From Abstract to Concrete Gödel's Incompleteness Theorems—Part II

Andrei Popescu Dmitriy Traytel

May 26, 2024

Abstract

We validate an abstract formulation of Gödels Second Incompleteness Theorem from a separate AFP entry by instantiating it to the case of *finite consistent extensions of the Hereditarily Finite (HF) Set theory*, i.e., consistent FOL theories extending the HF Set theory with a finite set of axioms.

The instantiation draws heavily on infrastructure previously developed by Larry Paulson in his direct formalisation of the concrete result. It strengthens Paulsons formalization of Gödel's Second from that entry by *not* assuming soundness, and in fact not relying on any notion of model or semantic interpretation. The strengthening was obtained by first replacing some of Paulsons semantic arguments with proofs within his HF calculus, and then plugging in some of Paulson's (modified) lemmas to instantiate our soundness-free Gödel's Second locale.

Contents

Chapter 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*) = *Neg* (*A*(*i*::=*x*))
| *Ex*: *atom j* ♯ (*i*, *x*) ⟹ (*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* ♯ *A* ⟹ *A*(*a*::=*x*) = *A*
⟨*proof*⟩

lemma *subst_fm_id* [*simp*]: *A*(*a*::=*Var a*) = *A*
⟨*proof*⟩

lemma *fresh_subst_fm_if* [*simp*]:
j ♯ (*A*(*i*::=*x*)) ⟷ (*atom i* ♯ *A* ∧ *j* ♯ *A*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *A* ∨ *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 Derived syntax

Ordered pairs

definition $HPair :: tm \Rightarrow tm \Rightarrow tm$
where $HPair\ a\ b = Eats\ (Eats\ Zero\ (Eats\ (Eats\ Zero\ b)\ a))\ (Eats\ (Eats\ Zero\ a)\ a)$

lemma *HPair_eqvt* [*eqvt*]: $(p \cdot HPair\ a\ b) = HPair\ (p \cdot a)\ (p \cdot b)$
⟨*proof*⟩

lemma *fresh_HPair* [*simp*]: $x \sharp HPair\ a\ b \longleftrightarrow (x \sharp a \wedge x \sharp b)$
⟨*proof*⟩

lemma *HPair_injective_iff* [*iff*]: $HPair\ a\ b = HPair\ a'\ b' \longleftrightarrow (a = a' \wedge b = b')$
⟨*proof*⟩

lemma *subst_tm_HPair* [*simp*]: $subst\ i\ x\ (HPair\ a\ b) = HPair\ (subst\ i\ x\ a)\ (subst\ i\ x\ b)$
⟨*proof*⟩

Ordinals

definition
$SUCC :: tm \Rightarrow tm$ **where**
$SUCC\ x \equiv Eats\ x\ x$

fun $ORD\_OF :: nat \Rightarrow tm$
where
$ORD\_OF\ 0 = Zero$
| $ORD\_OF\ (Suc\ k) = SUCC\ (ORD\_OF\ k)$

lemma *SUCC_fresh_iff* [*simp*]: $a \sharp SUCC\ t \longleftrightarrow a \sharp t$
⟨*proof*⟩

lemma *SUCC_eqvt* [*eqvt*]: $(p \cdot SUCC\ a) = SUCC\ (p \cdot a)$
⟨*proof*⟩

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

lemma *ORD_OF_fresh* [*simp*]: $a \sharp ORD\_OF\ n$
⟨*proof*⟩

lemma *ORD_OF_eqvt* [*eqvt*]: $(p \cdot ORD\_OF\ n) = ORD\_OF\ (p \cdot n)$
⟨*proof*⟩

1.1.5 Derived logical connectives

abbreviation *Imp* :: *fm* ⇒ *fm* ⇒ *fm* (**infixr** *IMP 125*)
where *Imp A B* ≡ *Disj* (*Neg A*) *B*

abbreviation *All* :: *name* ⇒ *fm* ⇒ *fm*
where *All i A* ≡ *Neg* (*Ex i* (*Neg A*))

abbreviation *All2* :: *name* ⇒ *tm* ⇒ *fm* ⇒ *fm* — bounded universal quantifier, for Sigma formulas
where *All2 i t A* ≡ *All i* ((*Var i IN t*) *IMP A*)

Conjunction

definition *Conj* :: *fm* ⇒ *fm* ⇒ *fm* (**infixr** *AND 135*)
where *Conj A B* ≡ *Neg* (*Disj* (*Neg A*) (*Neg B*))

lemma *Conj_eqvt* [*eqvt*]: *p* ∙ (*A AND B*) = (*p* ∙ *A*) *AND* (*p* ∙ *B*)
⟨*proof*⟩

lemma *fresh_Conj* [*simp*]: *a* ♯ *A AND B* ⟷ (*a* ♯ *A* ∧ *a* ♯ *B*)
⟨*proof*⟩

lemma *supp_Conj* [*simp*]: *supp* (*A AND B*) = *supp A* ∪ *supp B*
⟨*proof*⟩

lemma *size_Conj* [*simp*]: *size* (*A AND B*) = *size A* + *size B* + *4*
⟨*proof*⟩

lemma *Conj_injective_iff* [*iff*]: (*A AND B*) = (*A′ AND B′*) ⟷ (*A* = *A′* ∧ *B* = *B′*)
⟨*proof*⟩

lemma *subst_fm_Conj* [*simp*]: (*A AND B*)(*i*::=*x*) = (*A*(*i*::=*x*)) *AND* (*B*(*i*::=*x*))
⟨*proof*⟩

If and only if

definition *Iff* :: *fm* ⇒ *fm* ⇒ *fm* (**infixr** *IFF 125*)
where *Iff A B* = *Conj* (*Imp A B*) (*Imp B A*)

lemma *Iff_eqvt* [*eqvt*]: *p* ∙ (*A IFF B*) = (*p* ∙ *A*) *IFF* (*p* ∙ *B*)
⟨*proof*⟩

lemma *fresh_Iff* [*simp*]: *a* ♯ *A IFF B* ⟷ (*a* ♯ *A* ∧ *a* ♯ *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′*) ⟷ (*A* = *A′* ∧ *B* = *B′*)
⟨*proof*⟩

lemma *subst_fm_Iff* [*simp*]: (*A IFF B*)(*i*::=*x*) = (*A*(*i*::=*x*)) *IFF* (*B*(*i*::=*x*))
⟨*proof*⟩

1.2 Axioms and Theorems

1.2.1 Logical axioms

inductive_set *boolean_axioms* :: *fm set*

where
  Ident: $A\ IMP\ A \in boolean\_axioms$
| *DisjI1*: $A\ IMP\ (A\ OR\ B) \in boolean\_axioms$
| *DisjCont*: $(A\ OR\ A)\ IMP\ A \in boolean\_axioms$
| *DisjAssoc*: $(A\ OR\ (B\ OR\ C))\ IMP\ ((A\ OR\ B)\ OR\ C) \in boolean\_axioms$
| *DisjConj*: $(C\ OR\ A)\ IMP\ (((Neg\ C)\ OR\ B)\ IMP\ (A\ OR\ B)) \in boolean\_axioms$

inductive_set *special_axioms* :: *fm set* **where**
  I: $A(i::=x)\ IMP\ (Ex\ i\ A) \in special\_axioms$

inductive_set *induction_axioms* :: *fm set* **where**
  ind:
  atom $(j::name) \sharp (i,A)$
  $\Longrightarrow A(i::=Zero)\ IMP\ ((All\ i\ (All\ j\ (A\ IMP\ (A(i::=\ Var\ j)\ IMP\ A(i::=\ Eats(Var\ i)(Var\ j))))))$
    $IMP\ (All\ i\ A))$
  $\in induction\_axioms$

1.2.2 Concrete variables

declare *Abs_name_inject*[*simp*]

abbreviation
  $X0 \equiv Abs\_name\ (Atom\ (Sort\ ''SyntaxN.name''\ [])\ 0)$

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

abbreviation
  $X2 \equiv Abs\_name\ (Atom\ (Sort\ ''SyntaxN.name''\ [])\ 2)$

abbreviation
  $X3 \equiv Abs\_name\ (Atom\ (Sort\ ''SyntaxN.name''\ [])\ 3)$

abbreviation
  $X4 \equiv Abs\_name\ (Atom\ (Sort\ ''SyntaxN.name''\ [])\ 4)$

1.2.3 The HF axioms

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

definition *HF2* :: *fm* **where** — the axiom $(z = x \triangleleft y) = (\forall u.\ (u \in z) = (u \in x \vee u = y))$
  $HF2 \equiv Var\ X0\ EQ\ Eats\ (Var\ X1)\ (Var\ X2)\ IFF$
      $All\ X3\ (Var\ X3\ IN\ Var\ X0\ IFF\ Var\ X3\ IN\ Var\ X1\ OR\ Var\ X3\ EQ\ Var\ X2)$

definition *HF_axioms* **where** $HF\_axioms = \{HF1,\ HF2\}$

1.2.4 Equality axioms

definition *refl_ax* :: *fm* **where**
  $refl\_ax = Var\ X1\ EQ\ Var\ X1$

definition *eq_cong_ax* :: *fm* **where**
  $eq\_cong\_ax = ((Var\ X1\ EQ\ Var\ X2)\ AND\ (Var\ X3\ EQ\ Var\ X4))\ IMP$
      $((Var\ X1\ EQ\ Var\ X3)\ IMP\ (Var\ X2\ EQ\ Var\ X4))$

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*))

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*)))

definition *equality_axioms* :: *fm set* **where**
equality_axioms = {*refl_ax*, *eq_cong_ax*, *mem_cong_ax*, *eats_cong_ax*}

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*

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

1.2.6 Derived rules of inference

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

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

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

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

lemma *rotate2*: $insert\ A2\ (insert\ A1\ H) \vdash B \Longrightarrow insert\ A1\ (insert\ A2\ H) \vdash B$

⟨*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 ⊢ B OR A ⟹ H ⊢ A OR B*
⟨*proof*⟩

lemma *S*: **assumes** *H ⊢ A IMP (B IMP C) H′ ⊢ A IMP B* **shows** *H ∪ H′ ⊢ A IMP C*
⟨*proof*⟩

lemma *Assume*: *insert A H ⊢ 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 ⊢ B ⟹ H ⊢ A IMP B*
⟨*proof*⟩

lemma *DisjAssoc1*: *H ⊢ A OR (B OR C) ⟹ H ⊢ (A OR B) OR C*
⟨*proof*⟩

lemma *DisjAssoc2*: *H ⊢ (A OR B) OR C ⟹ H ⊢ A OR (B OR C)*
⟨*proof*⟩

lemma *Disj_commute_Imp*: *H ⊢ (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*: $H \vdash A \Longrightarrow insert\ A\ H \vdash B \Longrightarrow H \vdash B$
⟨*proof*⟩

lemma *cut_thin*: $HA \vdash A \Longrightarrow insert\ A\ HB \vdash B \Longrightarrow HA \cup HB \subseteq H \Longrightarrow H \vdash B$
⟨*proof*⟩

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

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

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

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

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

lemma *Disj_I2*: $H \vdash B \Longrightarrow H \vdash A\ OR\ B$
⟨*proof*⟩

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

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

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

lemma *NegNeg_I*: $H \vdash A \Longrightarrow H \vdash Neg\ (Neg\ A)$
⟨*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* ⊢ *A*(*i*::=*t*) *IFF A*(*i*::=*u*)
⟨*proof*⟩

lemma *Var_Eq_subst_Iff*: *insert* (*Var i EQ t*) *H* ⊢ *A*(*i*::=*t*) *IFF A*
⟨*proof*⟩

lemma *Var_Eq_imp_subst_Iff*: *H* ⊢ *Var i EQ t* ⟹ *H* ⊢ *A*(*i*::=*t*) *IFF A*
⟨*proof*⟩

1.4.5 Congruence Rules for Predicates

lemma *P1_cong*:
fixes *tms* :: *tm list*
assumes ⋀*i t x*. *atom i* ♯ *tms* ⟹ (*P t*)(*i*::=*x*) = *P* (*subst i x t*) **and** *H* ⊢ *x EQ x′*
shows *H* ⊢ *P x IFF P x′*
⟨*proof*⟩

lemma *P2_cong*:
fixes *tms* :: *tm list*
assumes *sub*: ⋀*i t u x*. *atom i* ♯ *tms* ⟹ (*P t u*)(*i*::=*x*) = *P* (*subst i x t*) (*subst i x u*)
and *eq*: *H* ⊢ *x EQ x′ H* ⊢ *y EQ y′*
shows *H* ⊢ *P x y IFF P x′ y′*
⟨*proof*⟩

lemma *P3_cong*:
fixes *tms* :: *tm list*
assumes *sub*: ⋀*i t u v x*. *atom i* ♯ *tms* ⟹
(*P t u v*)(*i*::=*x*) = *P* (*subst i x t*) (*subst i x u*) (*subst i x v*)
and *eq*: *H* ⊢ *x EQ x′ H* ⊢ *y EQ y′ H* ⊢ *z EQ z′*
shows *H* ⊢ *P x y z IFF P x′ y′ z′*
⟨*proof*⟩

lemma *P4_cong*:
fixes *tms* :: *tm list*
assumes *sub*: ⋀*i t1 t2 t3 t4 x*. *atom i* ♯ *tms* ⟹
(*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* ⊢ *x1 EQ x1′ H* ⊢ *x2 EQ x2′ H* ⊢ *x3 EQ x3′ H* ⊢ *x4 EQ x4′*
shows *H* ⊢ *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* ⊢ (*t EQ Zero*) *IFF* (*All i* (*Neg* ((*Var i*) *IN t*)))
⟨*proof*⟩

lemma *Mem_Zero_E* [*intro*!]: *insert* (*x IN Zero*) *H* ⊢ *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*

definition *Fls* **where** *Fls* ≡ *Zero IN Zero*

lemma *Fls_eqvt* [*eqvt*]: (*p* · *Fls*) = *Fls*
⟨*proof*⟩

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

lemma *Neg_I* [*intro*!]: *insert A H* ⊢ *Fls* ⟹ *H* ⊢ *Neg A*
⟨*proof*⟩

lemma *Neg_E* [*intro*!]: *H* ⊢ *A* ⟹ *insert* (*Neg A*) *H* ⊢ *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* ⊢ *A* ⟹ *insert B H* ⊢ *Fls* ⟹ *H* ⊢ *Neg* (*A IMP B*)
⟨*proof*⟩

lemma *Neg_Imp_E* [*intro*!]: *insert* (*Neg B*) (*insert A H*) ⊢ *C* ⟹ *insert* (*Neg* (*A IMP B*)) *H* ⊢ *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*⟩

