemma *quot-Exists-imp-Pf*: ⟦*Pf* ⟦«α *IMP* β»⟧*e*; *atom i* ♯ β⟧ ⟹ *Pf* ⟦«*Ex i* α *IMP* β»⟧*e*
⟨*proof*⟩

lemma *proved-imp-Pf*: **assumes** $H \vdash \alpha$ H={} **shows** *Pf* ⟦«α»⟧*e*
⟨*proof*⟩

corollary *proved-imp-proved-PfP*: {} ⊢ α ⟹ {} ⊢ *PfP* «α»
⟨*proof*⟩

6.2.2 Right-to-Left Proof

lemma *Sent-imp-hfthm*:
assumes $x \in Sent$ **shows** $\exists\, A.\ x = [\![\langle\!\langle A \rangle\!\rangle]\!] e \wedge \{\} \vdash A$
⟨*proof*⟩

lemma *Extra-ax-imp-hfthm*:
assumes $x \in Extra\text{-}ax$ **obtains** A **where** $x = [\![\langle\!\langle A \rangle\!\rangle]\!] e \wedge \{\} \vdash A$
⟨*proof*⟩

lemma *Equality-ax-imp-hfthm*:
assumes $x \in Equality\text{-}ax$ **obtains** A **where** $x = [\![\langle\!\langle A \rangle\!\rangle]\!] e \wedge \{\} \vdash A$
⟨*proof*⟩

lemma *HF-ax-imp-hfthm*:
assumes $x \in HF\text{-}ax$ **obtains** A **where** $x = [\![\langle\!\langle A \rangle\!\rangle]\!] e \wedge \{\} \vdash A$
⟨*proof*⟩

lemma *Special-ax-imp-hfthm*:
assumes $x \in Special\text{-}ax$ **obtains** A **where** $x = [\![\langle\!\langle A \rangle\!\rangle]\!] e\ \{\} \vdash A$
⟨*proof*⟩

lemma *Induction-ax-imp-hfthm*:
assumes $x \in Induction\text{-}ax$ **obtains** A **where** $x = [\![\langle\!\langle A \rangle\!\rangle]\!] e\ \{\} \vdash A$
⟨*proof*⟩

lemma *Exists-imp-hfthm*: $[\![Exists\ [\![\langle\!\langle A \rangle\!\rangle]\!] e\ y;\ \{\} \vdash A]\!] \Longrightarrow \exists\, B.\ y = [\![\langle\!\langle B \rangle\!\rangle]\!] e \wedge \{\} \vdash B$
⟨*proof*⟩

lemma *Subst-imp-hfthm*: $[\![Subst\ [\![\langle\!\langle A \rangle\!\rangle]\!] e\ y;\ \{\} \vdash A]\!] \Longrightarrow \exists\, B.\ y = [\![\langle\!\langle B \rangle\!\rangle]\!] e \wedge \{\} \vdash B$
⟨*proof*⟩

lemma *eval-Neg-imp-Neg*: $[\![\langle\!\langle \alpha \rangle\!\rangle]\!] e = q\text{-}Neg\ x \Longrightarrow \exists\, A.\ \alpha = Neg\ A \wedge [\![\langle\!\langle A \rangle\!\rangle]\!] e = x$
⟨*proof*⟩

lemma *eval-Disj-imp-Disj*: $[\![\langle\!\langle \alpha \rangle\!\rangle]\!] e = q\text{-}Disj\ x\ y \Longrightarrow \exists\, A\ B.\ \alpha = A\ OR\ B \wedge [\![\langle\!\langle A \rangle\!\rangle]\!] e = x \wedge [\![\langle\!\langle B \rangle\!\rangle]\!] e = y$
⟨*proof*⟩

lemma *Prf-imp-proved*: **assumes** *Prf s k x* **shows** $\exists\, A.\ x = [\![\langle\!\langle A \rangle\!\rangle]\!] e \wedge \{\} \vdash A$
⟨*proof*⟩

corollary *Pf-quot-imp-is-proved*: $Pf\ [\![\langle\!\langle \alpha \rangle\!\rangle]\!] e \Longrightarrow \{\} \vdash \alpha$
⟨*proof*⟩

Proposition 4.4!

theorem *proved-iff-proved-PfP*: $\{\} \vdash \alpha \longleftrightarrow \{\} \vdash PfP\ \langle\!\langle \alpha \rangle\!\rangle$
⟨*proof*⟩

end

Kapitel 7

Uniqueness Results: Syntactic Relations are Functions

theory *Functions*
imports *Coding-Predicates*
begin

7.0.1 SeqStTermP

lemma *not-IndP-VarP*: $\{IndP\ x,\ VarP\ x\} \vdash A$
⟨*proof*⟩

It IS a pair, but not just any pair.

lemma *IndP-HPairE*: *insert* (*IndP* (*HPair* (*HPair Zero* (*HPair Zero Zero*)) *x*))
$H \vdash A$
⟨*proof*⟩

lemma *atom-HPairE*:
assumes $H \vdash x\ EQ\ HPair\ (HPair\ Zero\ (HPair\ Zero\ Zero))\ y$
shows *insert* (*IndP x OR x NEQ v*) $H \vdash A$
⟨*proof*⟩

lemma *SeqStTermP-lemma*:
assumes *atom m* ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*n*,*sm*,*sm*′,*sn*,*sn*′) *atom n* ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*sm*,*sm*′,*sn*,*sn*′)
atom sm ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*sm*′,*sn*,*sn*′) *atom sm*′ ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*sn*,*sn*′)
atom sn ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*sn*′) *atom sn*′ ♯ (*v*,*i*,*t*,*u*,*s*,*k*)
shows { *SeqStTermP v i t u s k* }
⊢ ((*t EQ v AND u EQ i*) *OR*
((*IndP t OR t NEQ v*) *AND u EQ t*)) *OR*
Ex m (*Ex n* (*Ex sm* (*Ex sm*′ (*Ex sn* (*Ex sn*′ (*Var m IN k AND Var n IN k AND*
SeqStTermP v i (*Var sm*) (*Var sm*′) *s* (*Var m*) *AND*
SeqStTermP v i (*Var sn*) (*Var sn*′) *s* (*Var n*) *AND*

t *EQ Q-Eats* (*Var sm*) (*Var sn*) *AND*
u *EQ Q-Eats* (*Var sm′*) (*Var sn′*)))))))
⟨*proof*⟩

lemma *SeqStTermP-unique*: {*SeqStTermP v a t u s kk*, *SeqStTermP v a t u′ s′ kk′*} ⊢ u' *EQ u*
⟨*proof*⟩

theorem *SubstTermP-unique*: {*SubstTermP v tm t u*, *SubstTermP v tm t u′*} ⊢ u' *EQ u*
⟨*proof*⟩

7.0.2 *SubstAtomicP*

lemma *SubstTermP-eq*:
⟦H ⊢ *SubstTermP v tm x z*; *insert* (*SubstTermP v tm y z*) H ⊢ A⟧ ⟹ *insert* (*x EQ y*) H ⊢ A
⟨*proof*⟩

lemma *SubstAtomicP-unique*: {*SubstAtomicP v tm x y*, *SubstAtomicP v tm x y′*} ⊢ y' *EQ y*
⟨*proof*⟩

7.0.3 *SeqSubstFormP*

lemma *SeqSubstFormP-lemma*:
assumes *atom m* ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*n*,*sm*,*sm′*,*sn*,*sn′*) *atom n* ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*sm*,*sm′*,*sn*,*sn′*)
atom sm ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*sm′*,*sn*,*sn′*) *atom sm′* ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*sn*,*sn′*)
atom sn ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*sn′*) *atom sn′* ♯ (*v*,*u*,*x*,*y*,*s*,*k*)
shows { *SeqSubstFormP v u x y s k* }
⊢ *SubstAtomicP v u x y OR*
Ex m (*Ex n* (*Ex sm* (*Ex sm′* (*Ex sn* (*Ex sn′* (*Var m IN k AND Var n IN k AND*
SeqSubstFormP v u (*Var sm*) (*Var sm′*) *s* (*Var m*) *AND*
SeqSubstFormP v u (*Var sn*) (*Var sn′*) *s* (*Var n*) *AND*
(((*x EQ Q-Disj* (*Var sm*) (*Var sn*) *AND y EQ Q-Disj* (*Var sm′*) (*Var sn′*)) *OR*
(*x EQ Q-Neg* (*Var sm*) *AND y EQ Q-Neg* (*Var sm′*)) *OR*
(*x EQ Q-Ex* (*Var sm*) *AND y EQ Q-Ex* (*Var sm′*)))))))))))
⟨*proof*⟩

lemma
shows *Neg-SubstAtomicP-Fls*: {*y EQ Q-Neg z*, *SubstAtomicP v tm y y′*} ⊢ *Fls* (**is** *?thesis1*)
and *Disj-SubstAtomicP-Fls*: {*y EQ Q-Disj z w*, *SubstAtomicP v tm y y′*} ⊢ *Fls* (**is** *?thesis2*)
and *Ex-SubstAtomicP-Fls*: {*y EQ Q-Ex z*, *SubstAtomicP v tm y y′*} ⊢ *Fls* (**is** *?thesis3*)

⟨*proof*⟩

lemma *SeqSubstFormP-eq*:
⟦$H \vdash$ *SeqSubstFormP v tm x z s k*; *insert* (*SeqSubstFormP v tm y z s k*) $H \vdash A$⟧
$\Longrightarrow$ *insert* (*x EQ y*) $H \vdash A$
⟨*proof*⟩

lemma *SeqSubstFormP-unique*: {*SeqSubstFormP v a x y s kk*, *SeqSubstFormP v a x y′ s′ kk′*} $\vdash$ *y′ EQ y*
⟨*proof*⟩

7.0.4 *SubstFormP*

theorem *SubstFormP-unique*: {*SubstFormP v tm x y*, *SubstFormP v tm x y′*} $\vdash$ *y′ EQ y*
⟨*proof*⟩

end

Kapitel 8

Section 6 Material and Gödel's First Incompleteness Theorem

theory *Goedel-I*
imports *Pf-Predicates Functions*
begin

8.1 The Function W and Lemma 6.1

8.1.1 Predicate form, defined on sequences

definition *SeqWR* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
  where *SeqWR s k y* ≡ *LstSeq s k y* ∧ *app s 0 = 0* ∧
                    (∀ *l* ∈ *k*. *app s* (*succ l*) = *q-Eats* (*app s l*) (*app s l*))

nominal-function *SeqWRP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
  where ⟦*atom l* ♯ (*s*,*k*,*sl*); *atom sl* ♯ (*s*)⟧ ⟹
    SeqWRP s k y = LstSeqP s k y AND
        HPair Zero Zero IN s AND
        All2 l k (*Ex sl* (*HPair* (*Var l*) (*Var sl*) *IN s AND*
                    HPair (*SUCC* (*Var l*)) (*Q-Succ* (*Var sl*)) *IN s*))
  ⟨*proof*⟩

nominal-termination (*eqvt*)
  ⟨*proof*⟩

lemma
  shows *SeqWRP-fresh-iff* [*simp*]: *a* ♯ *SeqWRP s k y* ⟷ *a* ♯ *s* ∧ *a* ♯ *k* ∧ *a* ♯ *y* (**is** *?thesis1*)
    and *eval-fm-SeqWRP* [*simp*]: *eval-fm e* (*SeqWRP s k y*) ⟷ *SeqWR* ⟦*s*⟧*e* ⟦*k*⟧*e* ⟦*y*⟧*e* (**is** *?thesis2*)
    and *SeqWRP-sf* [*iff*]: *Sigma-fm* (*SeqWRP s k y*) (**is** *?thsf*)

⟨*proof*⟩

lemma *SeqWRP-subst* [*simp*]:
(*SeqWRP s k y*)(*i*::=*t*) = *SeqWRP* (*subst i t s*) (*subst i t k*) (*subst i t y*)
⟨*proof*⟩

lemma *SeqWRP-cong*:
assumes $H \vdash s\ EQ\ s'$ **and** $H \vdash k\ EQ\ k'$ **and** $H \vdash y\ EQ\ y'$
shows $H \vdash SeqWRP\ s\ k\ y\ IFF\ SeqWRP\ s'\ k'\ y'$
⟨*proof*⟩

declare *SeqWRP.simps* [*simp del*]