1.5.2 More properties of *Zero*

lemma *Eq_Zero_D*:

assumes $H \vdash t\ EQ\ Zero$ $H \vdash u\ IN\ t$ **shows** $H \vdash A$
⟨*proof*⟩

lemma *Eq_Zero_thm*:
assumes *atom* $i \sharp t$ **shows** $\{All\ i\ (Neg\ ((Var\ i)\ IN\ t))\} \vdash t\ EQ\ Zero$
⟨*proof*⟩

lemma *Eq_Zero_I*:
assumes *insi*: *insert* $((Var\ i)\ IN\ t)$ $H \vdash Fls$ **and** *i1*: *atom* $i \sharp t$ **and** *i2*: $\forall B \in H.\ atom\ i \sharp B$
shows $H \vdash t\ EQ\ Zero$
⟨*proof*⟩

1.5.3 Basic properties of *Eats*

lemma *Eq_Eats_iff*:
assumes *atom* $i \sharp (z,t,u)$
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* ⊢ *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* ⊢ *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* ⊢ *t EQ t′* ⟹ *H* ⊢ *A IFF A′* ⟹ ∀ *C* ∈ *H*. *atom i* ♯ *C* ⟹ *H* ⊢ (*All2 i t A*) *IFF* (*All2 i t′ A′*)
⟨*proof*⟩

lemma *All2_Zero_E* [*intro*!]: *H* ⊢ *B* ⟹ *insert* (*All2 i Zero A*) *H* ⊢ *B*
⟨*proof*⟩

lemma *All2_Eats_I_D*:
atom i ♯ (*t*,*u*) ⟹ { *All2 i t A*, *A*(*i*::=*u*)} ⊢ (*All2 i* (*Eats t u*) *A*)
⟨*proof*⟩

lemma *All2_Eats_I*:
⟦*atom i* ♯ (*t*,*u*); *H* ⊢ *All2 i t A*; *H* ⊢ *A*(*i*::=*u*)⟧ ⟹ *H* ⊢ (*All2 i* (*Eats t u*) *A*)
⟨*proof*⟩

lemma *All2_Eats_E1*:
⟦*atom i* ♯ (*t*,*u*); ∀ *C* ∈ *H*. *atom i* ♯ *C*⟧ ⟹ *insert* (*All2 i* (*Eats t u*) *A*) *H* ⊢ *All2 i t A*
⟨*proof*⟩

lemma *All2_Eats_E2*:
⟦*atom i* ♯ (*t*,*u*); ∀ *C* ∈ *H*. *atom i* ♯ *C*⟧ ⟹ *insert* (*All2 i* (*Eats t u*) *A*) *H* ⊢ *A*(*i*::=*u*)
⟨*proof*⟩

lemma *All2_Eats_E*:
assumes *i*: *atom i* ♯ (*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* $i \sharp t$
and *insert* (*All2* i t A) (*insert* ($A(i::=t)$) H) $\vdash B$
shows *insert* (*All2* i (*SUCC* t) A) $H \vdash B$
⟨*proof*⟩

lemma *All2_SUCC_E'*:
assumes $H \vdash u$ *EQ SUCC* t
and *atom* $i \sharp t \; \forall C \in H$. *atom* $i \sharp C$
and *insert* (*All2* i t A) (*insert* ($A(i::=t)$) H) $\vdash B$
shows *insert* (*All2* i u A) $H \vdash B$
⟨*proof*⟩

1.7 Induction

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

end

Chapter 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 ∈ *set e* ⟹ ∃ *k*. *lookup e n x* = *DBInd k* ∧ *n* ≤ *k* ∧ *k* < *n* + *length e*
⟨*proof*⟩

lemma *lookup_fresh*: *x* ♯ *lookup e n y* ⟷ *y* ∈ *set e* ∨ *x* ≠ *atom y*
  ⟨*proof*⟩

lemma *lookup_eqvt*[*eqvt*]: (*p* ∙ *lookup xs n x*) = *lookup* (*p* ∙ *xs*) (*p* ∙ *n*) (*p* ∙ *x*)

⟨*proof*⟩

lemma *lookup_inject* [*iff*]: (*lookup e n x* = *lookup e n y*) ⟷ *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* ♯ *trans_tm e t* ⟷ *i* ♯ *t* ∨ *i* ∈ *atom* ‘ *set e*
⟨*proof*⟩

lemma *trans_tm_forget*: *atom i* ♯ *t* ⟹ *trans_tm* [*i*] *t* = *trans_tm* [] *t*
⟨*proof*⟩

nominal_function (*invariant* λ(*xs*, _) *y*. *atom* ‘ *set xs* ♯* *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* ♯ *e* ⟹ *trans_fm e* (*Ex k A*) = *DBEx* (*trans_fm* (*k*#*e*) *A*)
⟨*proof*⟩

nominal_termination (*eqvt*)
⟨*proof*⟩

lemma *fresh_trans_fm* [*simp*]: *i* ♯ *trans_fm e A* ⟷ *i* ♯ *A* ∨ *i* ∈ *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 Characterising the Well-Formed de Bruijn Formulas

2.2.1 Well-Formed Terms

inductive *wf_dbtm* :: *dbtm* ⇒ *bool*
  where
    Zero: *wf_dbtm DBZero*
  | *Var*: *wf_dbtm* (*DBVar name*)
  | *Eats*: *wf_dbtm t1* ⟹ *wf_dbtm t2* ⟹ *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 ∃ *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* ⟷ (∃ *t*::*tm*. *x* = *trans_tm* [] *t*)
  ⟨*proof*⟩

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 i DBZero = *DBZero*
  | *subst_dbtm u i* (*DBVar name*) = (*if i* = *name then u else DBVar name*)
  | *subst_dbtm u i* (*DBInd j*) = *DBInd j*
  | *subst_dbtm u i* (*DBEats t1 t2*) = *DBEats* (*subst_dbtm u i t1*) (*subst_dbtm u i 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*⟩

2.2.2 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 i (*DBMem t1 t2*) = *DBMem* (*subst_dbtm u i t1*) (*subst_dbtm u i t2*)
| *subst_dbfm u i* (*DBEq t1 t2*) = *DBEq* (*subst_dbtm u i t1*) (*subst_dbtm u i t2*)
| *subst_dbfm u i* (*DBDisj A1 A2*) = *DBDisj* (*subst_dbfm u i A1*) (*subst_dbfm u i A2*)
| *subst_dbfm u i* (*DBNeg A*) = *DBNeg* (*subst_dbfm u i A*)
| *subst_dbfm u i* (*DBEx A*) = *DBEx* (*subst_dbfm u i 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)

inductive *wf_dbfm* :: *dbfm* ⇒ *bool*
where
Mem: *wf_dbtm t1* ⟹ *wf_dbtm t2* ⟹ *wf_dbfm* (*DBMem t1 t2*)
| *Eq*: *wf_dbtm t1* ⟹ *wf_dbtm t2* ⟹ *wf_dbfm* (*DBEq t1 t2*)
| *Disj*: *wf_dbfm A1* ⟹ *wf_dbfm A2* ⟹ *wf_dbfm* (*DBDisj A1 A2*)
| *Neg*: *wf_dbfm A* ⟹ *wf_dbfm* (*DBNeg A*)
| *Ex*: *wf_dbfm A* ⟹ *wf_dbfm* (*DBEx* (*abst_dbfm name 0 A*))

equivariance *wf_dbfm*

lemma *atom_fresh_abst_dbtm* [*simp*]: *atom i* ♯ *abst_dbtm i n t*
⟨*proof*⟩

lemma *atom_fresh_abst_dbfm* [*simp*]: *atom i* ♯ *abst_dbfm i n A*
⟨*proof*⟩

Setting up strong induction: "avoiding" 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* ⇒ *tm* **where**
  HTuple 0 = *HPair Zero Zero*
| *HTuple* (*Suc k*) = *HPair Zero* (*HTuple k*)

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

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

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

lemmas *HTS* = *HTuple_minus_1 HTuple.simps* — for freeness reasoning on codes

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_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*⟩

instantiation *fm* :: *quot*
begin
definition *quot_fm* :: $fm \Rightarrow tm$
where *quot_fm A* = *quot_dbfm* (*trans_fm* [] *A*)

instance ⟨*proof*⟩
end

lemma *quot_dbfm_fresh* [*simp*]: $s \sharp$ (*quot_dbfm A*)
⟨*proof*⟩

lemma *quot_fm_fresh* [*simp*]: **fixes** *A*::*fm* **shows** $s \sharp$ «*A*»
⟨*proof*⟩

lemma *quot_fm_permute* [*simp*]: **fixes** *A*:: *fm* **shows** $p \cdot$ «*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*⟩

lemmas *quot_simps* = *quot_Var quot_Eats quot_Eq quot_Mem quot_Disj quot_Neg quot_Ex*

2.5 Definitions Involving Coding

abbreviation *Q_Eats* :: *tm* ⇒ *tm* ⇒ *tm*
where *Q_Eats t u* ≡ *HPair* (*HTuple* (*Suc 0*)) (*HPair t u*)

abbreviation *Q_Succ* :: *tm* ⇒ *tm*
where *Q_Succ t* ≡ *Q_Eats t t*

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*)

abbreviation *Q_Mem* :: *tm* ⇒ *tm* ⇒ *tm*
where *Q_Mem t u* ≡ *HPair* (*HTuple 0*) (*HPair t u*)

abbreviation *Q_Eq* :: *tm* ⇒ *tm* ⇒ *tm*
where *Q_Eq t u* ≡ *HPair* (*HTuple 2*) (*HPair t u*)

abbreviation *Q_Disj* :: *tm* ⇒ *tm* ⇒ *tm*
where *Q_Disj t u* ≡ *HPair* (*HTuple 3*) (*HPair t u*)

abbreviation *Q_Neg* :: *tm* ⇒ *tm*
where *Q_Neg t* ≡ *HPair* (*HTuple 4*) *t*

abbreviation *Q_Conj* :: *tm* ⇒ *tm* ⇒ *tm*
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*

abbreviation *Q_Ex* :: *tm* ⇒ *tm*
where *Q_Ex t* ≡ *HPair* (*HTuple 5*) *t*

abbreviation *Q_All* :: *tm* ⇒ *tm*
where *Q_All t* ≡ *Q_Neg* (*Q_Ex* (*Q_Neg t*))

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*⟩

2.5.1 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*⟩

2.6 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*]: $i \sharp \mathit{vquot\_dbfm}\ V\ \mathit{fm} \longleftrightarrow i \sharp \mathit{fm} \vee i \notin \mathit{atom}\ `\ V$
⟨*proof*⟩

class *vquot* =
fixes *vquot* :: $'a \Rightarrow \mathit{name\ set} \Rightarrow \mathit{tm}$ (⌊_⌋_ [0,1000]1000)

instantiation *tm* :: *vquot*
begin
definition *vquot_tm* :: $\mathit{tm} \Rightarrow \mathit{name\ set} \Rightarrow \mathit{tm}$
where $\mathit{vquot\_tm}\ t\ V = \mathit{vquot\_dbtm}\ V\ (\mathit{trans\_tm}\ []\ t)$
instance ⟨*proof*⟩
end

lemma *vquot_dbtm_empty* [*simp*]: $\mathit{vquot\_dbtm}\ \{\}\ t = \mathit{quot\_dbtm}\ t$
⟨*proof*⟩

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

lemma *vquot_dbtm_eq*: $\mathit{atom}\ `\ V \cap \mathit{supp}\ t = \mathit{atom}\ `\ W \cap \mathit{supp}\ t \Longrightarrow \mathit{vquot\_dbtm}\ V\ t = \mathit{vquot\_dbtm}\ W\ t$
⟨*proof*⟩

instantiation *fm* :: *vquot*
begin
definition *vquot_fm* :: $\mathit{fm} \Rightarrow \mathit{name\ set} \Rightarrow \mathit{tm}$
where $\mathit{vquot\_fm}\ A\ V = \mathit{vquot\_dbfm}\ V\ (\mathit{trans\_fm}\ []\ A)$
instance ⟨*proof*⟩
end

lemma *vquot_fm_fresh* [*simp*]: **fixes** *A*::*fm* **shows** $i \sharp ⌊A⌋ V \longleftrightarrow i \sharp A \vee i \notin \mathit{atom}\ `\ V$
⟨*proof*⟩

lemma *vquot_dbfm_empty* [*simp*]: $\mathit{vquot\_dbfm}\ \{\}\ A = \mathit{quot\_dbfm}\ A$
⟨*proof*⟩

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

lemma *vquot_dbfm_eq*: $\mathit{atom}\ `\ V \cap \mathit{supp}\ A = \mathit{atom}\ `\ W \cap \mathit{supp}\ A \Longrightarrow \mathit{vquot\_dbfm}\ V\ A = \mathit{vquot\_dbfm}\ W\ A$
⟨*proof*⟩

lemma *vquot_fm_insert*:
fixes *A*::*fm* **shows** $\mathit{atom}\ i \notin \mathit{supp}\ A \Longrightarrow ⌊A⌋(\mathit{insert}\ i\ V) = ⌊A⌋ V$
⟨*proof*⟩