8.1.2 Predicate form of W

definition $WR :: hf \Rightarrow hf \Rightarrow bool$
where $WR\ x\ y \equiv (\exists\, s.\ SeqWR\ s\ x\ y)$

nominal-function $WRP :: tm \Rightarrow tm \Rightarrow fm$
where $[\![atom\ s \sharp (x,y)]\!] \Longrightarrow$
$WRP\ x\ y = Ex\ s\ (SeqWRP\ (Var\ s)\ x\ y)$
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *WRP-fresh-iff* [*simp*]: $a \sharp WRP\ x\ y \longleftrightarrow a \sharp x \wedge a \sharp y$ (**is** *?thesis1*)
and *eval-fm-WRP* [*simp*]: $eval\text{-}fm\ e\ (WRP\ x\ y) \longleftrightarrow WR\ [\![x]\!]e\ [\![y]\!]e$ (**is** *?thesis2*)
and *sigma-fm-WRP* [*simp*]: *Sigma-fm* (*WRP x y*) (**is** *?thsf*)
⟨*proof*⟩

lemma *WRP-subst* [*simp*]: (*WRP x y*)(*i*::=*t*) = *WRP* (*subst i t x*) (*subst i t y*)
⟨*proof*⟩

lemma *WRP-cong*: $H \vdash t\ EQ\ t' \Longrightarrow H \vdash u\ EQ\ u' \Longrightarrow H \vdash WRP\ t\ u\ IFF\ WRP\ t'\ u'$
⟨*proof*⟩

declare *WRP.simps* [*simp del*]

lemma *WR0-iff*: $WR\ 0\ y \longleftrightarrow y{=}0$
⟨*proof*⟩

lemma *WR0*: *WR 0 0*
⟨*proof*⟩

lemma *WR-succ-iff*: **assumes** *i*: *Ord i* **shows** $WR\ (succ\ i)\ z = (\exists\, y.\ z = q\text{-}Eats$

$y\ y \wedge WR\ i\ y$)
⟨*proof*⟩

lemma *WR-succ*: $Ord\ i \Longrightarrow WR\ (succ\ i)\ (q\text{-}Eats\ y\ y) = WR\ i\ y$
⟨*proof*⟩

lemma *WR-ord-of*: $WR\ (ord\text{-}of\ i)\ [\![«ORD\text{-}OF\ i»]\!]e$
⟨*proof*⟩

Lemma 6.1

lemma *WR-quot-Var*: $WR\ [\![«Var\ x»]\!]e\ [\![««Var\ x»»]\!]e$
⟨*proof*⟩

lemma *ground-WRP* [*simp*]: $ground\text{-}fm\ (WRP\ x\ y) \longleftrightarrow ground\ x \wedge ground\ y$
⟨*proof*⟩

lemma *prove-WRP*: $\{\} \vdash WRP\ «Var\ x»\ ««Var\ x»»$
⟨*proof*⟩

8.1.3 Proving that these relations are functions

lemma *SeqWRP-Zero-E*:
assumes $insert\ (y\ EQ\ Zero)\ H \vdash A \quad H \vdash k\ EQ\ Zero$
shows $insert\ (SeqWRP\ s\ k\ y)\ H \vdash A$
⟨*proof*⟩

lemma *SeqWRP-SUCC-lemma*:
assumes y': $atom\ y' \sharp (s,k,y)$
shows $\{SeqWRP\ s\ (SUCC\ k)\ y\} \vdash Ex\ y'\ (SeqWRP\ s\ k\ (Var\ y')\ AND\ y\ EQ\ Q\text{-}Succ\ (Var\ y'))$
⟨*proof*⟩

lemma *SeqWRP-SUCC-E*:
assumes y': $atom\ y' \sharp (s,k,y)$ **and** k': $H \vdash k'\ EQ\ (SUCC\ k)$
shows $insert\ (SeqWRP\ s\ k'\ y)\ H \vdash Ex\ y'\ (SeqWRP\ s\ k\ (Var\ y')\ AND\ y\ EQ\ Q\text{-}Succ\ (Var\ y'))$
⟨*proof*⟩

lemma *SeqWRP-unique*: $\{OrdP\ x,\ SeqWRP\ s\ x\ y,\ SeqWRP\ s'\ x\ y'\} \vdash y'\ EQ\ y$
⟨*proof*⟩

theorem *WRP-unique*: $\{OrdP\ x,\ WRP\ x\ y,\ WRP\ x\ y'\} \vdash y'\ EQ\ y$
⟨*proof*⟩

8.1.4 The equivalent function

definition $W :: hf \Rightarrow tm$
where $W \equiv hmemrec\ (\lambda f\ z.\ if\ z{=}0\ then\ Zero\ else\ Q\text{-}Eats\ (f\ (pred\ z))\ (f\ (pred\ z)))$

lemma *W0* [*simp*]: *W 0 = Zero*
⟨*proof*⟩

lemma *W-succ* [*simp*]: *Ord i ⟹ W (succ i) = Q-Eats (W i) (W i)*
⟨*proof*⟩

lemma *W-ord-of* [*simp*]: *W (ord-of i) = «ORD-OF i»*
⟨*proof*⟩

lemma *WR-iff-eq-W*: *Ord x ⟹ WR x y ⟷ y = ⟦W x⟧e*
⟨*proof*⟩

8.2 The Function HF and Lemma 6.2

definition *SeqHR* :: *hf ⇒ hf ⇒ hf ⇒ hf ⇒ bool*
where *SeqHR x x′ s k ≡*
BuildSeq2 (λy y′. Ord y ∧ WR y y′)
(λu u′ v v′ w w′. u = ⟨v,w⟩ ∧ u′ = q-HPair v′ w′) s k x x′

8.2.1 Defining the syntax: quantified body

nominal-function *SeqHRP* :: *tm ⇒ tm ⇒ tm ⇒ tm ⇒ fm*
where ⟦*atom l ♯ (s,k,sl,sl′,m,n,sm,sm′,sn,sn′);*
atom sl ♯ (s,sl′,m,n,sm,sm′,sn,sn′);
atom sl′ ♯ (s,m,n,sm,sm′,sn,sn′);
atom m ♯ (s,n,sm,sm′,sn,sn′);
atom n ♯ (s,sm,sm′,sn,sn′);
atom sm ♯ (s,sm′,sn,sn′);
atom sm′ ♯ (s,sn,sn′);
atom sn ♯ (s,sn′);
atom sn′ ♯ (s)⟧ ⟹
SeqHRP x x′ s k =
LstSeqP s k (HPair x x′) AND
All2 l (SUCC k) (Ex sl (Ex sl′ (HPair (Var l) (HPair (Var sl) (Var sl′)) IN s AND
((OrdP (Var sl) AND WRP (Var sl) (Var sl′)) OR
Ex m (Ex n (Ex sm (Ex sm′ (Ex sn (Ex sn′ (Var m IN Var l AND Var n IN Var l AND
HPair (Var m) (HPair (Var sm) (Var sm′)) IN s AND
HPair (Var n) (HPair (Var sn) (Var sn′)) IN s AND
Var sl EQ HPair (Var sm) (Var sn) AND
Var sl′ EQ Q-HPair (Var sm′) (Var sn′)))))))))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *SeqHRP-fresh-iff* [*simp*]:

$a \sharp SeqHRP\ x\ x'\ s\ k \longleftrightarrow a \sharp x \wedge a \sharp x' \wedge a \sharp s \wedge a \sharp k$ (**is** *?thesis1*)
and *eval-fm-SeqHRP* [*simp*]:
$eval\text{-}fm\ e\ (SeqHRP\ x\ x'\ s\ k) \longleftrightarrow SeqHR\ [\![x]\!]e\ [\![x']\!]e\ [\![s]\!]e\ [\![k]\!]e$ (**is** *?thesis2*)
and *SeqHRP-sf* [*iff*]: *Sigma-fm* (*SeqHRP x x' s k*) (**is** *?thsf*)
and *SeqHRP-imp-OrdP*: { *SeqHRP x y s k* } ⊢ *OrdP k* (**is** *?thord*)
⟨*proof*⟩

lemma *SeqHRP-subst* [*simp*]:
$(SeqHRP\ x\ x'\ s\ k)(i::=t) = SeqHRP\ (subst\ i\ t\ x)\ (subst\ i\ t\ x')\ (subst\ i\ t\ s)\ (subst\ i\ t\ k)$
⟨*proof*⟩

lemma *SeqHRP-cong*:
assumes $H \vdash x\ EQ\ x'$ **and** $H \vdash y\ EQ\ y'$ $H \vdash s\ EQ\ s'$ **and** $H \vdash k\ EQ\ k'$
shows $H \vdash SeqHRP\ x\ y\ s\ k\ IFF\ SeqHRP\ x'\ y'\ s'\ k'$
⟨*proof*⟩

8.2.2 Defining the syntax: main predicate

definition $HR :: hf \Rightarrow hf \Rightarrow bool$
where $HR\ x\ x' \equiv \exists s\ k.\ SeqHR\ x\ x'\ s\ k$

nominal-function $HRP :: tm \Rightarrow tm \Rightarrow fm$
where $[\![atom\ s \sharp (x,x',k);\ atom\ k \sharp (x,x')]\!] \Longrightarrow$
$HRP\ x\ x' = Ex\ s\ (Ex\ k\ (SeqHRP\ x\ x'\ (Var\ s)\ (Var\ k)))$
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *HRP-fresh-iff* [*simp*]: $a \sharp HRP\ x\ x' \longleftrightarrow a \sharp x \wedge a \sharp x'$ (**is** *?thesis1*)
and *eval-fm-HRP* [*simp*]: $eval\text{-}fm\ e\ (HRP\ x\ x') \longleftrightarrow HR\ [\![x]\!]e\ [\![x']\!]e$ (**is** *?thesis2*)
and *HRP-sf* [*iff*]: *Sigma-fm* (*HRP x x'*) (**is** *?thsf*)
⟨*proof*⟩

lemma *HRP-subst* [*simp*]: $(HRP\ x\ x')(i::=t) = HRP\ (subst\ i\ t\ x)\ (subst\ i\ t\ x')$
⟨*proof*⟩

8.2.3 Proving that these relations are functions

lemma *SeqHRP-lemma*:
assumes $atom\ m \sharp (x,x',s,k,n,sm,sm',sn,sn')\ atom\ n \sharp (x,x',s,k,sm,sm',sn,sn')$
$atom\ sm \sharp (x,x',s,k,sm',sn,sn')\ atom\ sm' \sharp (x,x',s,k,sn,sn')$
$atom\ sn \sharp (x,x',s,k,sn')\ atom\ sn' \sharp (x,x',s,k)$
shows { *SeqHRP x x' s k* }
⊢ $(OrdP\ x\ AND\ WRP\ x\ x')\ OR$
$Ex\ m\ (Ex\ n\ (Ex\ sm\ (Ex\ sm'\ (Ex\ sn\ (Ex\ sn'\ (Var\ m\ IN\ k\ AND\ Var\ n\ IN\ k\ AND$
$SeqHRP\ (Var\ sm)\ (Var\ sm')\ s\ (Var\ m)\ AND$

SeqHRP (Var sn) (Var sn′) s (Var n) AND
x EQ HPair (Var sm) (Var sn) AND
x′ EQ Q-HPair (Var sm′) (Var sn′)))))))
⟨*proof*⟩

lemma *SeqHRP-unique*: {*SeqHRP x y s u*, *SeqHRP x y′ s′ u′*} ⊢ *y′ EQ y*
⟨*proof*⟩

theorem *HRP-unique*: {*HRP x y*, *HRP x y′*} ⊢ *y′ EQ y*
⟨*proof*⟩

8.2.4 Finally The Function HF Itself

definition *HF* :: *hf* ⇒ *tm*
where *HF* ≡ *hmemrec* (λ*f z*. *if Ord z then W z else Q-HPair* (*f* (*hfst z*)) (*f* (*hsnd z*)))

lemma *HF-Ord* [*simp*]: *Ord i* ⟹ *HF i* = *W i*
⟨*proof*⟩

lemma *HF-pair* [*simp*]: *HF* (*hpair x y*) = *Q-HPair* (*HF x*) (*HF y*)
⟨*proof*⟩

lemma *SeqHR-hpair*: *SeqHR x1 x3 s1 k1* ⟹ *SeqHR x2 x4 s2 k2* ⟹ ∃ *s k*. *SeqHR* ⟨*x1*,*x2*⟩ (*q-HPair x3 x4*) *s k*
⟨*proof*⟩

lemma *HR-H*: *coding-hf x* ⟹ *HR x* ⟦*HF x*⟧*e*
⟨*proof*⟩

Lemma 6.2

lemma *HF-quot-coding-tm*: *coding-tm t* ⟹ *HF* ⟦*t*⟧*e* = «*t*»
⟨*proof*⟩

lemma *HR-quot-fm*: **fixes** *A*::*fm* **shows** *HR* ⟦«*A*»⟧*e* ⟦««*A*»»⟧*e*
⟨*proof*⟩

lemma *prove-HRP*: **fixes** *A*::*fm* **shows** {} ⊢ *HRP* «*A*» ««*A*»»
⟨*proof*⟩

8.3 The Function K and Lemma 6.3

nominal-function *KRP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where *atom y* ♯ (*v*,*x*,*x′*) ⟹
KRP v x x′ = *Ex y* (*HRP x* (*Var y*) *AND SubstFormP v* (*Var y*) *x x′*)
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *KRP-fresh-iff* [*simp*]: $a \sharp KRP\ v\ x\ x' \longleftrightarrow a \sharp v \wedge a \sharp x \wedge a \sharp x'$
⟨*proof*⟩

lemma *KRP-subst* [*simp*]: $(KRP\ v\ x\ x')(i::=t) = KRP\ (subst\ i\ t\ v)\ (subst\ i\ t\ x)\ (subst\ i\ t\ x')$
⟨*proof*⟩

declare *KRP.simps* [*simp del*]

lemma *prove-SubstFormP*: $\{\} \vdash SubstFormP\ «Var\ i»\ ««A»»\ «A»\ «A(i::=«A»)»$
⟨*proof*⟩

lemma *prove-KRP*: $\{\} \vdash KRP\ «Var\ i»\ «A»\ «A(i::=«A»)»$
⟨*proof*⟩

lemma *KRP-unique*: $\{KRP\ v\ x\ y,\ KRP\ v\ x\ y'\} \vdash y'\ EQ\ y$
⟨*proof*⟩

lemma *KRP-subst-fm*: $\{KRP\ «Var\ i»\ «\beta»\ (Var\ j)\} \vdash Var\ j\ EQ\ «\beta(i::=«\beta»)»$
⟨*proof*⟩

8.4 The Diagonal Lemma and Gödel's Theorem

lemma *diagonal*:
obtains δ **where** $\{\} \vdash \delta\ IFF\ \alpha(i::=«\delta»)$ $supp\ \delta = supp\ \alpha - \{atom\ i\}$
⟨*proof*⟩

Gödel's first incompleteness theorem: Our theory is incomplete. NB it is provably consistent

theorem *Goedel-I*:
obtains δ **where** $\{\} \vdash \delta\ IFF\ Neg\ (PfP\ «\delta»)$ $\neg\ \{\} \vdash \delta$ $\neg\ \{\} \vdash Neg\ \delta$
eval-fm e δ *ground-fm* δ
⟨*proof*⟩

end

Kapitel 9

Syntactic Preliminaries for the Second Incompleteness Theorem

theory *II-Prelims*
imports *Pf-Predicates*
begin

declare *IndP.simps* [*simp del*]

lemma *VarP-Var* [*intro*]: $H \vdash$ *VarP* «*Var i*»
⟨*proof*⟩

lemma *VarP-neq-IndP*: {*t EQ v*, *VarP v*, *IndP t*} ⊢ *Fls*
⟨*proof*⟩

lemma *OrdP-ORD-OF* [*intro*]: $H \vdash$ *OrdP* (*ORD-OF n*)
⟨*proof*⟩

lemma *Mem-HFun-Sigma-OrdP*: {*HPair t u IN f*, *HFun-Sigma f*} ⊢ *OrdP t*
⟨*proof*⟩

9.1 NotInDom

nominal-function *NotInDom* :: *tm* ⇒ *tm* ⇒ *fm*
  where *atom z* ♯ (*t*, *r*) ⟹ *NotInDom t r* = *All z* (*Neg* (*HPair t* (*Var z*) *IN r*))
⟨*proof*⟩

nominal-termination (*eqvt*)
  ⟨*proof*⟩

lemma *NotInDom-fresh-iff* [*simp*]: *a* ♯ *NotInDom t r* ⟷ *a* ♯ (*t*, *r*)
⟨*proof*⟩

lemma *subst-fm-NotInDom* [*simp*]: $(NotInDom\ t\ r)(i::=x) = NotInDom\ (subst\ i\ x\ t)\ (subst\ i\ x\ r)$
⟨*proof*⟩

lemma *NotInDom-cong*: $H \vdash t\ EQ\ t' \Longrightarrow H \vdash r\ EQ\ r' \Longrightarrow H \vdash NotInDom\ t\ r\ IFF\ NotInDom\ t'\ r'$
⟨*proof*⟩

lemma *NotInDom-Zero*: $H \vdash NotInDom\ t\ Zero$
⟨*proof*⟩

lemma *NotInDom-Fls*: $\{HPair\ d\ d'\ IN\ r,\ NotInDom\ d\ r\} \vdash A$
⟨*proof*⟩

lemma *NotInDom-Contra*: $H \vdash NotInDom\ d\ r \Longrightarrow H \vdash HPair\ x\ y\ IN\ r \Longrightarrow insert\ (x\ EQ\ d)\ H \vdash A$
⟨*proof*⟩

9.2 Restriction of a Sequence to a Domain

nominal-function *RestrictedP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $[\![atom\ x \sharp (y,f,k,g);\ atom\ y \sharp (f,k,g)]\!] \Longrightarrow$
$RestrictedP\ f\ k\ g =$
$g\ SUBS\ f\ AND$
$All\ x\ (All\ y\ (HPair\ (Var\ x)\ (Var\ y)\ IN\ g\ IFF$
$(Var\ x)\ IN\ k\ AND\ HPair\ (Var\ x)\ (Var\ y)\ IN\ f))$
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *RestrictedP-fresh-iff* [*simp*]: $a \sharp RestrictedP\ f\ k\ g \longleftrightarrow a \sharp f \wedge a \sharp k \wedge a \sharp g$
⟨*proof*⟩

lemma *subst-fm-RestrictedP* [*simp*]:
$(RestrictedP\ f\ k\ g)(i::=u) = RestrictedP\ (subst\ i\ u\ f)\ (subst\ i\ u\ k)\ (subst\ i\ u\ g)$
⟨*proof*⟩

lemma *RestrictedP-cong*:
$[\![H \vdash f\ EQ\ f';\ H \vdash k\ EQ\ A';\ H \vdash g\ EQ\ g']\!]$
$\Longrightarrow H \vdash RestrictedP\ f\ k\ g\ IFF\ RestrictedP\ f'\ A'\ g'$
⟨*proof*⟩

lemma *RestrictedP-Zero*: $H \vdash RestrictedP\ Zero\ k\ Zero$
⟨*proof*⟩

lemma *RestrictedP-Mem*: $\{\ RestrictedP\ s\ k\ s',\ HPair\ a\ b\ IN\ s,\ a\ IN\ k\ \} \vdash HPair$

a b IN s′
⟨*proof*⟩

lemma *RestrictedP-imp-Subset*: {*RestrictedP s k s′*} ⊢ *s′ SUBS s*
⟨*proof*⟩

lemma *RestrictedP-Mem2*:
{ *RestrictedP s k s′*, *HPair a b IN s′* } ⊢ *HPair a b IN s AND a IN k*
⟨*proof*⟩

lemma *RestrictedP-Mem-D*: *H* ⊢ *RestrictedP s k t* ⟹ *H* ⊢ *a IN t* ⟹ *insert* (*a IN s*) *H* ⊢ *A* ⟹ *H* ⊢ *A*
⟨*proof*⟩

lemma *RestrictedP-Eats*:
{ *RestrictedP s k s′*, *a IN k* } ⊢ *RestrictedP* (*Eats s* (*HPair a b*)) *k* (*Eats s′* (*HPair a b*)) ⟨*proof*⟩
lemma *exists-RestrictedP*:
assumes *s*: *atom s* ♯ (*f*,*k*)
shows *H* ⊢ *Ex s* (*RestrictedP f k* (*Var s*)) ⟨*proof*⟩
lemma *cut-RestrictedP*:
assumes *s*: *atom s* ♯ (*f*,*k*,*A*) **and** ∀ *C* ∈ *H*. *atom s* ♯ *C*
shows *insert* (*RestrictedP f k* (*Var s*)) *H* ⊢ *A* ⟹ *H* ⊢ *A*
⟨*proof*⟩

lemma *RestrictedP-NotInDom*: { *RestrictedP s k s′*, *Neg* (*j IN k*) } ⊢ *NotInDom j s′*
⟨*proof*⟩

declare *RestrictedP.simps* [*simp del*]

9.3 Applications to LstSeqP

lemma *HFun-Sigma-Eats*:
assumes *H* ⊢ *HFun-Sigma r H* ⊢ *NotInDom d r H* ⊢ *OrdP d*
shows *H* ⊢ *HFun-Sigma* (*Eats r* (*HPair d d′*)) ⟨*proof*⟩
lemma *HFun-Sigma-single* [*iff*]: *H* ⊢ *OrdP d* ⟹ *H* ⊢ *HFun-Sigma* (*Eats Zero* (*HPair d d′*))
⟨*proof*⟩

lemma *LstSeqP-single* [*iff*]: *H* ⊢ *LstSeqP* (*Eats Zero* (*HPair Zero x*)) *Zero x*
⟨*proof*⟩

lemma *NotInDom-LstSeqP-Eats*:
{ *NotInDom* (*SUCC k*) *s*, *LstSeqP s k y* } ⊢ *LstSeqP* (*Eats s* (*HPair* (*SUCC k*) *z*)) (*SUCC k*) *z*
⟨*proof*⟩

lemma *RestrictedP-HDomain-Incl*: {*HDomain-Incl s k*, *RestrictedP s k s′*} ⊢ *HDo-*

main-Incl s′ k
⟨*proof*⟩

lemma *RestrictedP-HFun-Sigma*: {*HFun-Sigma s*, *RestrictedP s k s′*} ⊢ *HFun-Sigma s′*
⟨*proof*⟩

lemma *RestrictedP-LstSeqP*:
{ *RestrictedP s* (*SUCC k*) *s′*, *LstSeqP s k y* } ⊢ *LstSeqP s′ k y*
⟨*proof*⟩

lemma *RestrictedP-LstSeqP-Eats*:
{ *RestrictedP s* (*SUCC k*) *s′*, *LstSeqP s k y* }
⊢ *LstSeqP* (*Eats s′* (*HPair* (*SUCC k*) *z*)) (*SUCC k*) *z*
⟨*proof*⟩

9.4 Ordinal Addition

9.4.1 Predicate form, defined on sequences

nominal-function *SeqHaddP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom l* ♯ (*sl*,*s*,*k*,*j*); *atom sl* ♯ (*s*,*j*)⟧ ⟹
SeqHaddP s j k y = *LstSeqP s k y AND*
HPair Zero j IN s AND
All2 l k (*Ex sl* (*HPair* (*Var l*) (*Var sl*) *IN s AND*
HPair (*SUCC* (*Var l*)) (*SUCC* (*Var sl*)) *IN s*))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *SeqHaddP-fresh-iff* [*simp*]: *a* ♯ *SeqHaddP s j k y* ⟷ *a* ♯ *s* ∧ *a* ♯ *j* ∧ *a* ♯ *k* ∧ *a* ♯ *y*
⟨*proof*⟩

lemma *SeqHaddP-subst* [*simp*]:
(*SeqHaddP s j k y*)(*i*::=*t*) = *SeqHaddP* (*subst i t s*) (*subst i t j*) (*subst i t k*) (*subst i t y*)
⟨*proof*⟩

declare *SeqHaddP.simps* [*simp del*]

nominal-function *HaddP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom s* ♯ (*x*,*y*,*z*)⟧ ⟹
HaddP x y z = *Ex s* (*SeqHaddP* (*Var s*) *x y z*)
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *HaddP-fresh-iff* [*simp*]: $a \sharp HaddP\ x\ y\ z \longleftrightarrow a \sharp x \wedge a \sharp y \wedge a \sharp z$
⟨*proof*⟩

lemma *HaddP-subst* [*simp*]: $(HaddP\ x\ y\ z)(i::=t) = HaddP\ (subst\ i\ t\ x)\ (subst\ i\ t\ y)\ (subst\ i\ t\ z)$
⟨*proof*⟩

lemma *HaddP-cong*: $[\![H \vdash t\ EQ\ t';\ H \vdash u\ EQ\ u';\ H \vdash v\ EQ\ v']\!] \Longrightarrow H \vdash HaddP\ t\ u\ v\ IFF\ HaddP\ t'\ u'\ v'$
⟨*proof*⟩

declare *HaddP.simps* [*simp del*]

lemma *HaddP-Zero2*: $H \vdash HaddP\ x\ Zero\ x$
⟨*proof*⟩

lemma *HaddP-imp-OrdP*: $\{HaddP\ x\ y\ z\} \vdash OrdP\ y$
⟨*proof*⟩

lemma *HaddP-SUCC2*: $\{HaddP\ x\ y\ z\} \vdash HaddP\ x\ (SUCC\ y)\ (SUCC\ z)$ ⟨*proof*⟩