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

end

Chapter 3

Basic Predicates

theory *Predicates*
imports *SyntaxN*
begin

3.1 The Subset Relation

nominal_function *Subset* :: *tm* ⇒ *tm* ⇒ *fm* (**infixr** *SUBS 150*)
where *atom z* ♯ (*t*, *u*) ⟹ *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* ♯ *t SUBS u* ⟷ *a* ♯ *t* ∧ *a* ♯ *u*
⟨*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* ⊢ (*Var i*) *IN u atom i* ♯ (*t*,*u*) ∀ *B* ∈ *H*. *atom i* ♯ *B*
shows *H* ⊢ *t SUBS u*
⟨*proof*⟩

lemma *Subset_D*:
assumes *major*: *H* ⊢ *t SUBS u* **and** *minor*: *H* ⊢ *a IN t* **shows** *H* ⊢ *a IN u*
⟨*proof*⟩

lemma *Subset_E*: *H* ⊢ *t SUBS u* ⟹ *H* ⊢ *a IN t* ⟹ *insert* (*a IN u*) *H* ⊢ *A* ⟹ *H* ⊢ *A*
⟨*proof*⟩

lemma *Subset_cong*: *H* ⊢ *t EQ t′* ⟹ *H* ⊢ *u EQ u′* ⟹ *H* ⊢ *t SUBS u IFF t′ SUBS u′*
⟨*proof*⟩

lemma *Set_MP*: *x SUBS y* ∈ *H* ⟹ *z IN x* ∈ *H* ⟹ *insert* (*z IN y*) *H* ⊢ *A* ⟹ *H* ⊢ *A*
⟨*proof*⟩

lemma *Zero_Subset_I* [*intro*!]: *H* ⊢ *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 ⊢ 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*: $\{\ a\ SUBS\ b,\ b\ SUBS\ c\ \} \vdash a\ SUBS\ c$
⟨*proof*⟩

lemma *Subset_trans*: $H \vdash a\ SUBS\ b \Longrightarrow H \vdash b\ SUBS\ c \Longrightarrow H \vdash a\ SUBS\ c$
⟨*proof*⟩

lemma *Subset_SUCC*: $H \vdash a\ SUBS\ (SUCC\ a)$
⟨*proof*⟩

lemma *All2_Subset_lemma*: $atom\ l \sharp (k',k) \Longrightarrow \{P\} \vdash P' \Longrightarrow \{All2\ l\ k\ P,\ k'\ SUBS\ k\} \vdash All2\ l\ k'\ P'$
⟨*proof*⟩

lemma *All2_Subset*: $[\![H \vdash All2\ l\ k\ P;\ H \vdash k'\ SUBS\ k;\ \{P\} \vdash P';\ atom\ l \sharp (k',\ k)]\!] \Longrightarrow H \vdash All2\ l\ k'\ P'$
⟨*proof*⟩

3.2 Extensionality

lemma *Extensionality*: $H \vdash x\ EQ\ y\ IFF\ x\ SUBS\ y\ AND\ y\ SUBS\ x$
⟨*proof*⟩

lemma *Equality_I*: $H \vdash y\ SUBS\ x \Longrightarrow H \vdash x\ SUBS\ y \Longrightarrow H \vdash x\ EQ\ y$
⟨*proof*⟩

lemma *EQ_imp_SUBS*: $insert\ (t\ EQ\ u)\ H \vdash (t\ SUBS\ u)$
⟨*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* ♯ *Disjoint t u* ⟷ *a* ♯ *t* ∧ *a* ♯ *u*
⟨*proof*⟩

lemma *subst_fm_Disjoint* [*simp*]:
(*Disjoint t u*)(*i*::=*x*) = *Disjoint* (*subst i x t*) (*subst i x u*)
⟨*proof*⟩

lemma *Disjoint_cong*: *H* ⊢ *t EQ t′* ⟹ *H* ⊢ *u EQ u′* ⟹ *H* ⊢ *Disjoint t u IFF Disjoint t′ u′*
⟨*proof*⟩

lemma *Disjoint_I*:
assumes *insert* ((*Var i*) *IN t*) (*insert* ((*Var i*) *IN u*) *H*) ⊢ *Fls*
atom i ♯ (*t*,*u*) ∀ *B* ∈ *H*. *atom i* ♯ *B*
shows *H* ⊢ *Disjoint t u*
⟨*proof*⟩

lemma *Disjoint_E*:
assumes *major*: *H* ⊢ *Disjoint t u* **and** *minor*: *H* ⊢ *a IN t H* ⊢ *a IN u* **shows** *H* ⊢ *A*
⟨*proof*⟩

lemma *Disjoint_commute*: { *Disjoint t u* } ⊢ *Disjoint u t*
⟨*proof*⟩

lemma *Disjoint_commute_I*: *H* ⊢ *Disjoint t u* ⟹ *H* ⊢ *Disjoint u t*
⟨*proof*⟩

lemma *Disjoint_commute_D*: *insert* (*Disjoint t u*) *H* ⊢ *A* ⟹ *insert* (*Disjoint u t*) *H* ⊢ *A*
⟨*proof*⟩

lemma *Zero_Disjoint_I1* [*iff*]: *H* ⊢ *Disjoint Zero t*
⟨*proof*⟩

lemma *Zero_Disjoint_I2* [*iff*]: *H* ⊢ *Disjoint t Zero*
⟨*proof*⟩

lemma *Disjoint_Eats_D1*: { *Disjoint* (*Eats x y*) *z* } ⊢ *Disjoint x z*
⟨*proof*⟩

lemma *Disjoint_Eats_D2*: { *Disjoint* (*Eats x y*) *z* } ⊢ *Neg*(*y IN z*)

⟨*proof*⟩

lemma *Disjoint_Eats_E*:
insert (*Disjoint x z*) (*insert* (*Neg*(*y IN z*)) *H*) ⊢ *A* ⟹ *insert* (*Disjoint* (*Eats x y*) *z*) *H* ⊢ *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*]: $a \sharp \mathit{OrdP}\ x \longleftrightarrow a \sharp x$ (**is** *?thesis1*)
⟨*proof*⟩

lemma *subst_fm_OrdP* [*simp*]: $(\mathit{OrdP}\ t)(i::=x) = \mathit{OrdP}\ (\mathit{subst}\ i\ x\ t)$
⟨*proof*⟩

lemma *OrdP_cong*: $H \vdash x\ \mathit{EQ}\ x' \Longrightarrow H \vdash \mathit{OrdP}\ x\ \mathit{IFF}\ \mathit{OrdP}\ x'$
⟨*proof*⟩

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

lemma *OrdP_Mem_E*:
assumes $\mathit{atom}\ z \sharp (k,l)$
$\mathit{insert}\ (\mathit{OrdP}\ k)\ H \vdash l\ \mathit{IN}\ k$
$\mathit{insert}\ (l\ \mathit{SUBS}\ k)\ (\mathit{insert}\ (\mathit{All2}\ z\ l\ (\mathit{Var}\ z\ \mathit{SUBS}\ l))\ H) \vdash A$
shows $\mathit{insert}\ (\mathit{OrdP}\ k)\ H \vdash A$
⟨*proof*⟩

lemma *OrdP_Mem_imp_Subset*:
assumes *k*: $H \vdash k\ \mathit{IN}\ l$ **and** *l*: $H \vdash \mathit{OrdP}\ l$ **shows** $H \vdash k\ \mathit{SUBS}\ l$
⟨*proof*⟩

lemma *SUCC_Subset_Ord_lemma*: $\{\ k'\ \mathit{IN}\ k,\ \mathit{OrdP}\ k\ \} \vdash \mathit{SUCC}\ k'\ \mathit{SUBS}\ k$
⟨*proof*⟩

lemma *SUCC_Subset_Ord*: $H \vdash k'\ \mathit{IN}\ k \Longrightarrow H \vdash \mathit{OrdP}\ k \Longrightarrow H \vdash \mathit{SUCC}\ k'\ \mathit{SUBS}\ k$
⟨*proof*⟩

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

lemma *OrdP_Trans*: $H \vdash\ \mathit{OrdP}\ k \Longrightarrow H \vdash i\ \mathit{IN}\ j \Longrightarrow H \vdash j\ \mathit{IN}\ k \Longrightarrow H\ \vdash i\ \mathit{IN}\ k$
⟨*proof*⟩

lemma *Ord_IN_Ord0*:
assumes *l*: $H \vdash l\ \mathit{IN}\ k$
shows $\mathit{insert}\ (\mathit{OrdP}\ k)\ H \vdash \mathit{OrdP}\ l$
⟨*proof*⟩

lemma *Ord_IN_Ord*: $H \vdash l\ \mathit{IN}\ k \Longrightarrow H \vdash \mathit{OrdP}\ k \Longrightarrow H \vdash \mathit{OrdP}\ l$
⟨*proof*⟩

lemma *OrdP_I*:
assumes $\mathit{insert}\ (\mathit{Var}\ y\ \mathit{IN}\ x)\ H \vdash (\mathit{Var}\ y)\ \mathit{SUBS}\ x$
and $\mathit{insert}\ (\mathit{Var}\ z\ \mathit{IN}\ \mathit{Var}\ y)\ (\mathit{insert}\ (\mathit{Var}\ y\ \mathit{IN}\ x)\ H) \vdash (\mathit{Var}\ z)\ \mathit{SUBS}\ (\mathit{Var}\ y)$
and $\mathit{atom}\ y \sharp (x,\ z)\ \forall B \in H.\ \mathit{atom}\ y \sharp B\ \ \mathit{atom}\ z \sharp x\ \forall B \in H.\ \mathit{atom}\ z \sharp B$
shows $H \vdash \mathit{OrdP}\ x$
⟨*proof*⟩

lemma *OrdP_Zero* [*simp*]: $H \vdash \mathit{OrdP}\ \mathit{Zero}$
⟨*proof*⟩

lemma *OrdP_SUCC_I0*: $\{\ \mathit{OrdP}\ k\ \} \vdash \mathit{OrdP}\ (\mathit{SUCC}\ k)$
⟨*proof*⟩

lemma *OrdP_SUCC_I*: $H \vdash OrdP\ k \Longrightarrow H \vdash OrdP\ (SUCC\ k)$
⟨*proof*⟩

lemma *Zero_In_OrdP*: $\{\ OrdP\ x\ \} \vdash x\ EQ\ Zero\ OR\ Zero\ IN\ x$
⟨*proof*⟩

lemma *OrdP_HPairE*: *insert* (*OrdP* (*HPair x y*)) $H \vdash A$
⟨*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*) $H \vdash A$
⟨*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*) $H \vdash A$
⟨*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 $\{\ OrdP\ (Var\ i)\ \} \vdash (All\ i\ (OrdP\ (Var\ i)\ IMP\ ((All2\ j\ (Var\ i)\ (A(i::=\ Var\ j)))\ IMP\ A)))\ IMP\ A$
⟨*proof*⟩

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

lemma *OrdIndH*:
assumes *atom* (*j*::*name*) ♯ (*i*,*A*)
and $H \vdash All\ i\ (OrdP\ (Var\ i)\ IMP\ (All2\ j\ (Var\ i)\ (A(i::=\ Var\ j))\ IMP\ A))$
shows *insert* (*OrdP* (*Var i*)) $H \vdash A$
⟨*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*} ⊢ *Zero IN SUCC k*
⟨*proof*⟩

3.8 The predicate *OrdNotEqP*

nominal_function *OrdNotEqP* :: *tm* ⇒ *tm* ⇒ *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* ♯ *OrdNotEqP x y* ⟷ *a* ♯ *x* ∧ *a* ♯ *y*
⟨*proof*⟩

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

lemma *OrdNotEqP_cong*: *H* ⊢ *x EQ x*′ ⟹ *H* ⊢ *y EQ y*′ ⟹ *H* ⊢ *OrdNotEqP x y IFF OrdNotEqP x*′ *y*′
⟨*proof*⟩

lemma *OrdNotEqP_self_contra*: {*x NEQ x*} ⊢ *Fls*
⟨*proof*⟩

lemma *OrdNotEqP_OrdP_E*: *insert* (*OrdP x*) (*insert* (*OrdP y*) *H*) ⊢ *A* ⟹ *insert* (*x NEQ y*) *H* ⊢ *A*
⟨*proof*⟩

lemma *OrdNotEqP_I*: *insert* (*x EQ y*) *H* ⊢ *Fls* ⟹ *H* ⊢ *OrdP x* ⟹ *H* ⊢ *OrdP y* ⟹ *H* ⊢ *x NEQ y*
⟨*proof*⟩

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

lemma *OrdNotEqP_imp_Neg_Eq*: {*x NEQ y*} ⊢ *Neg* (*x EQ y*)
⟨*proof*⟩

lemma *OrdNotEqP_E*: *H* ⊢ *x EQ y* ⟹ *insert* (*x NEQ y*) *H* ⊢ *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* ⊢ *A*(*i*::= *Zero*)
and *H* ⊢ *All i* (*OrdP* (*Var i*) *IMP A IMP* (*A*(*i*::= *SUCC* (*Var i*))))
shows *insert* (*OrdP* (*Var i*)) *H* ⊢ *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_sigma r ≡ ∀ *z* ∈ *r*. ∀ *z'* ∈ *r*. ∃ *x y x' y'*. *z* = ⟨*x*,*y*⟩ ∧ *z'* = ⟨*x'*,*y'*⟩ ∧ (*x*=*x'* ⟶ *y*=*y'*)

definition *hfun_sigma_ord* **where**

hfun_sigma_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* ⇒ *fm*
  where ⟦*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*) ⟧ ⟹
    HFun_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* ♯ *HFun_Sigma r* ⟷ *a* ♯ *r* (**is** *?thesis1*)
⟨*proof*⟩

lemma *HFun_Sigma_subst* [*simp*]: (*HFun_Sigma r*)(*i*::=*t*) = *HFun_Sigma* (*subst i t r*)
⟨*proof*⟩

lemma *HFun_Sigma_Zero*: *H* ⊢ *HFun_Sigma Zero*
⟨*proof*⟩

lemma *Subset_HFun_Sigma*: {*HFun_Sigma s*, *s' SUBS s*} ⊢ *HFun_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* ⊢ *HFun_Sigma r H* ⊢ *a IN r*
    and *xy*: *atom x* ♯ (*y*,*a*,*r*) *atom y* ♯ (*a*,*r*)
   shows *H* ⊢ (*Ex x* (*Ex y* (*a EQ HPair* (*Var x*) (*Var y*)))) (**is** _ ⊢ *?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*)
⟨*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* } ⊢ *HDomain_Incl r k'*