9.4.2 Proving that these relations are functions

lemma *SeqHaddP-Zero-E*: $\{SeqHaddP\ s\ w\ Zero\ z\} \vdash w\ EQ\ z$
⟨*proof*⟩

lemma *SeqHaddP-SUCC-lemma*:
assumes y': $atom\ y' \sharp (s,j,k,y)$
shows $\{SeqHaddP\ s\ j\ (SUCC\ k)\ y\} \vdash Ex\ y'\ (SeqHaddP\ s\ j\ k\ (Var\ y')\ AND\ y\ EQ\ SUCC\ (Var\ y'))$
⟨*proof*⟩

lemma *SeqHaddP-SUCC*:
assumes $H \vdash SeqHaddP\ s\ j\ (SUCC\ k)\ y\ \ atom\ y' \sharp (s,j,k,y)$
shows $H \vdash Ex\ y'\ (SeqHaddP\ s\ j\ k\ (Var\ y')\ AND\ y\ EQ\ SUCC\ (Var\ y'))$
⟨*proof*⟩

lemma *SeqHaddP-unique*: $\{OrdP\ x,\ SeqHaddP\ s\ w\ x\ y,\ SeqHaddP\ s'\ w\ x\ y'\} \vdash y'\ EQ\ y$ ⟨*proof*⟩
lemma *HaddP-unique*: $\{HaddP\ w\ x\ y,\ HaddP\ w\ x\ y'\} \vdash y'\ EQ\ y$
⟨*proof*⟩

lemma *HaddP-Zero1*: **assumes** $H \vdash OrdP\ x$ **shows** $H \vdash HaddP\ Zero\ x\ x$
⟨*proof*⟩

lemma *HaddP-Zero-D1*: $insert\ (HaddP\ Zero\ x\ y)\ H \vdash x\ EQ\ y$
⟨*proof*⟩

lemma *HaddP-Zero-D2*: *insert* (*HaddP x Zero y*) *H* ⊢ *x EQ y*
⟨*proof*⟩

lemma *HaddP-SUCC-Ex2*:
assumes *H* ⊢ *HaddP x* (*SUCC y*) *z atom z′* ♯ (*x*,*y*,*z*)
shows *H* ⊢ *Ex z′* (*HaddP x y* (*Var z′*) *AND z EQ SUCC* (*Var z′*))
⟨*proof*⟩

lemma *HaddP-SUCC1*: { *HaddP x y z* } ⊢ *HaddP* (*SUCC x*) *y* (*SUCC z*) ⟨*proof*⟩
lemma *HaddP-commute*: {*HaddP x y z*, *OrdP x*} ⊢ *HaddP y x z* ⟨*proof*⟩
lemma *HaddP-SUCC-Ex1*:
assumes *atom i* ♯ (*x*,*y*,*z*)
shows *insert* (*HaddP* (*SUCC x*) *y z*) (*insert* (*OrdP x*) *H*)
⊢ *Ex i* (*HaddP x y* (*Var i*) *AND z EQ SUCC* (*Var i*))
⟨*proof*⟩

lemma *HaddP-inv2*: {*HaddP x y z*, *HaddP x y′ z*, *OrdP x*} ⊢ *y′ EQ y* ⟨*proof*⟩
lemma *Mem-imp-subtract*: ⟨*proof*⟩
lemma *HaddP-OrdP*:
assumes *H* ⊢ *HaddP x y z H* ⊢ *OrdP x* **shows** *H* ⊢ *OrdP z* ⟨*proof*⟩
lemma *HaddP-Mem-cancel-left*:
assumes *H* ⊢ *HaddP x y′ z′ H* ⊢ *HaddP x y z H* ⊢ *OrdP x*
shows *H* ⊢ *z′ IN z IFF y′ IN y* ⟨*proof*⟩
lemma *HaddP-Mem-cancel-right-Mem*:
assumes *H* ⊢ *HaddP x′ y z′ H* ⊢ *HaddP x y z H* ⊢ *x′ IN x H* ⊢ *OrdP x*
shows *H* ⊢ *z′ IN z*
⟨*proof*⟩

lemma *HaddP-Mem-cases*:
assumes *H* ⊢ *HaddP k1 k2 k H* ⊢ *OrdP k1*
insert (*x IN k1*) *H* ⊢ *A*
insert (*Var i IN k2*) (*insert* (*HaddP k1* (*Var i*) *x*) *H*) ⊢ *A*
and *i*: *atom* (*i*::*name*) ♯ (*k1*,*k2*,*k*,*x*,*A*) **and** ∀ *C* ∈ *H*. *atom i* ♯ *C*
shows *insert* (*x IN k*) *H* ⊢ *A* ⟨*proof*⟩
lemma *HaddP-Mem-contra*:
assumes *H* ⊢ *HaddP x y z H* ⊢ *z IN x H* ⊢ *OrdP x*
shows *H* ⊢ *A*
⟨*proof*⟩

lemma *exists-HaddP*:
assumes *H* ⊢ *OrdP y atom j* ♯ (*x*,*y*)
shows *H* ⊢ *Ex j* (*HaddP x y* (*Var j*))
⟨*proof*⟩

lemma *HaddP-Mem-I*:
assumes *H* ⊢ *HaddP x y z H* ⊢ *OrdP x* **shows** *H* ⊢ *x IN SUCC z*
⟨*proof*⟩

9.5 A Shifted Sequence

nominal-function *ShiftP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
  where ⟦*atom x* ♯ (*x′*,*y*,*z*,*f*,*del*,*k*); *atom x′* ♯ (*y*,*z*,*f*,*del*,*k*); *atom y* ♯ (*z*,*f*,*del*,*k*); *atom z* ♯ (*f*,*del*,*g*,*k*)⟧ ⟹
    ShiftP f k del g =
      All z (*Var z IN g IFF*
      (*Ex x* (*Ex x′* (*Ex y* ((*Var z*) *EQ HPair* (*Var x′*) (*Var y*) *AND*
                    HaddP del (*Var x*) (*Var x′*) *AND*
                    HPair (*Var x*) (*Var y*) *IN f AND Var x IN k*)))))
⟨*proof*⟩

nominal-termination (*eqvt*)
  ⟨*proof*⟩

lemma *ShiftP-fresh-iff* [*simp*]: *a* ♯ *ShiftP f k del g* ⟷ *a* ♯ *f* ∧ *a* ♯ *k* ∧ *a* ♯ *del* ∧ *a* ♯ *g*
⟨*proof*⟩

lemma *subst-fm-ShiftP* [*simp*]:
  (*ShiftP f k del g*)(*i*::=*u*) = *ShiftP* (*subst i u f*) (*subst i u k*) (*subst i u del*) (*subst i u g*)
⟨*proof*⟩

lemma *ShiftP-Zero*: {} ⊢ *ShiftP Zero k d Zero*
⟨*proof*⟩

lemma *ShiftP-Mem1*:
  {*ShiftP f k del g*, *HPair a b IN f*, *HaddP del a a′*, *a IN k*} ⊢ *HPair a′ b IN g*
⟨*proof*⟩

lemma *ShiftP-Mem2*:
  assumes *atom u* ♯ (*f*,*k*,*del*,*a*,*b*)
  shows {*ShiftP f k del g*, *HPair a b IN g*} ⊢ *Ex u* ((*Var u*) *IN k AND HaddP del* (*Var u*) *a AND HPair* (*Var u*) *b IN f*)
⟨*proof*⟩

lemma *ShiftP-Mem-D*:
  assumes *H* ⊢ *ShiftP f k del g H* ⊢ *a IN g*
        atom x ♯ (*x′*,*y*,*a*,*f*,*del*,*k*) *atom x′* ♯ (*y*,*a*,*f*,*del*,*k*) *atom y* ♯ (*a*,*f*,*del*,*k*)
  shows *H* ⊢ (*Ex x* (*Ex x′* (*Ex y* (*a EQ HPair* (*Var x′*) (*Var y*) *AND*
                    HaddP del (*Var x*) (*Var x′*) *AND*
                    HPair (*Var x*) (*Var y*) *IN f AND Var x IN k*))))
        (**is** - ⊢ *?concl*)
⟨*proof*⟩

lemma *ShiftP-Eats-Eats*:
  {*ShiftP f k del g*, *HaddP del a a′*, *a IN k*}
  ⊢ *ShiftP* (*Eats f* (*HPair a b*)) *k del* (*Eats g* (*HPair a′ b*)) ⟨*proof*⟩

lemma *ShiftP-Eats-Neg*:
assumes *atom u* ♯ (*u′*,*v*,*f*,*k*,*del*,*g*,*c*) *atom u′* ♯ (*v*,*f*,*k*,*del*,*g*,*c*) *atom v* ♯ (*f*,*k*,*del*,*g*,*c*)
shows
{*ShiftP f k del g*,
Neg (*Ex u* (*Ex u′* (*Ex v* (*c EQ HPair* (*Var u*) (*Var v*) *AND Var u IN k AND HaddP del* (*Var u*) (*Var u′*)))))}
⊢ *ShiftP* (*Eats f c*) *k del g* ⟨*proof*⟩
lemma *exists-ShiftP*:
assumes *t*: *atom t* ♯ (*s*,*k*,*del*)
shows *H* ⊢ *Ex t* (*ShiftP s k del* (*Var t*)) ⟨*proof*⟩

9.6 Union of Two Sets

nominal-function *UnionP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where *atom i* ♯ (*x*,*y*,*z*) ⟹ *UnionP x y z* = *All i* (*Var i IN z IFF* (*Var i IN x OR Var i IN y*))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *UnionP-fresh-iff* [*simp*]: *a* ♯ *UnionP x y z* ⟷ *a* ♯ *x* ∧ *a* ♯ *y* ∧ *a* ♯ *z*
⟨*proof*⟩

lemma *subst-fm-UnionP* [*simp*]:
(*UnionP x y z*)(*i*::=*u*) = *UnionP* (*subst i u x*) (*subst i u y*) (*subst i u z*)
⟨*proof*⟩

lemma *Union-Zero1*: *H* ⊢ *UnionP Zero x x*
⟨*proof*⟩

lemma *Union-Eats*: {*UnionP x y z*} ⊢ *UnionP* (*Eats x a*) *y* (*Eats z a*)
⟨*proof*⟩

lemma *exists-Union-lemma*:
assumes *z*: *atom z* ♯ (*i*,*y*) **and** *i*: *atom i* ♯ *y*
shows {} ⊢ *Ex z* (*UnionP* (*Var i*) *y* (*Var z*))
⟨*proof*⟩

lemma *exists-UnionP*:
assumes *z*: *atom z* ♯ (*x*,*y*) **shows** *H* ⊢ *Ex z* (*UnionP x y* (*Var z*))
⟨*proof*⟩

lemma *UnionP-Mem1*: { *UnionP x y z*, *a IN x* } ⊢ *a IN z*
⟨*proof*⟩

lemma *UnionP-Mem2*: { *UnionP x y z*, *a IN y* } ⊢ *a IN z*
⟨*proof*⟩

lemma *UnionP-Mem*: { *UnionP x y z*, *a IN z* } ⊢ *a IN x OR a IN y*
⟨*proof*⟩

lemma *UnionP-Mem-E*:
assumes *H* ⊢ *UnionP x y z*
and *insert* (*a IN x*) *H* ⊢ *A*
and *insert* (*a IN y*) *H* ⊢ *A*
shows *insert* (*a IN z*) *H* ⊢ *A*
⟨*proof*⟩

9.7 Append on Sequences

nominal-function *SeqAppendP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom g1* ♯ (*g2*,*f1*,*k1*,*f2*,*k2*,*g*); *atom g2* ♯ (*f1*,*k1*,*f2*,*k2*,*g*)⟧ ⟹
SeqAppendP f1 k1 f2 k2 g =
(*Ex g1* (*Ex g2* (*RestrictedP f1 k1* (*Var g1*) *AND*
ShiftP f2 k2 k1 (*Var g2*) *AND*
UnionP (*Var g1*) (*Var g2*) *g*)))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma *SeqAppendP-fresh-iff* [*simp*]:
a ♯ *SeqAppendP f1 k1 f2 k2 g* ⟷ *a* ♯ *f1* ∧ *a* ♯ *k1* ∧ *a* ♯ *f2* ∧ *a* ♯ *k2* ∧ *a* ♯ *g*
⟨*proof*⟩

lemma *subst-fm-SeqAppendP* [*simp*]:
(*SeqAppendP f1 k1 f2 k2 g*)(*i*::=*u*) =
SeqAppendP (*subst i u f1*) (*subst i u k1*) (*subst i u f2*) (*subst i u k2*) (*subst i u g*)
⟨*proof*⟩

lemma *exists-SeqAppendP*:
assumes *atom g* ♯ (*f1*,*k1*,*f2*,*k2*)
shows *H* ⊢ *Ex g* (*SeqAppendP f1 k1 f2 k2* (*Var g*))
⟨*proof*⟩

lemma *SeqAppendP-Mem1*: {*SeqAppendP f1 k1 f2 k2 g*, *HPair x y IN f1*, *x IN k1*}
⊢ *HPair x y IN g*
⟨*proof*⟩

lemma *SeqAppendP-Mem2*: {*SeqAppendP f1 k1 f2 k2 g*, *HaddP k1 x x′*, *x IN k2*,
HPair x y IN f2} ⊢ *HPair x′ y IN g*
⟨*proof*⟩

lemma *SeqAppendP-Mem-E*:
assumes *H* ⊢ *SeqAppendP f1 k1 f2 k2 g*
and *insert* (*HPair x y IN f1*) (*insert* (*x IN k1*) *H*) ⊢ *A*

and *insert* (*HPair* (*Var u*) *y IN f2*) (*insert* (*HaddP k1* (*Var u*) *x*) (*insert* (*Var u IN k2*) *H*)) ⊢ *A*
and *u*: *atom u* ♯ (*f1*,*k1*,*f2*,*k2*,*x*,*y*,*g*,*A*) ∀ *C* ∈ *H*. *atom u* ♯ *C*
shows *insert* (*HPair x y IN g*) *H* ⊢ *A* ⟨*proof*⟩

9.8 LstSeqP and SeqAppendP

lemma *HDomain-Incl-SeqAppendP*: — The And eliminates the need to prove *cut5*
{*SeqAppendP f1 k1 f2 k2 g*, *HDomain-Incl f1 k1 AND HDomain-Incl f2 k2*,
HaddP k1 k2 k, *OrdP k1*} ⊢ *HDomain-Incl g k* ⟨*proof*⟩
declare *SeqAppendP.simps* [*simp del*]

lemma *HFun-Sigma-SeqAppendP*:
{*SeqAppendP f1 k1 f2 k2 g*, *HFun-Sigma f1*, *HFun-Sigma f2*, *OrdP k1*} ⊢ *HFun-Sigma g* ⟨*proof*⟩
lemma *LstSeqP-SeqAppendP*:
assumes *H* ⊢ *SeqAppendP f1* (*SUCC k1*) *f2* (*SUCC k2*) *g*
H ⊢ *LstSeqP f1 k1 y1 H* ⊢ *LstSeqP f2 k2 y2 H* ⊢ *HaddP k1 k2 k*
shows *H* ⊢ *LstSeqP g* (*SUCC k*) *y2*
⟨*proof*⟩

lemma *SeqAppendP-NotInDom*: {*SeqAppendP f1 k1 f2 k2 g*, *HaddP k1 k2 k*, *OrdP k1*} ⊢ *NotInDom k g*
⟨*proof*⟩

lemma *LstSeqP-SeqAppendP-Eats*:
assumes *H* ⊢ *SeqAppendP f1* (*SUCC k1*) *f2* (*SUCC k2*) *g*
H ⊢ *LstSeqP f1 k1 y1 H* ⊢ *LstSeqP f2 k2 y2 H* ⊢ *HaddP k1 k2 k*
shows *H* ⊢ *LstSeqP* (*Eats g* (*HPair* (*SUCC* (*SUCC k*)) *z*)) (*SUCC* (*SUCC k*)) *z*
⟨*proof*⟩

9.9 Substitution and Abstraction on Terms

9.9.1 Atomic cases

lemma *SeqStTermP-Var-same*:
assumes *atom s* ♯ (*k*,*v*,*i*) *atom k* ♯ (*v*,*i*)
shows {*VarP v*} ⊢ *Ex s* (*Ex k* (*SeqStTermP v i v i* (*Var s*) (*Var k*)))
⟨*proof*⟩

lemma *SeqStTermP-Var-diff*:
assumes *atom s* ♯ (*k*,*v*,*w*,*i*) *atom k* ♯ (*v*,*w*,*i*)
shows {*VarP v*, *VarP w*, *Neg* (*v EQ w*) } ⊢ *Ex s* (*Ex k* (*SeqStTermP v i w w* (*Var s*) (*Var k*)))
⟨*proof*⟩

lemma *SeqStTermP-Zero*:

assumes *atom s ♯ (k,v,i) atom k ♯ (v,i)*
shows {*VarP v*} ⊢ *Ex s (Ex k (SeqStTermP v i Zero Zero (Var s) (Var k)))*
⟨*proof*⟩
corollary *SubstTermP-Zero*: {*TermP t*} ⊢ *SubstTermP «Var v» t Zero Zero*
⟨*proof*⟩

corollary *SubstTermP-Var-same*: {*VarP v, TermP t*} ⊢ *SubstTermP v t v t*
⟨*proof*⟩

corollary *SubstTermP-Var-diff*: {*VarP v, VarP w, Neg (v EQ w), TermP t*} ⊢ *SubstTermP v t w w*
⟨*proof*⟩

lemma *SeqStTermP-Ind*:
assumes *atom s ♯ (k,v,t,i) atom k ♯ (v,t,i)*
shows {*VarP v, IndP t*} ⊢ *Ex s (Ex k (SeqStTermP v i t t (Var s) (Var k)))*
⟨*proof*⟩

corollary *SubstTermP-Ind*: {*VarP v, IndP w, TermP t*} ⊢ *SubstTermP v t w w*
⟨*proof*⟩

9.9.2 Non-atomic cases

lemma *SeqStTermP-Eats*:
assumes *sk*: *atom s ♯ (k,s1,s2,k1,k2,t1,t2,u1,u2,v,i)*
atom k ♯ (t1,t2,u1,u2,v,i)
shows {*SeqStTermP v i t1 u1 s1 k1, SeqStTermP v i t2 u2 s2 k2*}
⊢ *Ex s (Ex k (SeqStTermP v i (Q-Eats t1 t2) (Q-Eats u1 u2) (Var s) (Var k)))* ⟨*proof*⟩
theorem *SubstTermP-Eats*:
{*SubstTermP v i t1 u1, SubstTermP v i t2 u2*} ⊢ *SubstTermP v i (Q-Eats t1 t2) (Q-Eats u1 u2)*
⟨*proof*⟩

9.9.3 Substitution over a constant

lemma *SeqConstP-lemma*:
assumes *atom m ♯ (s,k,c,n,sm,sn) atom n ♯ (s,k,c,sm,sn)*
atom sm ♯ (s,k,c,sn) atom sn ♯ (s,k,c)
shows { *SeqConstP s k c* }
⊢ *c EQ Zero OR*
Ex m (Ex n (Ex sm (Ex sn (Var m IN k AND Var n IN k AND
SeqConstP s (Var m) (Var sm) AND
SeqConstP s (Var n) (Var sn) AND
c EQ Q-Eats (Var sm) (Var sn))))) ⟨*proof*⟩
lemma *SeqConstP-imp-SubstTermP*: {*SeqConstP s kk c, TermP t*} ⊢ *SubstTermP «Var w» t c c* ⟨*proof*⟩
theorem *SubstTermP-Const*: {*ConstP c, TermP t*} ⊢ *SubstTermP «Var w» t c c*
⟨*proof*⟩

9.10 Substitution on Formulas

9.10.1 Membership

lemma *SubstAtomicP-Mem*:
$\{SubstTermP\ v\ i\ x\ x',\ SubstTermP\ v\ i\ y\ y'\} \vdash SubstAtomicP\ v\ i\ (Q\text{-}Mem\ x\ y)\ (Q\text{-}Mem\ x'\ y')$
⟨*proof*⟩

lemma *SeqSubstFormP-Mem*:
assumes $atom\ s \sharp (k,x,y,x',y',v,i)\ atom\ k \sharp (x,y,x',y',v,i)$
shows $\{SubstTermP\ v\ i\ x\ x',\ SubstTermP\ v\ i\ y\ y'\}$
$\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q\text{-}Mem\ x\ y)\ (Q\text{-}Mem\ x'\ y')\ (Var\ s)\ (Var\ k)))$
⟨*proof*⟩

lemma *SubstFormP-Mem*:
$\{SubstTermP\ v\ i\ x\ x',\ SubstTermP\ v\ i\ y\ y'\} \vdash SubstFormP\ v\ i\ (Q\text{-}Mem\ x\ y)\ (Q\text{-}Mem\ x'\ y')$
⟨*proof*⟩

9.10.2 Equality

lemma *SubstAtomicP-Eq*:
$\{SubstTermP\ v\ i\ x\ x',\ SubstTermP\ v\ i\ y\ y'\} \vdash SubstAtomicP\ v\ i\ (Q\text{-}Eq\ x\ y)\ (Q\text{-}Eq\ x'\ y')$
⟨*proof*⟩

lemma *SeqSubstFormP-Eq*:
assumes *sk*: $atom\ s \sharp (k,x,y,x',y',v,i)\ atom\ k \sharp (x,y,x',y',v,i)$
shows $\{SubstTermP\ v\ i\ x\ x',\ SubstTermP\ v\ i\ y\ y'\}$
$\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q\text{-}Eq\ x\ y)\ (Q\text{-}Eq\ x'\ y')\ (Var\ s)\ (Var\ k)))$
⟨*proof*⟩

lemma *SubstFormP-Eq*:
$\{SubstTermP\ v\ i\ x\ x',\ SubstTermP\ v\ i\ y\ y'\} \vdash SubstFormP\ v\ i\ (Q\text{-}Eq\ x\ y)\ (Q\text{-}Eq\ x'\ y')$
⟨*proof*⟩

9.10.3 Negation

lemma *SeqSubstFormP-Neg*:
assumes $atom\ s \sharp (k,s1,k1,x,x',v,i)\ atom\ k \sharp (s1,k1,x,x',v,i)$
shows $\{SeqSubstFormP\ v\ i\ x\ x'\ s1\ k1,\ TermP\ i,\ VarP\ v\}$
$\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q\text{-}Neg\ x)\ (Q\text{-}Neg\ x')\ (Var\ s)\ (Var\ k)))$
⟨*proof*⟩
theorem *SubstFormP-Neg*: $\{SubstFormP\ v\ i\ x\ x'\} \vdash SubstFormP\ v\ i\ (Q\text{-}Neg\ x)\ (Q\text{-}Neg\ x')$
⟨*proof*⟩

9.10.4 Disjunction

lemma *SeqSubstFormP-Disj*:
assumes *atom s* ♯ (*k*,*s1*,*s2*,*k1*,*k2*,*x*,*y*,*x′*,*y′*,*v*,*i*) *atom k* ♯ (*s1*,*s2*,*k1*,*k2*,*x*,*y*,*x′*,*y′*,*v*,*i*)
shows {*SeqSubstFormP v i x x′ s1 k1*,
SeqSubstFormP v i y y′ s2 k2, *TermP i*, *VarP v*}
⊢ *Ex s* (*Ex k* (*SeqSubstFormP v i* (*Q-Disj x y*) (*Q-Disj x′ y′*) (*Var s*) (*Var k*))) ⟨*proof*⟩
theorem *SubstFormP-Disj*:
{*SubstFormP v i x x′*, *SubstFormP v i y y′*} ⊢ *SubstFormP v i* (*Q-Disj x y*) (*Q-Disj x′ y′*)
⟨*proof*⟩

9.10.5 Existential

lemma *SeqSubstFormP-Ex*:
assumes *atom s* ♯ (*k*,*s1*,*k1*,*x*,*x′*,*v*,*i*) *atom k* ♯ (*s1*,*k1*,*x*,*x′*,*v*,*i*)
shows {*SeqSubstFormP v i x x′ s1 k1*, *TermP i*, *VarP v*}
⊢ *Ex s* (*Ex k* (*SeqSubstFormP v i* (*Q-Ex x*) (*Q-Ex x′*) (*Var s*) (*Var k*))) ⟨*proof*⟩
theorem *SubstFormP-Ex*: {*SubstFormP v i x x′*} ⊢ *SubstFormP v i* (*Q-Ex x*) (*Q-Ex x′*)
⟨*proof*⟩