⟨*proof*⟩

lemma *HDomain_Incl_Subset*: $H \vdash HDomain\_Incl\ r\ k \Longrightarrow H \vdash k'\ SUBS\ k \Longrightarrow H \vdash HDomain\_Incl\ r\ k'$
⟨*proof*⟩

lemma *HDomain_Incl_Mem_Ord*: $H \vdash HDomain\_Incl\ r\ k \Longrightarrow H \vdash k'\ IN\ k \Longrightarrow H \vdash OrdP\ k \Longrightarrow H \vdash HDomain\_Incl\ r\ k'$
⟨*proof*⟩

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

lemma *HDomain_Incl_Eats*: $\{\ HDomain\_Incl\ r\ d\ \} \vdash HDomain\_Incl\ (Eats\ r\ (HPair\ d\ d'))\ (SUCC\ d)$
⟨*proof*⟩

lemma *HDomain_Incl_Eats_I*: $H \vdash HDomain\_Incl\ r\ d \Longrightarrow H \vdash HDomain\_Incl\ (Eats\ r\ (HPair\ d\ d'))\ (SUCC\ d)$
⟨*proof*⟩

3.13 *HPair* is Provably Injective

lemma *Doubleton_E*:
assumes $insert\ (a\ EQ\ c)\ (insert\ (b\ EQ\ d)\ H) \vdash A$
$insert\ (a\ EQ\ d)\ (insert\ (b\ EQ\ c)\ H) \vdash A$
shows $insert\ ((Eats\ (Eats\ Zero\ b)\ a)\ EQ\ (Eats\ (Eats\ Zero\ d)\ c))\ H \vdash A$
⟨*proof*⟩

lemma *HFST*: $\{HPair\ a\ b\ EQ\ HPair\ c\ d\} \vdash a\ EQ\ c$
⟨*proof*⟩

lemma *b_EQ_d_1*: $\{a\ EQ\ c,\ a\ EQ\ d,\ b\ EQ\ c\} \vdash b\ EQ\ d$
⟨*proof*⟩

lemma *HSND*: $\{HPair\ a\ b\ EQ\ HPair\ c\ d\} \vdash b\ EQ\ d$
⟨*proof*⟩

lemma *HPair_E* [*intro*!]:
assumes $insert\ (a\ EQ\ c)\ (insert\ (b\ EQ\ d)\ H) \vdash A$
shows $insert\ (HPair\ a\ b\ EQ\ HPair\ c\ d)\ H \vdash A$
⟨*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*: $H \vdash HFun\_Sigma\ r$
and *b*: $H \vdash HPair\ a\ b\ IN\ r$
and *b'*: $H \vdash HPair\ a\ b'\ IN\ r$
shows $H \vdash b\ EQ\ b'$
⟨*proof*⟩

3.14 *SUCC* is Provably Injective

lemma *SUCC_SUBS_lemma*: $\{SUCC\ x\ SUBS\ SUCC\ y\} \vdash x\ SUBS\ y$
⟨*proof*⟩

lemma *SUCC_SUBS*: $insert\ (SUCC\ x\ SUBS\ SUCC\ y)\ H \vdash x\ SUBS\ y$
⟨*proof*⟩

lemma *SUCC_inject*: $insert\ (SUCC\ x\ EQ\ SUCC\ y)\ H \vdash x\ EQ\ y$
⟨*proof*⟩

lemma *SUCC_inject_E* [*intro*!]: $insert\ (x\ EQ\ y)\ H \vdash A \Longrightarrow insert\ (SUCC\ x\ EQ\ SUCC\ y)\ H \vdash A$
⟨*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*: $\{OrdP\ x,\ y\ IN\ x\} \vdash SUCC\ y\ IN\ SUCC\ x$
⟨*proof*⟩

lemma *OrdP_IN_SUCC*: $H \vdash OrdP\ x \Longrightarrow H \vdash y\ IN\ x \Longrightarrow H \vdash SUCC\ y\ IN\ SUCC\ x$
⟨*proof*⟩

lemma *OrdP_IN_SUCC_D_lemma*: $\{OrdP\ x,\ SUCC\ y\ IN\ SUCC\ x\} \vdash y\ IN\ x$
⟨*proof*⟩

lemma *OrdP_IN_SUCC_D*: $H \vdash OrdP\ x \Longrightarrow H \vdash SUCC\ y\ IN\ SUCC\ x \Longrightarrow H \vdash y\ IN\ x$
⟨*proof*⟩

lemma *OrdP_IN_SUCC_Iff*: $H \vdash OrdP\ y \Longrightarrow H \vdash SUCC\ x\ IN\ SUCC\ y\ IFF\ x\ IN\ y$
⟨*proof*⟩

3.15 The predicate *LstSeqP*

lemma *hfun_sigma_ord_iff*: $hfun\_sigma\_ord\ s \longleftrightarrow OrdDom\ s \wedge hfun\_sigma\ s$
⟨*proof*⟩

lemma *hfun_sigma_iff*: $hfun\_sigma\ r \longleftrightarrow hfunction\ r \wedge hrelation\ r$
⟨*proof*⟩

lemma *Seq_iff*: $Seq\ r\ d \longleftrightarrow d \leq hdomain\ r \wedge hfun\_sigma\ r$
⟨*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*)
⟨*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

Chapter 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 ⋀*k*::*name*. *Sigma_fm* (*Var k EQ u*)
shows *Sigma_fm* (*Var i SUBS Eats t u*)
⟨*proof*⟩

lemma *Eq_Eats_sf*:
assumes ⋀*j*::*name*. *Sigma_fm* (*Var j EQ t*)
and ⋀*k*::*name*. *Sigma_fm* (*Var k EQ u*)
shows *Sigma_fm* (*Var i EQ Eats t u*)
⟨*proof*⟩

lemma *Eats_Mem_sf*:
assumes ⋀*j*::*name*. *Sigma_fm* (*Var j EQ t*)
and ⋀*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* $\Longrightarrow$ *atom j ♯ (i,t,A)* $\Longrightarrow$
{} $\vdash$ (*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
{} $\vdash$*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*⟩

end

Chapter 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 *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 *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* ⊢ *x EQ x′* ⟹ *H* ⊢ *VarP x IFF VarP x′*
⟨*proof*⟩

lemma *VarP_HPairE* [*intro*!]: *insert* (*VarP* (*HPair x y*)) *H* ⊢ *A*
⟨*proof*⟩

5.1.2 De Bruijn Indexes

abbreviation *Q_Ind* :: *tm* ⇒ *tm*
where *Q_Ind k* ≡ *HPair* (*HTuple 6*) *k*

nominal_function *IndP* :: *tm* ⇒ *fm*
where *atom m* ♯ *x* ⟹

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 ♯ IndP x ⟷ a ♯ x* (**is** *?thesis1*)
and *IndP_sf* [*iff*]: *Sigma_fm (IndP x)* (**is** *?thsf*)
and *OrdP_IndP_Q_Ind*: *{OrdP x} ⊢ IndP (Q_Ind x)* (**is** *?thqind*)
⟨*proof*⟩

lemma *IndP_Q_Ind*: *H ⊢ OrdP x ⟹ H ⊢ IndP (Q_Ind x)*
⟨*proof*⟩

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

lemma *IndP_cong*: *H ⊢ x EQ x' ⟹ H ⊢ IndP x IFF IndP x'*
⟨*proof*⟩

5.1.3 Various syntactic lemmas

5.2 The predicate *SeqCTermP*, for Terms and Constants

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 ♯ SeqCTermP vf s k t ⟷ a ♯ s ∧ a ♯ k ∧ a ♯ t (**is** *?thesis1*)
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 *SeqTermP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
  where *SeqTermP* ≡ *SeqCTermP True*

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

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

5.3 The predicates *TermP* and *ConstP*

5.3.1 Definition

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

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

5.3.2 Correctness properties for constants

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

5.4 Abstraction over terms

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 *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*]

5.4.1 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 *AbstTermP_sf* [*iff*]:
Sigma_fm (AbstTermP v i t u) (**is** *?thsf*)
and *AbstTermP_imp_VarP*:
{ AbstTermP v i t u } ⊢ VarP v (**is** *?thvar*)
and *AbstTermP_imp_OrdP*:
{ AbstTermP v i t u } ⊢ OrdP i (**is** *?thord*)
⟨*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*]

5.5 Substitution over terms

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 *SubstTermP_sf* [*iff*]:
      Sigma_fm (*SubstTermP v i t u*) (**is** *?thsf*)
    and *SubstTermP_imp_TermP*:
      { *SubstTermP v i t u* } ⊢ *TermP i* (**is** *?thterm*)
    and *SubstTermP_imp_VarP*:
      { *SubstTermP v i t u* } ⊢ *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* ⊢ *v EQ v*′; *H* ⊢ *i EQ i*′; *H* ⊢ *t EQ t*′; *H* ⊢ *u EQ u*′⟧
  ⟹ *H* ⊢ *SubstTermP v i t u IFF SubstTermP v*′ *i*′ *t*′ *u*′
  ⟨*proof*⟩

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

5.6 Abstraction over formulas

5.6.1 The predicate *AbstAtomicP*

nominal_function *AbstAtomicP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
  where ⟦*atom t* ♯ (*v*,*i*,*y*,*y*′,*t*′,*u*,*u*′); *atom t*′ ♯ (*v*,*i*,*y*,*y*′,*u*,*u*′);
      atom u ♯ (*v*,*i*,*y*,*y*′,*u*′); *atom u*′ ♯ (*v*,*i*,*y*,*y*′)⟧ ⟹
    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_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 *AbstAtomicP_fresh_iff* [*simp*]:
      a ♯ *AbstAtomicP v i y y*′ ⟷ *a* ♯ *v* ∧ *a* ♯ *i* ∧ *a* ♯ *y* ∧ *a* ♯ *y*′ (**is** *?thesis1*)
    and *AbstAtomicP_sf* [*iff*]: *Sigma_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*

nominal_function $SeqAbstFormP :: tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $[\![atom\ l \sharp (s,k,v,sli,sl,sl',m,n,smi,sm,sm',sni,sn,sn');$
$atom\ sli \sharp (s,v,sl,sl',m,n,smi,sm,sm',sni,sn,sn');$
$atom\ sl \sharp (s,v,sl',m,n,smi,sm,sm',sni,sn,sn');$
$atom\ sl' \sharp (s,v,m,n,smi,sm,sm',sni,sn,sn');$
$atom\ m \sharp (s,n,smi,sm,sm',sni,sn,sn');$
$atom\ n \sharp (s,smi,sm,sm',sni,sn,sn');\ atom\ smi \sharp (s,sm,sm',sni,sn,sn');$
$atom\ sm \sharp (s,sm',sni,sn,sn');\ atom\ sm' \sharp (s,sni,sn,sn');$
$atom\ sni \sharp (s,sn,sn');\ atom\ sn \sharp (s,sn');\ atom\ sn' \sharp (s)]\!] \Longrightarrow$
$SeqAbstFormP\ v\ i\ x\ x'\ s\ k =$
$LstSeqP\ s\ k\ (HPair\ i\ (HPair\ x\ x'))\ AND$
$All2\ l\ (SUCC\ k)\ (Ex\ sli\ (Ex\ sl\ (Ex\ sl'\ (HPair\ (Var\ l)\ (HPair\ (Var\ sli)\ (HPair\ (Var\ sl)\ (Var\ sl')))$
$IN\ s\ AND$
$(AbstAtomicP\ v\ (Var\ sli)\ (Var\ sl)\ (Var\ sl')\ OR$
$OrdP\ (Var\ sli)\ AND$
$Ex\ m\ (Ex\ n\ (Ex\ smi\ (Ex\ sm\ (Ex\ sm'\ (Ex\ sni\ (Ex\ sn\ (Ex\ sn'$
$(Var\ m\ IN\ Var\ l\ AND\ Var\ n\ IN\ Var\ l\ AND$
$HPair\ (Var\ m)\ (HPair\ (Var\ smi)\ (HPair\ (Var\ sm)\ (Var\ sm')))\ IN\ s\ AND$
$HPair\ (Var\ n)\ (HPair\ (Var\ sni)\ (HPair\ (Var\ sn)\ (Var\ sn')))\ IN\ s\ AND$
$((Var\ sli\ EQ\ Var\ smi\ AND\ Var\ sli\ EQ\ Var\ sni\ AND$
$Var\ sl\ EQ\ Q\_Disj\ (Var\ sm)\ (Var\ sn)\ AND$
$Var\ sl'\ EQ\ Q\_Disj\ (Var\ sm')\ (Var\ sn'))\ OR$
$(Var\ sli\ EQ\ Var\ smi\ AND$
$Var\ sl\ EQ\ Q\_Neg\ (Var\ sm)\ AND\ Var\ sl'\ EQ\ Q\_Neg\ (Var\ sm'))\ OR$
$(SUCC\ (Var\ sli)\ EQ\ Var\ smi\ AND$
$Var\ sl\ EQ\ Q\_Ex\ (Var\ sm)\ AND\ Var\ sl'\ EQ\ Q\_Ex\ (Var\ sm'))))))))))))))))$
⟨*proof*⟩

nominal_termination (*eqvt*)
⟨*proof*⟩

lemma
shows *SeqAbstFormP_fresh_iff* [*simp*]:
$a \sharp SeqAbstFormP\ v\ i\ x\ x'\ s\ k \longleftrightarrow a \sharp v \wedge a \sharp i \wedge a \sharp x \wedge a \sharp x' \wedge a \sharp s \wedge a \sharp k$ (**is** *?thesis1*)
and *SeqAbstFormP_sf* [*iff*]:
$Sigma\_fm\ (SeqAbstFormP\ v\ i\ x\ x'\ s\ k)$ (**is** *?thsf*)
and *SeqAbstFormP_imp_OrdP*:
$\{\ SeqAbstFormP\ v\ u\ x\ x'\ s\ k\ \} \vdash OrdP\ k$ (**is** *?thOrd*)
and *SeqAbstFormP_imp_LstSeqP*:
$\{\ SeqAbstFormP\ v\ u\ x\ x'\ s\ k\ \} \vdash LstSeqP\ s\ k\ (HPair\ u\ (HPair\ x\ x'))$ (**is** *?thLstSeq*)
⟨*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

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 *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*]

5.7 Substitution over formulas

5.7.1 The predicate *SubstAtomicP*

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 *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* ⊢ *v EQ v′*; *H* ⊢ *tm EQ tm′*; *H* ⊢ *x EQ x′*; *H* ⊢ *y EQ y′*⟧
 ⟹ *H* ⊢ *SubstAtomicP v tm x y IFF SubstAtomicP v′ tm′ x′ y′*
⟨*proof*⟩

5.7.2 The predicate *SubstMakeForm*

nominal_function *SeqSubstFormP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
  where $[\![atom\ l \sharp (s,k,v,u,sl,sl',m,n,sm,sm',sn,sn');$
    $atom\ sl \sharp (s,v,u,sl',m,n,sm,sm',sn,sn');$
    $atom\ sl' \sharp (s,v,u,m,n,sm,sm',sn,sn');$
    $atom\ m \sharp (s,n,sm,sm',sn,sn');\ atom\ n \sharp (s,sm,sm',sn,sn');$
    $atom\ sm \sharp (s,sm',sn,sn');\ atom\ sm' \sharp (s,sn,sn');$
    $atom\ sn \sharp (s,sn');\ atom\ sn' \sharp s]\!] \Longrightarrow$
   $SeqSubstFormP\ v\ u\ x\ x'\ s\ k =$
    $LstSeqP\ s\ k\ (HPair\ x\ x')\ AND$
    $All2\ l\ (SUCC\ k)\ (Ex\ sl\ (Ex\ sl'\ (HPair\ (Var\ l)\ (HPair\ (Var\ sl)\ (Var\ sl'))\ IN\ s\ AND$
      $(SubstAtomicP\ v\ u\ (Var\ sl)\ (Var\ sl')\ OR$
      $Ex\ m\ (Ex\ n\ (Ex\ sm\ (Ex\ sm'\ (Ex\ sn\ (Ex\ sn'\ (Var\ m\ IN\ Var\ l\ AND\ Var\ n\ IN\ Var\ l\ AND$
        $HPair\ (Var\ m)\ (HPair\ (Var\ sm)\ (Var\ sm'))\ IN\ s\ AND$
        $HPair\ (Var\ n)\ (HPair\ (Var\ sn)\ (Var\ sn'))\ IN\ s\ AND$
        $((Var\ sl\ EQ\ Q\_Disj\ (Var\ sm)\ (Var\ sn)\ AND$
         $Var\ sl'\ EQ\ Q\_Disj\ (Var\ sm')\ (Var\ sn'))\ OR$
         $(Var\ sl\ EQ\ Q\_Neg\ (Var\ sm)\ AND\ Var\ sl'\ EQ\ Q\_Neg\ (Var\ sm'))\ OR$
         $(Var\ sl\ EQ\ Q\_Ex\ (Var\ sm)\ AND\ Var\ sl'\ EQ\ Q\_Ex\ (Var\ sm'))))))))))))$
  ⟨*proof*⟩

nominal_termination (*eqvt*)
  ⟨*proof*⟩

lemma
  shows *SeqSubstFormP_fresh_iff* [*simp*]:
    $a \sharp SeqSubstFormP\ v\ u\ x\ x'\ s\ k \longleftrightarrow a \sharp v \wedge a \sharp u \wedge a \sharp x \wedge a \sharp x' \wedge a \sharp s \wedge a \sharp k$ (**is** *?thesis1*)
   and *SeqSubstFormP_sf* [*iff*]:
    $Sigma\_fm\ (SeqSubstFormP\ v\ u\ x\ x'\ s\ k)$ (**is** *?thsf*)
   and *SeqSubstFormP_imp_OrdP*:
    $\{\ SeqSubstFormP\ v\ u\ x\ x'\ s\ k\ \} \vdash OrdP\ k$ (**is** *?thOrd*)
   and *SeqSubstFormP_imp_LstSeqP*:
    $\{\ SeqSubstFormP\ v\ u\ x\ x'\ s\ k\ \} \vdash LstSeqP\ s\ k\ (HPair\ x\ x')$ (**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*:
  $[\![H \vdash t\ EQ\ t';\ H \vdash u\ EQ\ u';\ H \vdash s\ EQ\ s';\ H \vdash k\ EQ\ k']\!]$
   $\Longrightarrow H \vdash SeqSubstFormP\ v\ i\ t\ u\ s\ k\ IFF\ SeqSubstFormP\ v\ i\ t'\ u'\ s'\ k'$
   ⟨*proof*⟩

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

5.7.3 Defining the syntax: the main SubstForm predicate

nominal_function *SubstFormP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
  where $[\![atom\ s \sharp (v,i,x,x',k);\ atom\ k \sharp (v,i,x,x')]\!] \Longrightarrow$
    $SubstFormP\ v\ i\ x\ x' =$
     $VarP\ v\ AND\ TermP\ i\ AND\ Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ x\ x'\ (Var\ s)\ (Var\ k)))$
  ⟨*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 *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* ⊢ *v EQ v′*; *H* ⊢ *i EQ i′*; *H* ⊢ *t EQ t′*; *H* ⊢ *u EQ u′*⟧
⟹ *H* ⊢ *SubstFormP v i t u IFF SubstFormP v′ i′ t′ u′*
⟨*proof*⟩

lemma *ground_SubstFormP* [*simp*]: *ground_fm* (*SubstFormP v y x x′*) ⟷ *ground v* ∧ *ground y* ∧ *ground x* ∧ *ground x′*
⟨*proof*⟩

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

5.8 The predicate *AtomicP*

nominal_function *AtomicP* :: *tm* ⇒ *fm*
where ⟦*atom t* ♯ (*u*,*y*); *atom u* ♯ *y*⟧ ⟹
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* ♯ *AtomicP y* ⟷ *a* ♯ *y* (**is** *?thesis1*)
and *AtomicP_sf* [*iff*]: *Sigma_fm* (*AtomicP y*) (**is** *?thsf*)
⟨*proof*⟩

lemma *AtompicP_subst* [*simp*]: (*AtomicP t*)(*j*::=*w*) = *AtomicP* (*subst j w t*)
⟨*proof*⟩

5.9 The predicate *MakeForm*

nominal_function *MakeFormP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom v* ♯ (*y*,*u*,*w*,*au*); *atom au* ♯ (*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 ♯ *MakeFormP y u w* ⟷ *a* ♯ *y* ∧ *a* ♯ *u* ∧ *a* ♯ *w* (**is** *?thesis1*)

and *MakeFormP_sf* [*iff*]:
Sigma_fm (*MakeFormP y u w*) (**is** *?thsf*)
⟨*proof*⟩

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

lemma *MakeFormP_subst* [*simp*]: (*MakeFormP y u t*)(*j*::=*w*) = *MakeFormP* (*subst j w y*) (*subst j w u*) (*subst j w t*)
⟨*proof*⟩

5.10 The predicate *SeqFormP*

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 *SeqFormP_sf* [*iff*]: *Sigma_fm* (*SeqFormP s k t*) (**is** *?thsf*)
and *SeqFormP_imp_OrdP*:
{ *SeqFormP s k t* } ⊢ *OrdP k* (**is** *?thOrd*)
and *SeqFormP_imp_LstSeqP*:
{ *SeqFormP s k t* } ⊢ *LstSeqP s k t* (**is** *?thLstSeq*)
⟨*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

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 *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 The predicate *VarNonOccFormP* (Derived from *SubstFormP*)

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 *VarNonOccFormP_sf* [*iff*]: *Sigma_fm* (*VarNonOccFormP v y*) (**is** *?thsf*)
⟨*proof*⟩

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

end

Chapter 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

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* ♯ *SentP x* ⟷ *a* ♯ *x* (**is** *?thesis1*)
   and *SentP_sf* [*iff*]: *Sigma_fm* (*SentP x*) (**is** *?thsf*)
⟨*proof*⟩

6.1.2 The predicate *Equality_axP*, for the Equality Axioms

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*⟩

6.1.3 The predicate *HF_axP*, for the HF Axioms

function *HF_axP* :: *tm* ⇒ *fm*
  where *HF_axP x* = *x EQ* «*HF1*» *OR x EQ* «*HF2*»

⟨*proof*⟩

termination
⟨*proof*⟩

lemma *HF_axP_sf* [*iff*]: *Sigma_fm* (*HF_axP t*)
⟨*proof*⟩

6.1.4 The specialisation axioms

Defining the syntax

nominal_function *Special_axP* :: *tm* ⇒ *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*⟧ ⟹
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* ⟷ *a* ♯ *p* (**is** *?thesis1*)
and *Special_axP_sf* [*iff*]: *Sigma_fm* (*Special_axP p*) (**is** *?thesis3*)
⟨*proof*⟩

6.1.5 The induction axioms

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 *Induction_axP_sf* [*iff*]: *Sigma_fm* (*Induction_axP p*) (**is** *?thesis3*)
⟨*proof*⟩

6.1.6 The predicate *AxiomP*, for any Axioms

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_I*:
  {} ⊢ *AxiomP* «*extra_axiom*»
  {} ⊢ *SentP x* ⟹ {} ⊢ *AxiomP x*
  {} ⊢ *Equality_axP x* ⟹ {} ⊢ *AxiomP x*
  {} ⊢ *HF_axP x* ⟹ {} ⊢ *AxiomP x*
  {} ⊢ *Special_axP x* ⟹ {} ⊢ *AxiomP x*
  {} ⊢ *Induction_axP x* ⟹ {} ⊢ *AxiomP x*
  ⟨*proof*⟩

lemma *AxiomP_eqvt* [*eqvt*]: (*p* ∙ *AxiomP x*) = *AxiomP* (*p* ∙ *x*)
  ⟨*proof*⟩

lemma *AxiomP_fresh_iff* [*simp*]: *a* ♯ *AxiomP x* ⟷ *a* ♯ *x*
  ⟨*proof*⟩

lemma *AxiomP_sf* [*iff*]: *Sigma_fm* (*AxiomP t*)
  ⟨*proof*⟩

6.1.7 The predicate *ModPonP*, for the inference rule Modus Ponens

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 *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

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*]: $a \sharp \mathit{ExistsP}\ p\ q \longleftrightarrow a \sharp p \wedge a \sharp q$ (**is** *?thesis1*)
and *ExistsP_sf* [*iff*]: *Sigma_fm* (*ExistsP p q*) (**is** *?thesis3*)
⟨*proof*⟩

lemma *ExistsP_subst* [*simp*]: $(\mathit{ExistsP}\ p\ q)(j::=w) = \mathit{ExistsP}\ (\mathit{subst}\ j\ w\ p)\ (\mathit{subst}\ j\ w\ q)$
⟨*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

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*]: $a \sharp \mathit{SubstP}\ p\ q \longleftrightarrow a \sharp p \wedge a \sharp q$ (**is** *?thesis1*)
and *SubstP_sf* [*iff*]: *Sigma_fm* (*SubstP p q*) (**is** *?thesis3*)
⟨*proof*⟩

lemma *SubstP_subst* [*simp*]: $(\mathit{SubstP}\ p\ q)(j::=w) = \mathit{SubstP}\ (\mathit{subst}\ j\ w\ p)\ (\mathit{subst}\ j\ w\ q)$
⟨*proof*⟩

6.1.10 The predicate *PrfP*

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*]: $a \sharp \mathit{PrfP}\ s\ k\ t \longleftrightarrow a \sharp s \wedge a \sharp k \wedge a \sharp t$ (**is** *?thesis1*)
and *PrfP_imp_OrdP* [*simp*]: $\{\mathit{PrfP}\ s\ k\ t\} \vdash \mathit{OrdP}\ k$ (**is** *?thord*)
and *PrfP_imp_LstSeqP* [*simp*]: $\{\mathit{PrfP}\ s\ k\ t\} \vdash \mathit{LstSeqP}\ s\ k\ t$ (**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*

nominal_function *PfP* :: *tm* ⇒ *fm*
where ⟦*atom k* ♯ (*s*,*y*); *atom s* ♯ *y*⟧ ⟹
PfP y = *Ex k* (*Ex s* (*PrfP* (*Var s*) (*Var k*) *y*))
⟨*proof*⟩

nominal_termination (*eqvt*)
⟨*proof*⟩

lemma
shows *PfP_fresh_iff* [*simp*]: *a* ♯ *PfP y* ⟷ *a* ♯ *y* (**is** *?thesis1*)
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*⟩

end

Chapter 7

Syntactic Preliminaries for the Second Incompleteness Theorem

theory *II_Prelims*
imports *Pf_Predicates*
begin

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

lemma *OrdP_ORD_OF* [*intro*]: $H \vdash \mathit{OrdP}\ (\mathit{ORD\_OF}\ n)$
⟨*proof*⟩

lemma *VarP_Var* [*intro*]: $H \vdash \mathit{VarP}\ «\mathit{Var}\ i»$
⟨*proof*⟩

lemma *VarP_neq_IndP*: $\{t\ \mathit{EQ}\ v,\ \mathit{VarP}\ v,\ \mathit{IndP}\ t\} \vdash \mathit{Fls}$
⟨*proof*⟩

lemma *Mem_HFun_Sigma_OrdP*: $\{\mathit{HPair}\ t\ u\ \mathit{IN}\ f,\ \mathit{HFun\_Sigma}\ f\} \vdash \mathit{OrdP}\ t$
⟨*proof*⟩