9.11 Constant Terms

lemma *ConstP-Zero*: {} ⊢ *ConstP Zero*
⟨*proof*⟩

lemma *SeqConstP-Eats*:
assumes *atom s* ♯ (*k*,*s1*,*s2*,*k1*,*k2*,*t1*,*t2*) *atom k* ♯ (*s1*,*s2*,*k1*,*k2*,*t1*,*t2*)
shows {*SeqConstP s1 k1 t1*, *SeqConstP s2 k2 t2*}
⊢ *Ex s* (*Ex k* (*SeqConstP* (*Var s*) (*Var k*) (*Q-Eats t1 t2*))) ⟨*proof*⟩
theorem *ConstP-Eats*: {*ConstP t1*, *ConstP t2*} ⊢ *ConstP* (*Q-Eats t1 t2*)
⟨*proof*⟩

9.12 Proofs

lemma *PrfP-inference*:
assumes *atom s* ♯ (*k*,*s1*,*s2*,*k1*,*k2*,*α1*,*α2*,*β*) *atom k* ♯ (*s1*,*s2*,*k1*,*k2*,*α1*,*α2*,*β*)
shows {*PrfP s1 k1 α1*, *PrfP s2 k2 α2*, *ModPonP α1 α2 β OR ExistsP α1 β OR SubstP α1 β*}
⊢ *Ex k* (*Ex s* (*PrfP* (*Var s*) (*Var k*) *β*)) ⟨*proof*⟩
corollary *PfP-inference*: {*PfP α1*, *PfP α2*, *ModPonP α1 α2 β OR ExistsP α1 β OR SubstP α1 β*} ⊢ *PfP β*
⟨*proof*⟩

theorem *PfP-implies-SubstForm-PfP*:
assumes *H* ⊢ *PfP y H* ⊢ *SubstFormP x t y z*

shows $H \vdash PfP\ z$
⟨*proof*⟩

theorem *PfP-implies-ModPon-PfP*: $[\![H \vdash PfP\ (Q\text{-}Imp\ x\ y);\ H \vdash PfP\ x]\!] \Longrightarrow H \vdash PfP\ y$
⟨*proof*⟩

corollary *PfP-implies-ModPon-PfP-quot*: $[\![H \vdash PfP\ «\alpha\ IMP\ \beta»;\ H \vdash PfP\ «\alpha»]\!] \Longrightarrow H \vdash PfP\ «\beta»$
⟨*proof*⟩

end

Kapitel 10

Pseudo-Coding: Section 7 Material

theory *Pseudo-Coding*
imports *II-Prelims*
begin

10.1 General Lemmas

lemma *Collect-disj-Un*: $\{f\ i\ |i.\ P\ i \vee Q\ i\} = \{f\ i\ |i.\ P\ i\} \cup \{f\ i\ |i.\ Q\ i\}$
⟨*proof*⟩

abbreviation *Q-Subset* :: *tm* ⇒ *tm* ⇒ *tm*
where *Q-Subset t u* ≡ (*Q-All* (*Q-Imp* (*Q-Mem* (*Q-Ind Zero*) *t*) (*Q-Mem* (*Q-Ind Zero*) *u*)))

lemma *NEQ-quot-tm*: *i*≠*j* ⟹ {} ⊢ «*Var i*» *NEQ* «*Var j*»
⟨*proof*⟩

lemma *EQ-quot-tm-Fls*: *i*≠*j* ⟹ *insert* («*Var i*» *EQ* «*Var j*») *H* ⊢ *Fls*
⟨*proof*⟩

lemma *perm-commute*: *a* ♯ *p* ⟹ *a′* ♯ *p* ⟹ (*a* ⇌ *a′*) + *p* = *p* + (*a* ⇌ *a′*)
⟨*proof*⟩

lemma *perm-self-inverseI*: ⟦−*p* = *q*; *a* ♯ *p*; *a′* ♯ *p*⟧ ⟹ − ((*a* ⇌ *a′*) + *p*) = (*a* ⇌ *a′*) + *q*
⟨*proof*⟩

lemma *fresh-image*:
fixes *f* :: *'a* ⇒ *'b*::*fs* **shows** *finite A* ⟹ *i* ♯ *f* ‘ *A* ⟷ (∀ *x*∈*A*. *i* ♯ *f x*)
⟨*proof*⟩

lemma *atom-in-atom-image* [*simp*]: *atom j* ∈ *atom* ‘ *V* ⟷ *j* ∈ *V*

⟨*proof*⟩

lemma *fresh-star-empty* [*simp*]: {} ♯∗ *bs*
⟨*proof*⟩

declare *fresh-star-insert* [*simp*]

lemma *fresh-star-finite-insert*:
fixes *S* :: ('*a*::*fs*) *set* **shows** *finite S* ⟹ *a* ♯∗ *insert x S* ⟷ *a* ♯∗ *x* ∧ *a* ♯∗ *S*
⟨*proof*⟩

lemma *fresh-finite-Diff-single* [*simp*]:
fixes *V* :: *name set* **shows** *finite V* ⟹ *a* ♯ (*V* − {*j*}) ⟷ (*a* ♯ *j* ⟶ *a* ♯ *V*)
⟨*proof*⟩

lemma *fresh-image-atom* [*simp*]: *finite A* ⟹ *i* ♯ *atom* ‘ *A* ⟷ *i* ♯ *A*
⟨*proof*⟩

lemma *atom-fresh-star-atom-set-conv*: ⟦*atom i* ♯ *bs*; *finite bs*⟧ ⟹ *bs* ♯∗ *i*
⟨*proof*⟩

lemma *notin-V*:
assumes *p*: *atom i* ♯ *p* **and** *V*: *finite V atom* ‘ (*p* ∙ *V*) ♯∗ *V*
shows *i* ∉ *V i* ∉ *p* ∙ *V*
⟨*proof*⟩

10.2 Simultaneous Substitution

definition *ssubst* :: *tm* ⇒ *name set* ⇒ (*name* ⇒ *tm*) ⇒ *tm*
where *ssubst t V F* = *Finite-Set.fold* (λ*i*. *subst i* (*F i*)) *t V*

definition *make-F* :: *name set* ⇒ *perm* ⇒ *name* ⇒ *tm*
where *make-F Vs p* ≡ λ*i*. *if i* ∈ *Vs then Var* (*p* ∙ *i*) *else Var i*

lemma *ssubst-empty* [*simp*]: *ssubst t* {} *F* = *t*
⟨*proof*⟩

Renaming a finite set of variables. Based on the theorem *at-set-avoiding*

locale *quote-perm* =
fixes *p* :: *perm* **and** *Vs* :: *name set* **and** *F* :: *name* ⇒ *tm*
assumes *p*: *atom* ‘ (*p* ∙ *Vs*) ♯∗ *Vs*
and *pinv*: −*p* = *p*
and *Vs*: *finite Vs*
defines *F* ≡ *make-F Vs p*
begin

lemma *F-unfold*: *F i* = (*if i* ∈ *Vs then Var* (*p* ∙ *i*) *else Var i*)
⟨*proof*⟩

lemma *finite-V* [*simp*]: $V \subseteq Vs \Longrightarrow finite\ V$
⟨*proof*⟩

lemma *perm-exits-Vs*: $i \in Vs \Longrightarrow (p \cdot i) \notin Vs$
⟨*proof*⟩

lemma *atom-fresh-perm*: $[\![x \in Vs;\ y \in Vs]\!] \Longrightarrow atom\ x \sharp p \cdot y$
⟨*proof*⟩

lemma *fresh-pj*: $[\![a \sharp p;\ j \in Vs]\!] \Longrightarrow a \sharp p \cdot j$
⟨*proof*⟩

lemma *fresh-Vs*: $a \sharp p \Longrightarrow a \sharp Vs$
⟨*proof*⟩

lemma *fresh-pVs*: $a \sharp p \Longrightarrow a \sharp p \cdot Vs$
⟨*proof*⟩

lemma assumes $V \subseteq Vs\ a \sharp p$
shows *fresh-pV* [*simp*]: $a \sharp p \cdot V$ **and** *fresh-V* [*simp*]: $a \sharp V$
⟨*proof*⟩

lemma *qp-insert*:
fixes *i*::*name* **and** *i'*::*name*
assumes $atom\ i \sharp p\ atom\ i' \sharp (i,p)$
shows $quote\text{-}perm\ ((atom\ i \rightleftharpoons atom\ i') + p)\ (insert\ i\ Vs)$
⟨*proof*⟩

lemma *subst-F-left-commute*: $subst\ x\ (F\ x)\ (subst\ y\ (F\ y)\ t) = subst\ y\ (F\ y)\ (subst\ x\ (F\ x)\ t)$
⟨*proof*⟩

lemma
assumes $finite\ V\ i \notin V$
shows *ssubst-insert*: $ssubst\ t\ (insert\ i\ V)\ F = subst\ i\ (F\ i)\ (ssubst\ t\ V\ F)$ (**is** *?thesis1*)
and *ssubst-insert2*: $ssubst\ t\ (insert\ i\ V)\ F = ssubst\ (subst\ i\ (F\ i)\ t)\ V\ F$ (**is** *?thesis2*)
⟨*proof*⟩

lemma *ssubst-insert-if*:
$finite\ V \Longrightarrow$
$ssubst\ t\ (insert\ i\ V)\ F = (if\ i \in V\ then\ ssubst\ t\ V\ F$
$else\ subst\ i\ (F\ i)\ (ssubst\ t\ V\ F))$
⟨*proof*⟩

lemma *ssubst-single* [*simp*]: $ssubst\ t\ \{i\}\ F = subst\ i\ (F\ i)\ t$
⟨*proof*⟩

lemma *ssubst-Var-if* [*simp*]:
  assumes *finite V*
    shows *ssubst* (*Var i*) *V F* = (*if i* ∈ *V then F i else Var i*)
⟨*proof*⟩

lemma *ssubst-Zero* [*simp*]: *finite V* ⟹ *ssubst Zero V F* = *Zero*
  ⟨*proof*⟩

lemma *ssubst-Eats* [*simp*]: *finite V* ⟹ *ssubst* (*Eats t u*) *V F* = *Eats* (*ssubst t V F*) (*ssubst u V F*)
  ⟨*proof*⟩

lemma *ssubst-SUCC* [*simp*]: *finite V* ⟹ *ssubst* (*SUCC t*) *V F* = *SUCC* (*ssubst t V F*)
  ⟨*proof*⟩

lemma *ssubst-ORD-OF* [*simp*]: *finite V* ⟹ *ssubst* (*ORD-OF n*) *V F* = *ORD-OF n*
  ⟨*proof*⟩

lemma *ssubst-HPair* [*simp*]:
  finite V ⟹ *ssubst* (*HPair t u*) *V F* = *HPair* (*ssubst t V F*) (*ssubst u V F*)
  ⟨*proof*⟩

lemma *ssubst-HTuple* [*simp*]: *finite V* ⟹ *ssubst* (*HTuple n*) *V F* = (*HTuple n*)
  ⟨*proof*⟩

lemma *ssubst-Subset*:
  assumes *finite V* **shows** *ssubst* ⌊*t SUBS u*⌋ *V V F* = *Q-Subset* (*ssubst* ⌊*t*⌋ *V V F*) (*ssubst* ⌊*u*⌋ *V V F*)
⟨*proof*⟩

lemma *fresh-ssubst*:
  assumes *finite V a* ♯ *p* ∙ *V a* ♯ *t*
    shows *a* ♯ *ssubst t V F*
⟨*proof*⟩

lemma *fresh-ssubst′*:
  assumes *finite V atom i* ♯ *t atom* (*p* ∙ *i*) ♯ *t*
    shows *atom i* ♯ *ssubst t V F*
⟨*proof*⟩

lemma *ssubst-vquot-Ex*:
  ⟦*finite V*; *atom i* ♯ *p* ∙ *V*⟧
   ⟹ *ssubst* ⌊*Ex i A*⌋(*insert i V*) (*insert i V*) *F* = *ssubst* ⌊*Ex i A*⌋ *V V F*
   ⟨*proof*⟩

lemma *ground-ssubst-eq*: ⟦*finite V*; *supp t* = {}⟧ ⟹ *ssubst t V F* = *t*
  ⟨*proof*⟩

lemma *ssubst-quot-tm* [*simp*]:
fixes *t*::*tm* **shows** *finite V* $\Longrightarrow$ *ssubst* «*t*» *V F* = «*t*»
⟨*proof*⟩

lemma *ssubst-quot-fm* [*simp*]:
fixes *A*::*fm* **shows** *finite V* $\Longrightarrow$ *ssubst* «*A*» *V F* = «*A*»
⟨*proof*⟩

lemma *atom-in-p-Vs*: ⟦*i* ∈ *p* ∙ *V*; *V* ⊆ *Vs*⟧ $\Longrightarrow$ *i* ∈ *p* ∙ *Vs*
⟨*proof*⟩