7.1 NotInDom

nominal_function *NotInDom* :: $tm \Rightarrow tm \Rightarrow fm$
where $\mathit{atom}\ z \sharp (t, r) \Longrightarrow \mathit{NotInDom}\ t\ r = \mathit{All}\ z\ (\mathit{Neg}\ (\mathit{HPair}\ t\ (\mathit{Var}\ z)\ \mathit{IN}\ r))$
⟨*proof*⟩

nominal_termination (*eqvt*)
⟨*proof*⟩

lemma *NotInDom_fresh_iff* [*simp*]: $a \sharp \mathit{NotInDom}\ t\ r \longleftrightarrow a \sharp (t, r)$
⟨*proof*⟩

lemma *subst_fm_NotInDom* [*simp*]: $(\mathit{NotInDom}\ t\ r)(i::=x) = \mathit{NotInDom}\ (\mathit{subst}\ i\ x\ t)\ (\mathit{subst}\ i\ x\ r)$
⟨*proof*⟩

lemma *NotInDom_cong*: $H \vdash t\ \mathit{EQ}\ t' \Longrightarrow H \vdash r\ \mathit{EQ}\ r' \Longrightarrow H \vdash \mathit{NotInDom}\ t\ r\ \mathit{IFF}\ \mathit{NotInDom}\ t'\ r'$
⟨*proof*⟩

lemma *NotInDom_Zero*: $H \vdash \mathit{NotInDom}\ t\ \mathit{Zero}$
⟨*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*⟩

7.2 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*: $\{RestrictedP\ s\ k\ s'\} \vdash s'\ SUBS\ s$
⟨*proof*⟩

lemma *RestrictedP_Mem2*:
$\{\ RestrictedP\ s\ k\ s',\ HPair\ a\ b\ IN\ s'\ \} \vdash HPair\ a\ b\ IN\ s\ AND\ a\ IN\ k$
⟨*proof*⟩

lemma *RestrictedP_Mem_D*: $H \vdash RestrictedP\ s\ k\ t \Longrightarrow H \vdash a\ IN\ t \Longrightarrow insert\ (a\ IN\ s)\ H \vdash A \Longrightarrow H \vdash A$
⟨*proof*⟩

lemma *RestrictedP_Eats*:
$\{\ RestrictedP\ s\ k\ s',\ a\ IN\ k\ \} \vdash RestrictedP\ (Eats\ s\ (HPair\ a\ b))\ k\ (Eats\ s'\ (HPair\ a\ b))$ ⟨*proof*⟩
lemma *exists_RestrictedP*:
assumes *s*: $atom\ s \sharp (f,k)$
shows $H \vdash Ex\ s\ (RestrictedP\ f\ k\ (Var\ s))$ ⟨*proof*⟩
lemma *cut_RestrictedP*:
assumes *s*: $atom\ s \sharp (f,k,A)$ **and** $\forall C \in H.\ atom\ s \sharp C$
shows $insert\ (RestrictedP\ f\ k\ (Var\ s))\ H \vdash A \Longrightarrow H \vdash A$
⟨*proof*⟩

lemma *RestrictedP_NotInDom*: { *RestrictedP s k s′*, *Neg* (*j IN k*) } ⊢ *NotInDom j s′*
⟨*proof*⟩

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

7.3 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′*} ⊢ *HDomain_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*⟩

7.4 Ordinal Addition

7.4.1 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* ♯ *HaddP x y z* ⟷ *a* ♯ *x* ∧ *a* ♯ *y* ∧ *a* ♯ *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* ⊢ *t EQ t*′; *H* ⊢ *u EQ u*′; *H* ⊢ *v EQ v*′⟧ ⟹ *H* ⊢ *HaddP t u v IFF HaddP t*′ *u*′ *v*′
  ⟨*proof*⟩

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

lemma *HaddP_Zero2*: *H* ⊢ *HaddP x Zero x*
⟨*proof*⟩

lemma *HaddP_imp_OrdP*: {*HaddP x y z*} ⊢ *OrdP y*
⟨*proof*⟩

lemma *HaddP_SUCC2*: {*HaddP x y z*} ⊢ *HaddP x* (*SUCC y*) (*SUCC z*) ⟨*proof*⟩

7.4.2 Proving that these relations are functions

lemma *SeqHaddP_Zero_E*: {*SeqHaddP s w Zero z*} ⊢ *w EQ z*
⟨*proof*⟩

lemma *SeqHaddP_SUCC_lemma*:
  assumes *y*′: *atom y*′ ♯ (*s*,*j*,*k*,*y*)
  shows {*SeqHaddP s j* (*SUCC k*) *y*} ⊢ *Ex y*′ (*SeqHaddP s j k* (*Var y*′) *AND y EQ SUCC* (*Var y*′))
⟨*proof*⟩

lemma *SeqHaddP_SUCC*:
  assumes *H* ⊢ *SeqHaddP s j* (*SUCC k*) *y* *atom y*′ ♯ (*s*,*j*,*k*,*y*)
  shows *H* ⊢ *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*′} ⊢ *y*′ *EQ y* ⟨*proof*⟩
lemma *HaddP_unique*: {*HaddP w x y*, *HaddP w x y*′} ⊢ *y*′ *EQ y*
⟨*proof*⟩

lemma *HaddP_Zero1*: **assumes** *H* ⊢ *OrdP x* **shows** *H* ⊢ *HaddP Zero x x*
⟨*proof*⟩

lemma *HaddP_Zero_D1*: *insert* (*HaddP Zero x y*) *H* ⊢ *x EQ y*
  ⟨*proof*⟩

lemma *HaddP_Zero_D2*: *insert* (*HaddP x Zero y*) *H* ⊢ *x EQ y*
  ⟨*proof*⟩

lemma *HaddP_SUCC_Ex2*:
assumes $H \vdash HaddP\ x\ (SUCC\ y)\ z\ atom\ z' \sharp (x,y,z)$
shows $H \vdash Ex\ z'\ (HaddP\ x\ y\ (Var\ z')\ AND\ z\ EQ\ SUCC\ (Var\ z'))$
⟨*proof*⟩

lemma *HaddP_SUCC1*: $\{\ HaddP\ x\ y\ z\ \} \vdash HaddP\ (SUCC\ x)\ y\ (SUCC\ z)$ ⟨*proof*⟩
lemma *HaddP_commute*: $\{HaddP\ x\ y\ z,\ OrdP\ x\} \vdash HaddP\ y\ x\ z$ ⟨*proof*⟩
lemma *HaddP_SUCC_Ex1*:
assumes $atom\ i \sharp (x,y,z)$
shows $insert\ (HaddP\ (SUCC\ x)\ y\ z)\ (insert\ (OrdP\ x)\ H)$
$\vdash Ex\ i\ (HaddP\ x\ y\ (Var\ i)\ AND\ z\ EQ\ SUCC\ (Var\ i))$
⟨*proof*⟩

lemma *HaddP_inv2*: $\{HaddP\ x\ y\ z,\ HaddP\ x\ y'\ z,\ OrdP\ x\} \vdash y'\ EQ\ y$ ⟨*proof*⟩
lemma *Mem_imp_subtract*: ⟨*proof*⟩
lemma *HaddP_OrdP*:
assumes $H \vdash HaddP\ x\ y\ z\ H \vdash OrdP\ x$ **shows** $H \vdash OrdP\ z$ ⟨*proof*⟩
lemma *HaddP_Mem_cancel_left*:
assumes $H \vdash HaddP\ x\ y'\ z'\ H \vdash HaddP\ x\ y\ z\ H \vdash OrdP\ x$
shows $H \vdash z'\ IN\ z\ IFF\ y'\ IN\ y$ ⟨*proof*⟩
lemma *HaddP_Mem_cancel_right_Mem*:
assumes $H \vdash HaddP\ x'\ y\ z'\ H \vdash HaddP\ x\ y\ z\ H \vdash x'\ IN\ x\ H \vdash OrdP\ x$
shows $H \vdash z'\ IN\ z$
⟨*proof*⟩

lemma *HaddP_Mem_cases*:
assumes $H \vdash HaddP\ k1\ k2\ k\ H \vdash OrdP\ k1$
$insert\ (x\ IN\ k1)\ H \vdash A$
$insert\ (Var\ i\ IN\ k2)\ (insert\ (HaddP\ k1\ (Var\ i)\ x)\ H) \vdash A$
and i: $atom\ (i::name) \sharp (k1,k2,k,x,A)$ **and** $\forall\, C \in H.\ atom\ i \sharp C$
shows $insert\ (x\ IN\ k)\ H \vdash A$ ⟨*proof*⟩
lemma *HaddP_Mem_contra*:
assumes $H \vdash HaddP\ x\ y\ z\ H \vdash z\ IN\ x\ H \vdash OrdP\ x$
shows $H \vdash A$
⟨*proof*⟩

lemma *exists_HaddP*:
assumes $H \vdash OrdP\ y\ atom\ j \sharp (x,y)$
shows $H \vdash Ex\ j\ (HaddP\ x\ y\ (Var\ j))$
⟨*proof*⟩

lemma *HaddP_Mem_I*:
assumes $H \vdash HaddP\ x\ y\ z\ H \vdash OrdP\ x$ **shows** $H \vdash x\ IN\ SUCC\ z$
⟨*proof*⟩

7.5 A Shifted Sequence

nominal_function *ShiftP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $[\![atom\ x \sharp (x',y,z,f,del,k);\ atom\ x' \sharp (y,z,f,del,k);\ atom\ y \sharp (z,f,del,k);\ atom\ z \sharp (f,del,g,k)]\!] \Longrightarrow$
$ShiftP\ f\ k\ del\ g =$
$All\ z\ (Var\ z\ IN\ g\ IFF$
$(Ex\ x\ (Ex\ x'\ (Ex\ y\ ((Var\ z)\ EQ\ HPair\ (Var\ x')\ (Var\ y)\ AND$
$HaddP\ del\ (Var\ x)\ (Var\ x')\ AND$
$HPair\ (Var\ x)\ (Var\ y)\ IN\ f\ AND\ Var\ x\ IN\ k)))))$
⟨*proof*⟩

nominal_termination (*eqvt*)
⟨*proof*⟩

lemma *ShiftP_fresh_iff* [*simp*]: $a \sharp \mathit{ShiftP}\ f\ k\ \mathit{del}\ g \longleftrightarrow a \sharp f \wedge a \sharp k \wedge a \sharp \mathit{del} \wedge a \sharp g$
⟨*proof*⟩

lemma *subst_fm_ShiftP* [*simp*]:
$(\mathit{ShiftP}\ f\ k\ \mathit{del}\ g)(i::=u) = \mathit{ShiftP}\ (\mathit{subst}\ i\ u\ f)\ (\mathit{subst}\ i\ u\ k)\ (\mathit{subst}\ i\ u\ \mathit{del})\ (\mathit{subst}\ i\ u\ g)$
⟨*proof*⟩

lemma *ShiftP_Zero*: $\{\} \vdash \mathit{ShiftP}\ \mathit{Zero}\ k\ d\ \mathit{Zero}$
⟨*proof*⟩

lemma *ShiftP_Mem1*:
$\{\mathit{ShiftP}\ f\ k\ \mathit{del}\ g,\ \mathit{HPair}\ a\ b\ \mathit{IN}\ f,\ \mathit{HaddP}\ \mathit{del}\ a\ a',\ a\ \mathit{IN}\ k\} \vdash \mathit{HPair}\ a'\ b\ \mathit{IN}\ g$
⟨*proof*⟩

lemma *ShiftP_Mem2*:
assumes $\mathit{atom}\ u \sharp (f,k,\mathit{del},a,b)$
shows $\{\mathit{ShiftP}\ f\ k\ \mathit{del}\ g,\ \mathit{HPair}\ a\ b\ \mathit{IN}\ g\} \vdash \mathit{Ex}\ u\ ((\mathit{Var}\ u)\ \mathit{IN}\ k\ \mathit{AND}\ \mathit{HaddP}\ \mathit{del}\ (\mathit{Var}\ u)\ a\ \mathit{AND}\ \mathit{HPair}\ (\mathit{Var}\ u)\ b\ \mathit{IN}\ f)$
⟨*proof*⟩

lemma *ShiftP_Mem_D*:
assumes $H \vdash \mathit{ShiftP}\ f\ k\ \mathit{del}\ g\ H \vdash a\ \mathit{IN}\ g$
$\mathit{atom}\ x \sharp (x',y,a,f,\mathit{del},k)\ \mathit{atom}\ x' \sharp (y,a,f,\mathit{del},k)\ \mathit{atom}\ y \sharp (a,f,\mathit{del},k)$
shows $H \vdash (\mathit{Ex}\ x\ (\mathit{Ex}\ x'\ (\mathit{Ex}\ y\ (a\ \mathit{EQ}\ \mathit{HPair}\ (\mathit{Var}\ x')\ (\mathit{Var}\ y)\ \mathit{AND}$
$\mathit{HaddP}\ \mathit{del}\ (\mathit{Var}\ x)\ (\mathit{Var}\ x')\ \mathit{AND}$
$\mathit{HPair}\ (\mathit{Var}\ x)\ (\mathit{Var}\ y)\ \mathit{IN}\ f\ \mathit{AND}\ \mathit{Var}\ x\ \mathit{IN}\ k))))$
(**is** $\_ \vdash$ *?concl*)
⟨*proof*⟩

lemma *ShiftP_Eats_Eats*:
$\{\mathit{ShiftP}\ f\ k\ \mathit{del}\ g,\ \mathit{HaddP}\ \mathit{del}\ a\ a',\ a\ \mathit{IN}\ k\}$
$\vdash \mathit{ShiftP}\ (\mathit{Eats}\ f\ (\mathit{HPair}\ a\ b))\ k\ \mathit{del}\ (\mathit{Eats}\ g\ (\mathit{HPair}\ a'\ b))$ ⟨*proof*⟩
lemma *ShiftP_Eats_Neg*:
assumes $\mathit{atom}\ u \sharp (u',v,f,k,\mathit{del},g,c)\ \mathit{atom}\ u' \sharp (v,f,k,\mathit{del},g,c)\ \mathit{atom}\ v \sharp (f,k,\mathit{del},g,c)$
shows
$\{\mathit{ShiftP}\ f\ k\ \mathit{del}\ g,$
$\mathit{Neg}\ (\mathit{Ex}\ u\ (\mathit{Ex}\ u'\ (\mathit{Ex}\ v\ (c\ \mathit{EQ}\ \mathit{HPair}\ (\mathit{Var}\ u)\ (\mathit{Var}\ v)\ \mathit{AND}\ \mathit{Var}\ u\ \mathit{IN}\ k\ \mathit{AND}\ \mathit{HaddP}\ \mathit{del}\ (\mathit{Var}\ u)\ (\mathit{Var}\ u'))))))\}$
$\vdash \mathit{ShiftP}\ (\mathit{Eats}\ f\ c)\ k\ \mathit{del}\ g$ ⟨*proof*⟩
lemma *exists_ShiftP*:
assumes *t*: $\mathit{atom}\ t \sharp (s,k,\mathit{del})$
shows $H \vdash \mathit{Ex}\ t\ (\mathit{ShiftP}\ s\ k\ \mathit{del}\ (\mathit{Var}\ t))$ ⟨*proof*⟩