10.3 The Main Theorems of Section 7

lemma *SubstTermP-vquot-dbtm*:
assumes *w*: *w* ∈ *Vs* − *V* **and** *V*: *V* ⊆ *Vs* *V'* = *p* ∙ *V*
and *s*: *supp dbtm* ⊆ *atom* ‘ *Vs*
shows
insert (*ConstP* (*F w*)) {*ConstP* (*F i*) | *i*. *i* ∈ *V*}
⊢ *SubstTermP* «*Var w*» (*F w*)
(*ssubst* (*vquot-dbtm V dbtm*) *V F*)
(*subst w* (*F w*) (*ssubst* (*vquot-dbtm* (*insert w V*) *dbtm*) *V F*))
⟨*proof*⟩

lemma *SubstFormP-vquot-dbfm*:
assumes *w*: *w* ∈ *Vs* − *V* **and** *V*: *V* ⊆ *Vs* *V'* = *p* ∙ *V*
and *s*: *supp dbfm* ⊆ *atom* ‘ *Vs*
shows
insert (*ConstP* (*F w*)) {*ConstP* (*F i*) | *i*. *i* ∈ *V*}
⊢ *SubstFormP* «*Var w*» (*F w*)
(*ssubst* (*vquot-dbfm V dbfm*) *V F*)
(*subst w* (*F w*) (*ssubst* (*vquot-dbfm* (*insert w V*) *dbfm*) *V F*))
⟨*proof*⟩

Lemmas 7.5 and 7.6

lemma *ssubst-SubstFormP*:
fixes *A*::*fm*
assumes *w*: *w* ∈ *Vs* − *V* **and** *V*: *V* ⊆ *Vs* *V'* = *p* ∙ *V*
and *s*: *supp A* ⊆ *atom* ‘ *Vs*
shows
insert (*ConstP* (*F w*)) {*ConstP* (*F i*) | *i*. *i* ∈ *V*}
⊢ *SubstFormP* «*Var w*» (*F w*)
(*ssubst* ⌊*A*⌋ *V V F*)
(*ssubst* ⌊*A*⌋(*insert w V*) (*insert w V*) *F*)
⟨*proof*⟩

Theorem 7.3

theorem *PfP-implies-PfP-ssubst*:
fixes *β*::*fm*

assumes β: $\{\} \vdash PfP\ «\beta»$
and V: $V \subseteq Vs$
and s: $supp\ \beta \subseteq atom\ `\ Vs$
shows $\{ConstP\ (F\ i) \mid i.\ i \in V\} \vdash PfP\ (ssubst\ \lfloor\beta\rfloor\ V\ V\ F)$
⟨*proof*⟩

end

end

Kapitel 11

Quotations of the Free Variables

theory *Quote*
imports *Pseudo-Coding*
begin

11.1 Sequence version of the "Special p-Function, F*"

The definition below describes a relation, not a function. This material relates to Section 8, but omits the ordering of the universe.

definition *SeqQuote* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SeqQuote x x′ s k* ≡
BuildSeq2 (λ*y y′*. *y*=*0* ∧ *y′* = *0*)
(λ*u u′ v v′ w w′*. *u* = *v* ◁ *w* ∧ *u′* = *q-Eats v′ w′*) *s k x x′*

11.1.1 Defining the syntax: quantified body

nominal-function *SeqQuoteP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom l* ♯ (*s*,*k*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*);
atom sl ♯ (*s*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*); *atom sl′* ♯ (*s*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*);
atom m ♯ (*s*,*n*,*sm*,*sm′*,*sn*,*sn′*); *atom n* ♯ (*s*,*sm*,*sm′*,*sn*,*sn′*);
atom sm ♯ (*s*,*sm′*,*sn*,*sn′*); *atom sm′* ♯ (*s*,*sn*,*sn′*);
atom sn ♯ (*s*,*sn′*); *atom sn′* ♯ *s*⟧ ⟹
SeqQuoteP t u s k =
LstSeqP s k (*HPair t u*) *AND*
All2 l (*SUCC k*) (*Ex sl* (*Ex sl′* (*HPair* (*Var l*) (*HPair* (*Var sl*) (*Var sl′*)) *IN s AND*
((*Var sl EQ Zero AND Var sl′ EQ Zero*) *OR*
Ex m (*Ex n* (*Ex sm* (*Ex sm′* (*Ex sn* (*Ex sn′* (*Var m IN Var l AND Var n IN Var l AND*
HPair (*Var m*) (*HPair* (*Var sm*) (*Var sm′*)) *IN s AND*

HPair (Var n) (HPair (Var sn) (Var sn')) IN s AND
Var sl EQ Eats (Var sm) (Var sn) AND
Var sl' EQ Q-Eats (Var sm') (Var sn'))))))))))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *SeqQuoteP-fresh-iff* [*simp*]:
a ♯ SeqQuoteP t u s k ⟷ a ♯ t ∧ a ♯ u ∧ a ♯ s ∧ a ♯ k (**is** *?thesis1*)
and *eval-fm-SeqQuoteP* [*simp*]:
eval-fm e (SeqQuoteP t u s k) ⟷ SeqQuote ⟦t⟧e ⟦u⟧e ⟦s⟧e ⟦k⟧e (**is** *?thesis2*)
and *SeqQuoteP-sf* [*iff*]:
Sigma-fm (SeqQuoteP t u s k) (**is** *?thsf*)
and *SeqQuoteP-imp-OrdP*:
{ SeqQuoteP t u s k } ⊢ OrdP k (**is** *?thord*)
and *SeqQuoteP-imp-LstSeqP*:
{ SeqQuoteP t u s k } ⊢ LstSeqP s k (HPair t u) (**is** *?thlstseq*)
⟨*proof*⟩

lemma *SeqQuoteP-subst* [*simp*]:
(SeqQuoteP t u s k)(j::=w) =
SeqQuoteP (subst j w t) (subst j w u) (subst j w s) (subst j w k)
⟨*proof*⟩

declare *SeqQuoteP.simps* [*simp del*]

11.1.2 Correctness properties

lemma *SeqQuoteP-lemma*:
fixes *m*::*name* **and** *sm*::*name* **and** *sm'*::*name* **and** *n*::*name* **and** *sn*::*name* **and** *sn'*::*name*
assumes *atom m ♯ (t,u,s,k,n,sm,sm',sn,sn')* *atom n ♯ (t,u,s,k,sm,sm',sn,sn')*
atom sm ♯ (t,u,s,k,sm',sn,sn') *atom sm' ♯ (t,u,s,k,sn,sn')*
atom sn ♯ (t,u,s,k,sn') *atom sn' ♯ (t,u,s,k)*
shows *{ SeqQuoteP t u s k }*
⊢ (t EQ Zero AND u EQ Zero) OR
Ex m (Ex n (Ex sm (Ex sm' (Ex sn (Ex sn' (Var m IN k AND Var n IN k AND
SeqQuoteP (Var sm) (Var sm') s (Var m) AND
SeqQuoteP (Var sn) (Var sn') s (Var n) AND
t EQ Eats (Var sm) (Var sn) AND
u EQ Q-Eats (Var sm') (Var sn')))))))
⟨*proof*⟩

11.2 The "special function" itself

definition *Quote* :: *hf ⇒ hf ⇒ bool*

where *Quote x x′* ≡ ∃ *s k*. *SeqQuote x x′ s k*

11.2.1 Defining the syntax

nominal-function *QuoteP* :: *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom s* ♯ (*t*,*u*,*k*); *atom k* ♯ (*t*,*u*)⟧ ⟹
QuoteP t u = *Ex s* (*Ex k* (*SeqQuoteP t u* (*Var s*) (*Var k*)))
⟨*proof*⟩

nominal-termination (*eqvt*)
⟨*proof*⟩

lemma
shows *QuoteP-fresh-iff* [*simp*]: *a* ♯ *QuoteP t u* ⟷ *a* ♯ *t* ∧ *a* ♯ *u* (**is** *?thesis1*)
and *eval-fm-QuoteP* [*simp*]: *eval-fm e* (*QuoteP t u*) ⟷ *Quote* ⟦*t*⟧*e* ⟦*u*⟧*e* (**is** *?thesis2*)
and *QuoteP-sf* [*iff*]: *Sigma-fm* (*QuoteP t u*) (**is** *?thsf*)
⟨*proof*⟩

lemma *QuoteP-subst* [*simp*]:
(*QuoteP t u*)(*j*::=*w*) = *QuoteP* (*subst j w t*) (*subst j w u*)
⟨*proof*⟩

declare *QuoteP.simps* [*simp del*]

11.2.2 Correctness properties

lemma *Quote-0*: *Quote 0 0*
⟨*proof*⟩

lemma *QuoteP-Zero*: {} ⊢ *QuoteP Zero Zero*
⟨*proof*⟩

lemma *SeqQuoteP-Eats*:
assumes *atom s* ♯ (*k*,*s1*,*s2*,*k1*,*k2*,*t1*,*t2*,*u1*,*u2*) *atom k* ♯ (*s1*,*s2*,*k1*,*k2*,*t1*,*t2*,*u1*,*u2*)
shows {*SeqQuoteP t1 u1 s1 k1*, *SeqQuoteP t2 u2 s2 k2*} ⊢
Ex s (*Ex k* (*SeqQuoteP* (*Eats t1 t2*) (*Q-Eats u1 u2*) (*Var s*) (*Var k*)))
⟨*proof*⟩

lemma *QuoteP-Eats*: {*QuoteP t1 u1*, *QuoteP t2 u2*} ⊢ *QuoteP* (*Eats t1 t2*) (*Q-Eats u1 u2*)
⟨*proof*⟩

lemma *exists-QuoteP*:
assumes *j*: *atom j* ♯ *x* **shows** {} ⊢ *Ex j* (*QuoteP x* (*Var j*))
⟨*proof*⟩

lemma *QuoteP-imp-ConstP*: { *QuoteP x y* } ⊢ *ConstP y*

⟨*proof*⟩

lemma *SeqQuoteP-imp-QuoteP*: $\{SeqQuoteP\ t\ u\ s\ k\} \vdash QuoteP\ t\ u$
⟨*proof*⟩

lemmas *QuoteP-I* = *SeqQuoteP-imp-QuoteP* [*THEN cut1*]

11.3 The Operator *quote-all*

11.3.1 Definition and basic properties

definition *quote-all* :: [*perm*, *name set*] ⇒ *fm set*
  where $quote\text{-}all\ p\ V = \{QuoteP\ (Var\ i)\ (Var\ (p \cdot i)) \mid i.\ i \in V\}$

lemma *quote-all-empty* [*simp*]: *quote-all p* {} = {}
  ⟨*proof*⟩

lemma *quote-all-insert* [*simp*]:
  $quote\text{-}all\ p\ (insert\ i\ V) = insert\ (QuoteP\ (Var\ i)\ (Var\ (p \cdot i)))\ (quote\text{-}all\ p\ V)$
  ⟨*proof*⟩

lemma *finite-quote-all* [*simp*]: *finite V* ⟹ *finite* (*quote-all p V*)
  ⟨*proof*⟩

lemma *fresh-quote-all* [*simp*]: *finite V* ⟹ $i \sharp quote\text{-}all\ p\ V \longleftrightarrow i \sharp V \wedge i \sharp p \cdot V$
  ⟨*proof*⟩

lemma *fresh-quote-all-mem*: ⟦$A \in quote\text{-}all\ p\ V$; *finite V*; $i \sharp V$; $i \sharp p \cdot V$⟧ ⟹ $i \sharp A$
  ⟨*proof*⟩

lemma *quote-all-perm-eq*:
  assumes *finite V atom i* ♯ (*p*, *V*) *atom i′* ♯ (*p*, *V*)
    shows $quote\text{-}all\ ((atom\ i \rightleftharpoons atom\ i') + p)\ V = quote\text{-}all\ p\ V$
⟨*proof*⟩

11.3.2 Transferring theorems to the level of derivability

context *quote-perm*
begin

lemma *QuoteP-imp-ConstP-F-hyps*:
  assumes $Us \subseteq Vs$ $\{ConstP\ (F\ i) \mid i.\ i \in Us\} \vdash A$ **shows** $quote\text{-}all\ p\ Us \vdash A$
⟨*proof*⟩

Lemma 8.3

theorem *quote-all-PfP-ssubst*:
  assumes β: $\{\} \vdash \beta$
    and V: $V \subseteq Vs$

and *s*: *supp* $\beta \subseteq$ *atom* ‘ *Vs*
shows *quote-all p V* $\vdash$ *PfP* (*ssubst* $\lfloor\beta\rfloor$ *V V F*)
⟨*proof*⟩