7.6 Union of Two Sets

nominal_function $\mathit{UnionP} :: \mathit{tm} \Rightarrow \mathit{tm} \Rightarrow \mathit{tm} \Rightarrow \mathit{fm}$
where $\mathit{atom}\ i \sharp (x,y,z) \Longrightarrow \mathit{UnionP}\ x\ y\ z = \mathit{All}\ i\ (\mathit{Var}\ i\ \mathit{IN}\ z\ \mathit{IFF}\ (\mathit{Var}\ i\ \mathit{IN}\ x\ \mathit{OR}\ \mathit{Var}\ i\ \mathit{IN}\ y))$
⟨*proof*⟩

nominal_termination (*eqvt*)
⟨*proof*⟩

lemma *UnionP_fresh_iff* [*simp*]: $a \sharp \mathit{UnionP}\ x\ y\ z \longleftrightarrow a \sharp x \wedge a \sharp y \wedge a \sharp z$
⟨*proof*⟩

lemma *subst_fm_UnionP* [*simp*]:
$(\mathit{UnionP}\ x\ y\ z)(i::=u) = \mathit{UnionP}\ (\mathit{subst}\ i\ u\ x)\ (\mathit{subst}\ i\ u\ y)\ (\mathit{subst}\ i\ u\ z)$

⟨*proof*⟩

lemma *Union_Zero1*: $H \vdash$ *UnionP Zero x x*
⟨*proof*⟩

lemma *Union_Eats*: {*UnionP x y z*} $\vdash$ *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 {} $\vdash$ *Ex z* (*UnionP* (*Var i*) *y* (*Var z*))
⟨*proof*⟩

lemma *exists_UnionP*:
assumes *z*: *atom z* ♯ (*x*,*y*) **shows** $H \vdash$ *Ex z* (*UnionP x y* (*Var z*))
⟨*proof*⟩

lemma *UnionP_Mem1*: { *UnionP x y z*, *a IN x* } $\vdash$ *a IN z*
⟨*proof*⟩

lemma *UnionP_Mem2*: { *UnionP x y z*, *a IN y* } $\vdash$ *a IN z*
⟨*proof*⟩

lemma *UnionP_Mem*: { *UnionP x y z*, *a IN z* } $\vdash$ *a IN x OR a IN y*
⟨*proof*⟩

lemma *UnionP_Mem_E*:
assumes $H \vdash$ *UnionP x y z*
and *insert* (*a IN x*) $H \vdash A$
and *insert* (*a IN y*) $H \vdash A$
shows *insert* (*a IN z*) $H \vdash A$
⟨*proof*⟩

7.7 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 $H \vdash$ *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*⟩

7.8 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*⟩

7.9 Substitution and Abstraction on Terms

7.9.1 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*⟩

7.9.2 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*⟩

7.9.3 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*⟩

7.10 Substitution on Formulas

7.10.1 Membership

lemma *SubstAtomicP_Mem*:
{*SubstTermP v i x x′*, *SubstTermP v i y y′*} ⊢ *SubstAtomicP v i* (*Q_Mem x y*) (*Q_Mem x′ y′*)
⟨*proof*⟩

lemma *SeqSubstFormP_Mem*:
assumes *atom s* ♯ (*k*,*x*,*y*,*x′*,*y′*,*v*,*i*) *atom k* ♯ (*x*,*y*,*x′*,*y′*,*v*,*i*)
shows {*SubstTermP v i x x′*, *SubstTermP v i y y′*}
⊢ *Ex s* (*Ex k* (*SeqSubstFormP v i* (*Q_Mem x y*) (*Q_Mem x′ y′*) (*Var s*) (*Var k*)))
⟨*proof*⟩

lemma *SubstFormP_Mem*:

$\{SubstTermP\ v\ i\ x\ x',\ SubstTermP\ v\ i\ y\ y'\} \vdash SubstFormP\ v\ i\ (Q\_Mem\ x\ y)\ (Q\_Mem\ x'\ y')$
⟨*proof*⟩

7.10.2 Equality

lemma *SubstAtomicP_Eq*:
$\{SubstTermP\ v\ i\ x\ x',\ SubstTermP\ v\ i\ y\ y'\} \vdash SubstAtomicP\ v\ i\ (Q\_Eq\ x\ y)\ (Q\_Eq\ x'\ y')$
⟨*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 $\{SubstTermP\ v\ i\ x\ x',\ SubstTermP\ v\ i\ y\ y'\}$
$\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q\_Eq\ x\ y)\ (Q\_Eq\ x'\ y')\ (Var\ s)\ (Var\ k)))$
⟨*proof*⟩

lemma *SubstFormP_Eq*:
$\{SubstTermP\ v\ i\ x\ x',\ SubstTermP\ v\ i\ y\ y'\} \vdash SubstFormP\ v\ i\ (Q\_Eq\ x\ y)\ (Q\_Eq\ x'\ y')$
⟨*proof*⟩

7.10.3 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 $\{SeqSubstFormP\ v\ i\ x\ x'\ s1\ k1,\ TermP\ i,\ VarP\ v\}$
$\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q\_Neg\ x)\ (Q\_Neg\ x')\ (Var\ s)\ (Var\ k)))$ ⟨*proof*⟩
theorem *SubstFormP_Neg*: $\{SubstFormP\ v\ i\ x\ x'\} \vdash SubstFormP\ v\ i\ (Q\_Neg\ x)\ (Q\_Neg\ x')$
⟨*proof*⟩

7.10.4 Disjunction

lemma *SeqSubstFormP_Disj*:
assumes $atom\ s \sharp (k,s1,s2,k1,k2,x,y,x',y',v,i)\ atom\ k \sharp (s1,s2,k1,k2,x,y,x',y',v,i)$
shows $\{SeqSubstFormP\ v\ i\ x\ x'\ s1\ k1,$
$SeqSubstFormP\ v\ i\ y\ y'\ s2\ k2,\ TermP\ i,\ VarP\ v\}$
$\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q\_Disj\ x\ y)\ (Q\_Disj\ x'\ y')\ (Var\ s)\ (Var\ k)))$ ⟨*proof*⟩
theorem *SubstFormP_Disj*:
$\{SubstFormP\ v\ i\ x\ x',\ SubstFormP\ v\ i\ y\ y'\} \vdash SubstFormP\ v\ i\ (Q\_Disj\ x\ y)\ (Q\_Disj\ x'\ y')$
⟨*proof*⟩

7.10.5 Existential

lemma *SeqSubstFormP_Ex*:
assumes $atom\ s \sharp (k,s1,k1,x,x',v,i)\ atom\ k \sharp (s1,k1,x,x',v,i)$
shows $\{SeqSubstFormP\ v\ i\ x\ x'\ s1\ k1,\ TermP\ i,\ VarP\ v\}$
$\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q\_Ex\ x)\ (Q\_Ex\ x')\ (Var\ s)\ (Var\ k)))$ ⟨*proof*⟩
theorem *SubstFormP_Ex*: $\{SubstFormP\ v\ i\ x\ x'\} \vdash SubstFormP\ v\ i\ (Q\_Ex\ x)\ (Q\_Ex\ x')$
⟨*proof*⟩

7.11 Constant Terms

lemma *ConstP_Zero*: $\{\} \vdash ConstP\ Zero$
⟨*proof*⟩

lemma *SeqConstP_Eats*:
assumes $atom\ s \sharp (k,s1,s2,k1,k2,t1,t2)\ atom\ k \sharp (s1,s2,k1,k2,t1,t2)$
shows $\{SeqConstP\ s1\ k1\ t1,\ SeqConstP\ s2\ k2\ t2\}$
$\vdash Ex\ s\ (Ex\ k\ (SeqConstP\ (Var\ s)\ (Var\ k)\ (Q\_Eats\ t1\ t2)))$ ⟨*proof*⟩
theorem *ConstP_Eats*: $\{ConstP\ t1,\ ConstP\ t2\} \vdash ConstP\ (Q\_Eats\ t1\ t2)$

⟨*proof*⟩

lemma *TermP_Zero*: {} ⊢ *TermP Zero*
⟨*proof*⟩

lemma *TermP_Var*: {} ⊢ *TermP* «*Var x*»
⟨*proof*⟩

lemma *SeqTermP_Eats*:
assumes *atom s* ♯ (*k*,*s1*,*s2*,*k1*,*k2*,*t1*,*t2*) *atom k* ♯ (*s1*,*s2*,*k1*,*k2*,*t1*,*t2*)
shows {*SeqTermP s1 k1 t1*, *SeqTermP s2 k2 t2*}
⊢ *Ex s* (*Ex k* (*SeqTermP* (*Var s*) (*Var k*) (*Q_Eats t1 t2*))) ⟨*proof*⟩
theorem *TermP_Eats*: {*TermP t1*, *TermP t2*} ⊢ *TermP* (*Q_Eats t1 t2*)
⟨*proof*⟩

7.12 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* ⊢ *PfP z*
⟨*proof*⟩

theorem *PfP_implies_ModPon_PfP*: ⟦*H* ⊢ *PfP* (*Q_Imp x y*); *H* ⊢ *PfP x*⟧ ⟹ *H* ⊢ *PfP y*
⟨*proof*⟩

corollary *PfP_implies_ModPon_PfP_quot*: ⟦*H* ⊢ *PfP* «*α IMP β*»; *H* ⊢ *PfP* «*α*»⟧ ⟹ *H* ⊢ *PfP* «*β*»
⟨*proof*⟩

lemma *TermP_quot*:
fixes *α* :: *tm*
shows {} ⊢ *TermP* «*α*»
⟨*proof*⟩

lemma *TermP_quot_dbtm*:
fixes *α* :: *tm*
assumes *wf_dbtm u*
shows {} ⊢ *TermP* (*quot_dbtm u*)
⟨*proof*⟩

7.13 Formulas

7.14 Abstraction on Formulas

7.14.1 Membership

lemma *AbstAtomicP_Mem*:
{*AbstTermP v i x x′*, *AbstTermP v i y y′*} ⊢ *AbstAtomicP v i* (*Q_Mem x y*) (*Q_Mem x′ y′*)
⟨*proof*⟩

lemma *SeqAbstFormP_Mem*:
assumes *atom s* ♯ (*k*,*x*,*y*,*x′*,*y′*,*v*,*i*) *atom k* ♯ (*x*,*y*,*x′*,*y′*,*v*,*i*)
shows {*AbstTermP v i x x′*, *AbstTermP v i y y′*}
⊢ *Ex s* (*Ex k* (*SeqAbstFormP v i* (*Q_Mem x y*) (*Q_Mem x′ y′*) (*Var s*) (*Var k*)))
⟨*proof*⟩

lemma *AbstFormP_Mem*:
{*AbstTermP v i x x′*, *AbstTermP v i y y′*} ⊢ *AbstFormP v i* (*Q_Mem x y*) (*Q_Mem x′ y′*)
⟨*proof*⟩

7.14.2 Equality

lemma *AbstAtomicP_Eq*:
{*AbstTermP v i x x′*, *AbstTermP v i y y′*} ⊢ *AbstAtomicP v i* (*Q_Eq x y*) (*Q_Eq x′ y′*)
⟨*proof*⟩

lemma *SeqAbstFormP_Eq*:
assumes *sk*: *atom s* ♯ (*k*,*x*,*y*,*x′*,*y′*,*v*,*i*) *atom k* ♯ (*x*,*y*,*x′*,*y′*,*v*,*i*)
shows {*AbstTermP v i x x′*, *AbstTermP v i y y′*}
⊢ *Ex s* (*Ex k* (*SeqAbstFormP v i* (*Q_Eq x y*) (*Q_Eq x′ y′*) (*Var s*) (*Var k*)))
⟨*proof*⟩

lemma *AbstFormP_Eq*:
{*AbstTermP v i x x′*, *AbstTermP v i y y′*} ⊢ *AbstFormP v i* (*Q_Eq x y*) (*Q_Eq x′ y′*)
⟨*proof*⟩