Lemma 8.4

corollary *quote-all-MonPon-PfP-ssubst*:
assumes *A*: {} $\vdash$ α *IMP* β
and *V*: *V* $\subseteq$ *Vs*
and *s*: *supp* $\alpha \subseteq$ *atom* ‘ *Vs supp* $\beta \subseteq$ *atom* ‘ *Vs*
shows *quote-all p V* $\vdash$ *PfP* (*ssubst* $\lfloor\alpha\rfloor$ *V V F*) *IMP PfP* (*ssubst* $\lfloor\beta\rfloor$ *V V F*)
⟨*proof*⟩

Lemma 8.4b

corollary *quote-all-MonPon2-PfP-ssubst*:
assumes *A*: {} $\vdash$ α*1 IMP* α*2 IMP* β
and *V*: *V* $\subseteq$ *Vs*
and *s*: *supp* α*1* $\subseteq$ *atom* ‘ *Vs supp* α*2* $\subseteq$ *atom* ‘ *Vs supp* $\beta \subseteq$ *atom* ‘ *Vs*
shows *quote-all p V* $\vdash$ *PfP* (*ssubst* $\lfloor\alpha$*1*$\rfloor$ *V V F*) *IMP PfP* (*ssubst* $\lfloor\alpha$*2*$\rfloor$ *V V F*) *IMP PfP* (*ssubst* $\lfloor\beta\rfloor$ *V V F*)
⟨*proof*⟩

lemma *quote-all-Disj-I1-PfP-ssubst*:
assumes *V* $\subseteq$ *Vs supp* $\alpha \subseteq$ *atom* ‘ *Vs supp* $\beta \subseteq$ *atom* ‘ *Vs*
and *prems*: *H* $\vdash$ *PfP* (*ssubst* $\lfloor\alpha\rfloor$ *V V F*) *quote-all p V* $\subseteq$ *H*
shows *H* $\vdash$ *PfP* (*ssubst* $\lfloor\alpha$ *OR* $\beta\rfloor$ *V V F*)
⟨*proof*⟩

lemma *quote-all-Disj-I2-PfP-ssubst*:
assumes *V* $\subseteq$ *Vs supp* $\alpha \subseteq$ *atom* ‘ *Vs supp* $\beta \subseteq$ *atom* ‘ *Vs*
and *prems*: *H* $\vdash$ *PfP* (*ssubst* $\lfloor\beta\rfloor$ *V V F*) *quote-all p V* $\subseteq$ *H*
shows *H* $\vdash$ *PfP* (*ssubst* $\lfloor\alpha$ *OR* $\beta\rfloor$ *V V F*)
⟨*proof*⟩

lemma *quote-all-Conj-I-PfP-ssubst*:
assumes *V* $\subseteq$ *Vs supp* $\alpha \subseteq$ *atom* ‘ *Vs supp* $\beta \subseteq$ *atom* ‘ *Vs*
and *prems*: *H* $\vdash$ *PfP* (*ssubst* $\lfloor\alpha\rfloor$ *V V F*) *H* $\vdash$ *PfP* (*ssubst* $\lfloor\beta\rfloor$ *V V F*) *quote-all p V* $\subseteq$ *H*
shows *H* $\vdash$ *PfP* (*ssubst* $\lfloor\alpha$ *AND* $\beta\rfloor$ *V V F*)
⟨*proof*⟩

lemma *quote-all-Contra-PfP-ssubst*:
assumes *V* $\subseteq$ *Vs supp* $\alpha \subseteq$ *atom* ‘ *Vs*
shows *quote-all p V*
$\vdash$ *PfP* (*ssubst* $\lfloor\alpha\rfloor$ *V V F*) *IMP PfP* (*ssubst* $\lfloor$*Neg* $\alpha\rfloor$ *V V F*) *IMP PfP* (*ssubst* $\lfloor$*Fls*$\rfloor$ *V V F*)
⟨*proof*⟩

lemma *fresh-ssubst-dbtm*: $[\![$*atom i* $\sharp$ *p*$\cdot$*V*; *V* $\subseteq$ *Vs*$]\!] \Longrightarrow$ *atom i* $\sharp$ *ssubst* (*vquot-dbtm V t*) *V F*

⟨*proof*⟩

lemma *fresh-ssubst-dbfm*: ⟦*atom i* ♯ *p*·*V*; *V* ⊆ *Vs*⟧ ⟹ *atom i* ♯ *ssubst* (*vquot-dbfm V A*) *V F*
⟨*proof*⟩

lemma *fresh-ssubst-fm*:
fixes *A*::*fm* **shows** ⟦*atom i* ♯ *p*·*V*; *V* ⊆ *Vs*⟧ ⟹ *atom i* ♯ *ssubst* (⌊*A*⌋ *V*) *V F*
⟨*proof*⟩

end

11.4 Star Property. Equality and Membership: Lemmas 9.3 and 9.4

lemma *SeqQuoteP-Mem-imp-QMem-and-Subset*:
assumes *atom i* ♯ (*j*,*j'*,*i'*,*si*,*ki*,*sj*,*kj*) *atom i'* ♯ (*j*,*j'*,*si*,*ki*,*sj*,*kj*)
atom j ♯ (*j'*,*si*,*ki*,*sj*,*kj*) *atom j'* ♯ (*si*,*ki*,*sj*,*kj*)
atom si ♯ (*ki*,*sj*,*kj*) *atom sj* ♯ (*ki*,*kj*)
shows {*SeqQuoteP* (*Var i*) (*Var i'*) (*Var si*) *ki*, *SeqQuoteP* (*Var j*) (*Var j'*) (*Var sj*) *kj*}
⊢ (*Var i IN Var j IMP PfP* (*Q-Mem* (*Var i'*) (*Var j'*))) *AND*
(*Var i SUBS Var j IMP PfP* (*Q-Subset* (*Var i'*) (*Var j'*)))
⟨*proof*⟩

lemma
assumes *atom i* ♯ (*j*,*j'*,*i'*) *atom i'* ♯ (*j*,*j'*) *atom j* ♯ (*j'*)
shows *QuoteP-Mem-imp-QMem*:
{*QuoteP* (*Var i*) (*Var i'*), *QuoteP* (*Var j*) (*Var j'*), *Var i IN Var j*}
⊢ *PfP* (*Q-Mem* (*Var i'*) (*Var j'*)) (**is** *?thesis1*)
and *QuoteP-Mem-imp-QSubset*:
{*QuoteP* (*Var i*) (*Var i'*), *QuoteP* (*Var j*) (*Var j'*), *Var i SUBS Var j*}
⊢ *PfP* (*Q-Subset* (*Var i'*) (*Var j'*)) (**is** *?thesis2*)
⟨*proof*⟩

11.5 Star Property. Universal Quantifier: Lemma 9.7

lemma (**in** *quote-perm*) *SeqQuoteP-Mem-imp-All2*:
assumes *IH*: *insert* (*QuoteP* (*Var i*) (*Var i'*)) (*quote-all p Vs*)
⊢ *α IMP PfP* (*ssubst* ⌊*α*⌋(*insert i Vs*) (*insert i Vs*) *Fi*)
and *sp*: *supp α* − {*atom i*} ⊆ *atom* ‘ *Vs*
and *j*: *j* ∈ *Vs* **and** *j'*: *p* · *j* = *j'*
and *pi*: *pi* = (*atom i* ⇌ *atom i'*) + *p*
and *Fi*: *Fi* = *make-F* (*insert i Vs*) *pi*
and *atoms*: *atom i* ♯ (*j*,*j'*,*s*,*k*,*p*) *atom i'* ♯ (*i*,*p*,*α*)

atom j ♯ (*j′*,*s*,*k*,*α*) *atom j′* ♯ (*s*,*k*,*α*)
atom s ♯ (*k*,*α*) *atom k* ♯ (*α*,*p*)
shows *insert* (*SeqQuoteP* (*Var j*) (*Var j′*) (*Var s*) (*Var k*)) (*quote-all p* (*Vs*−{*j*}))
⊢ *All2 i* (*Var j*) *α IMP PfP* (*ssubst* ⌊*All2 i* (*Var j*) *α*⌋ *Vs Vs F*)
⟨*proof*⟩

lemma (**in** *quote-perm*) *quote-all-Mem-imp-All2*:
assumes *IH*: *insert* (*QuoteP* (*Var i*) (*Var i′*)) (*quote-all p Vs*)
⊢ *α IMP PfP* (*ssubst* ⌊*α*⌋(*insert i Vs*) (*insert i Vs*) *Fi*)
and *supp* (*All2 i* (*Var j*) *α*) ⊆ *atom* ‘ *Vs*
and *j*: *atom j* ♯ (*i*,*α*) **and** *i*: *atom i* ♯ *p* **and** *i′*: *atom i′* ♯ (*i*,*p*,*α*)
and *pi*: *pi* = (*atom i* ⇌ *atom i′*) + *p*
and *Fi*: *Fi* = *make-F* (*insert i Vs*) *pi*
shows *insert* (*All2 i* (*Var j*) *α*) (*quote-all p Vs*) ⊢ *PfP* (*ssubst* ⌊*All2 i* (*Var j*) *α*⌋ *Vs Vs F*)
⟨*proof*⟩

11.6 The Derivability Condition, Theorem 9.1

lemma *SpecI*: *H* ⊢ *A IMP Ex i A*
⟨*proof*⟩

lemma *star*:
fixes *p* :: *perm* **and** *F* :: *name* ⇒ *tm*
assumes *C*: *ss-fm α*
and *p*: *atom* ‘ (*p* ∙ *V*) ♯* *V* −*p* = *p*
and *V*: *finite V supp α* ⊆ *atom* ‘ *V*
and *F*: *F* = *make-F V p*
shows *insert α* (*quote-all p V*) ⊢ *PfP* (*ssubst* ⌊*α*⌋ *V V F*)
⟨*proof*⟩

theorem *Provability*:
assumes *Sigma-fm α ground-fm α*
shows {*α*} ⊢ *PfP* «*α*»
⟨*proof*⟩

end

Kapitel 12

Gödel's Second Incompleteness Theorem

theory *Goedel-II*
imports *Goedel-I Quote*
begin

The connection between *Quote* and *HR* (for interest only).

lemma *Quote-q-Eats* [*intro*]:
Quote y y' $\Longrightarrow$ *Quote z z'* $\Longrightarrow$ *Quote* (*y* $\triangleleft$ *z*) (*q-Eats y' z'*)
⟨*proof*⟩

lemma *Quote-q-Succ* [*intro*]: *Quote y y'* $\Longrightarrow$ *Quote* (*succ y*) (*q-Succ y'*)
⟨*proof*⟩

lemma *HR-imp-eq-H*: *HR x z* $\Longrightarrow$ *z* = $[\![$*HF x*$]\!]e$
⟨*proof*⟩

lemma *HR-Ord-D*: *HR x y* $\Longrightarrow$ *Ord x* $\Longrightarrow$ *WR x y*
⟨*proof*⟩

lemma *WR-Quote*: *WR* (*ord-of i*) *y* $\Longrightarrow$ *Quote* (*ord-of i*) *y*
⟨*proof*⟩

lemma [*simp*]: ⟨⟨*0*,*0*,*0*⟩, *x*, *y*⟩ = *q-Eats x y*
⟨*proof*⟩

lemma *HR-imp-Quote*: *coding-hf x* $\Longrightarrow$ *HR x y* $\Longrightarrow$ *Quote x y*
⟨*proof*⟩

interpretation *qp0*: *quote-perm 0* {} *make-F* {} *0*
⟨*proof*⟩

lemma *MonPon-PfP-implies-PfP*:

$[\![\{\} \vdash \alpha \ IMP \ \beta;\ ground\text{-}fm\ \alpha;\ ground\text{-}fm\ \beta]\!] \Longrightarrow \{PfP\ «\alpha»\} \vdash PfP\ «\beta»$
⟨*proof*⟩

lemma *PfP-quot-contra*: *ground-fm* $\alpha \Longrightarrow \{\} \vdash PfP\ «\alpha»\ IMP\ PfP\ «Neg\ \alpha»\ IMP\ PfP\ «Fls»$
⟨*proof*⟩

Gödel's second incompleteness theorem: Our theory cannot prove its own consistency.

theorem *Goedel-II*: $\neg\ \{\} \vdash Neg\ (PfP\ «Fls»)$
⟨*proof*⟩

end

Literaturverzeichnis

[1] G. S. Boolos. *The Logic of Provability*. Cambridge University Press, 1993.

[2] S. Feferman et al., editors. *Kurt Gödel: Collected Works*, volume I. Oxford University Press, 1986.

[3] 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.