7.14.3 Negation

lemma *SeqAbstFormP_Neg*:
assumes *atom s* ♯ (*k*,*s1*,*k1*,*x*,*x′*,*v*,*i*) *atom k* ♯ (*s1*,*k1*,*x*,*x′*,*v*,*i*)
shows {*SeqAbstFormP v i x x′ s1 k1*, *OrdP i*, *VarP v*}
⊢ *Ex s* (*Ex k* (*SeqAbstFormP v i* (*Q_Neg x*) (*Q_Neg x′*) (*Var s*) (*Var k*))) ⟨*proof*⟩
theorem *AbstFormP_Neg*: {*AbstFormP v i x x′*} ⊢ *AbstFormP v i* (*Q_Neg x*) (*Q_Neg x′*)
⟨*proof*⟩

7.14.4 Disjunction

lemma *SeqAbstFormP_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 {*SeqAbstFormP v i x x′ s1 k1*,
SeqAbstFormP v i y y′ s2 k2, *OrdP i*, *VarP v*}
⊢ *Ex s* (*Ex k* (*SeqAbstFormP v i* (*Q_Disj x y*) (*Q_Disj x′ y′*) (*Var s*) (*Var k*))) ⟨*proof*⟩
theorem *AbstFormP_Disj*:
{*AbstFormP v i x x′*, *AbstFormP v i y y′*} ⊢ *AbstFormP v i* (*Q_Disj x y*) (*Q_Disj x′ y′*)
⟨*proof*⟩

7.14.5 Existential

lemma *SeqAbstFormP_Ex*:
assumes *atom s* ♯ (*k*,*s1*,*k1*,*x*,*x′*,*v*,*i*) *atom k* ♯ (*s1*,*k1*,*x*,*x′*,*v*,*i*)
shows {*SeqAbstFormP v* (*SUCC i*) *x x′ s1 k1*, *OrdP i*, *VarP v*}
⊢ *Ex s* (*Ex k* (*SeqAbstFormP v i* (*Q_Ex x*) (*Q_Ex x′*) (*Var s*) (*Var k*))) ⟨*proof*⟩
theorem *AbstFormP_Ex*: {*AbstFormP v* (*SUCC i*) *x x′*} ⊢ *AbstFormP v i* (*Q_Ex x*) (*Q_Ex x′*)
⟨*proof*⟩

corollary *AbstTermP_Zero*: {*OrdP t*} ⊢ *AbstTermP* «*Var v*» *t Zero Zero*
⟨*proof*⟩

corollary *AbstTermP_Var_same*: $\{VarP\ v,\ OrdP\ t\} \vdash AbstTermP\ v\ t\ v\ (Q\_Ind\ t)$
⟨*proof*⟩

corollary *AbstTermP_Var_diff*: $\{VarP\ v,\ VarP\ w,\ Neg\ (v\ EQ\ w),\ OrdP\ t\} \vdash AbstTermP\ v\ t\ w\ w$
⟨*proof*⟩

theorem *AbstTermP_Eats*:
$\{AbstTermP\ v\ i\ t1\ u1,\ AbstTermP\ v\ i\ t2\ u2\} \vdash AbstTermP\ v\ i\ (Q\_Eats\ t1\ t2)\ (Q\_Eats\ u1\ u2)$
⟨*proof*⟩

corollary *AbstTermP_Ind*: $\{VarP\ v,\ IndP\ w,\ OrdP\ t\} \vdash AbstTermP\ v\ t\ w\ w$
⟨*proof*⟩

lemma *ORD_OF_EQ_diff*: $x \neq y \Longrightarrow \{ORD\_OF\ x\ EQ\ ORD\_OF\ y\} \vdash Fls$
⟨*proof*⟩

lemma *quot_Var_EQ_diff*: $i \neq x \Longrightarrow \{«Var\ i»\ EQ\ «Var\ x»\} \vdash Fls$
⟨*proof*⟩

lemma *AbstTermP_dbtm*: $\{\} \vdash AbstTermP\ «Var\ i»\ (ORD\_OF\ n)\ (quot\_dbtm\ u)\ (quot\_dbtm\ (abst\_dbtm\ i\ n\ u))$
⟨*proof*⟩

lemma *AbstFormP_dbfm*: $\{\} \vdash AbstFormP\ «Var\ i»\ (ORD\_OF\ n)\ (quot\_dbfm\ db)\ (quot\_dbfm\ (abst\_dbfm\ i\ n\ db))$
⟨*proof*⟩

lemmas *AbstFormP* = *AbstFormP_dbfm*[**where** *db*=*trans_fm* [] *A* **and** *n* = *0* **for** *A*,
simplified, *folded quot_fm_def*, *unfolded abst_trans_fm*]

lemma *SubstTermP_trivial_dbtm*:
$atom\ i \sharp u \Longrightarrow \{\} \vdash SubstTermP\ «Var\ i»\ Zero\ (quot\_dbtm\ u)\ (quot\_dbtm\ u)$
⟨*proof*⟩

lemma *SubstTermP_dbtm*: $wf\_dbtm\ t \Longrightarrow$
$\{\} \vdash SubstTermP\ «Var\ i»\ (quot\_dbtm\ t)\ (quot\_dbtm\ u)\ (quot\_dbtm\ (subst\_dbtm\ t\ i\ u))$
⟨*proof*⟩

lemma *SubstFormP_trivial_dbfm*:
fixes *X* :: *fm*
assumes $atom\ i \sharp db$
shows $\{\} \vdash SubstFormP\ «Var\ i»\ Zero\ (quot\_dbfm\ db)\ (quot\_dbfm\ db)$
⟨*proof*⟩

lemma *SubstFormP_dbfm*:
assumes *wf_dbtm t*
shows $\{\} \vdash SubstFormP\ «Var\ i»\ (quot\_dbtm\ t)\ (quot\_dbfm\ db)\ (quot\_dbfm\ (subst\_dbfm\ t\ i\ db))$
⟨*proof*⟩

lemmas *SubstFormP_trivial* = *SubstFormP_trivial_dbfm*[**where** *db*=*trans_fm* [] *A* **for** *A*,
simplified, *folded quot_tm_def quot_fm_def quot_subst_eq*]
lemmas *SubstFormP* = *SubstFormP_dbfm*[*OF wf_dbtm_trans_tm*, **where** *db*=*trans_fm* [] *A* **for** *A*,
simplified, *folded quot_tm_def quot_fm_def quot_subst_eq*]
lemmas *SubstFormP_Zero* = *SubstFormP_dbfm*[*OF wf_dbtm.Zero*, **where** *db*=*trans_fm* [] *A* **for** *A*,
simplified, *folded trans_tm.simps*[*of* []], *folded quot_tm_def quot_fm_def quot_subst_eq*]

lemma *AtomicP_Mem*:
{*TermP x*, *TermP y*} ⊢ *AtomicP* (*Q_Mem x y*)
⟨*proof*⟩

lemma *AtomicP_Eq*:
{*TermP x*, *TermP y*} ⊢ *AtomicP* (*Q_Eq x y*)
⟨*proof*⟩

lemma *SeqFormP_Mem*:
assumes *atom s* ♯ (*k*,*x*,*y*) *atom k* ♯ (*x*,*y*)
shows {*TermP x*, *TermP y*} ⊢ *Ex k* (*Ex s* (*SeqFormP* (*Var s*) (*Var k*) (*Q_Mem x y*)))
⟨*proof*⟩

lemma *SeqFormP_Eq*:
assumes *atom s* ♯ (*k*,*x*,*y*) *atom k* ♯ (*x*,*y*)
shows {*TermP x*, *TermP y*} ⊢ *Ex k* (*Ex s* (*SeqFormP* (*Var s*) (*Var k*) (*Q_Eq x y*)))
⟨*proof*⟩

lemma *FormP_Mem*:
{*TermP x*, *TermP y*} ⊢ *FormP* (*Q_Mem x y*)
⟨*proof*⟩

lemma *FormP_Eq*:
{*TermP x*, *TermP y*} ⊢ *FormP* (*Q_Eq x y*)
⟨*proof*⟩

7.14.6 MakeForm

lemma *MakeFormP_Neg*: {} ⊢ *MakeFormP* (*Q_Neg x*) *x y*
⟨*proof*⟩

lemma *MakeFormP_Disj*: {} ⊢ *MakeFormP* (*Q_Disj x y*) *x y*
⟨*proof*⟩

lemma *MakeFormP_Ex*: {*AbstFormP v Zero t x*} ⊢ *MakeFormP* (*Q_Ex x*) *t y*
⟨*proof*⟩

7.14.7 Negation

lemma *SeqFormP_Neg*:
assumes *atom s* ♯ (*k*,*s1*,*k1*,*x*) *atom k* ♯ (*s1*,*k1*,*x*)
shows {*SeqFormP s1 k1 x*} ⊢ *Ex k* (*Ex s* (*SeqFormP* (*Var s*) (*Var k*) (*Q_Neg x*))) ⟨*proof*⟩
theorem *FormP_Neg*: {*FormP x*} ⊢ *FormP* (*Q_Neg x*)
⟨*proof*⟩

7.14.8 Disjunction

lemma *SeqFormP_Disj*:
assumes *atom s* ♯ (*k*,*s1*,*s2*,*k1*,*k2*,*x*,*y*) *atom k* ♯ (*s1*,*s2*,*k1*,*k2*,*x*,*y*)
shows {*SeqFormP s1 k1 x*, *SeqFormP s2 k2 y*}
⊢ *Ex k* (*Ex s* (*SeqFormP* (*Var s*) (*Var k*) (*Q_Disj x y*))) ⟨*proof*⟩
theorem *FormP_Disj*:
{*FormP x*, *FormP y*} ⊢ *FormP* (*Q_Disj x y*)
⟨*proof*⟩

7.14.9 Existential

lemma *SeqFormP_Ex*:
assumes *atom s* ♯ (*k*,*s1*,*k1*,*x*,*y*,*v*) *atom k* ♯ (*s1*,*k1*,*x*,*y*,*v*)

shows {*SeqFormP s1 k1 x*,*AbstFormP v Zero x y*, *VarP v*} ⊢ *Ex k* (*Ex s* (*SeqFormP* (*Var s*) (*Var k*) (*Q_Ex y*)))
⟨*proof*⟩

theorem *FormP_Ex*: {*FormP t*, *AbstFormP* «*Var i*» *Zero t x*} ⊢ *FormP* (*Q_Ex x*)
⟨*proof*⟩

lemma *FormP_quot_dbfm*:
fixes *A* :: *dbfm*
shows *wf_dbfm A* ⟹ {} ⊢ *FormP* (*quot_dbfm A*)
⟨*proof*⟩

lemma *FormP_quot*:
fixes *A* :: *fm*
shows {} ⊢ *FormP* «*A*»
⟨*proof*⟩

lemma *PfP_I*:
assumes {} ⊢ *PrfP S K A*
shows {} ⊢ *PfP A*
⟨*proof*⟩

lemmas *PfP_Single_I* = *PfP_I*[*of Eats Zero* (*HPair Zero* «*A*») *Zero* **for** *A*]

lemma *PfP_extra*: {} ⊢ *PfP* «*extra_axiom*»
⟨*proof*⟩

lemma *SentP_I*:
assumes *A* ∈ *boolean_axioms*
shows {} ⊢ *SentP* «*A*»
⟨*proof*⟩

lemma *SentP_subst* [*simp*]: (*SentP A*)(*j*::=*w*) = *SentP* (*subst j w A*)
⟨*proof*⟩

theorem *proved_imp_proved_PfP*:
assumes {} ⊢ *α*
shows {} ⊢ *PfP* «*α*»
⟨*proof*⟩

end

Chapter 8

Pseudo-Coding: Section 7 Material

theory *Pseudo_Coding*
imports *II_Prelims*
begin

8.1 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* $V \Longrightarrow a \sharp (V - \{j\}) \longleftrightarrow (a \sharp j \longrightarrow a \sharp V)$
⟨*proof*⟩

lemma *fresh_image_atom* [*simp*]: *finite* $A \Longrightarrow i \sharp$ *atom* ‘ $A \longleftrightarrow i \sharp A$
⟨*proof*⟩

lemma *atom_fresh_star_atom_set_conv*: $[\![$*atom* $i \sharp$ *bs*; *finite bs*$]\!] \Longrightarrow$ *bs* $\sharp*$ *i*
⟨*proof*⟩

lemma *notin_V*:
assumes *p*: *atom* $i \sharp p$ **and** *V*: *finite V atom* ‘ $(p \cdot V) \sharp* V$
shows $i \notin V$ $i \notin p \cdot V$
⟨*proof*⟩

8.2 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* ‘ $(p \cdot Vs) \sharp* Vs$
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* ⊆ *Vs* ⟹ *finite V*
⟨*proof*⟩

lemma *perm_exits_Vs*: *i* ∈ *Vs* ⟹ (*p* · *i*) ∉ *Vs*
⟨*proof*⟩

lemma *atom_fresh_perm*: [*x* ∈ *Vs*; *y* ∈ *Vs*] ⟹ *atom x* $\sharp$ *p* · *y*
⟨*proof*⟩

lemma *fresh_pj*: [*a* $\sharp$ *p*; *j* ∈ *Vs*] ⟹ *a* $\sharp$ *p* · *j*
⟨*proof*⟩

lemma *fresh_Vs*: *a* $\sharp$ *p* ⟹ *a* $\sharp$ *Vs*
⟨*proof*⟩

lemma *fresh_pVs*: *a* $\sharp$ *p* ⟹ *a* $\sharp$ *p* · *Vs*
⟨*proof*⟩

lemma assumes *V* ⊆ *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* ♯ *p atom i'* ♯ (*i*,*p*)
shows *quote_perm* ((*atom i* ⇌ *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* ∉ *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 ⟹
ssubst t (*insert i V*) *F* = (*if i* ∈ *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* $(p \cdot i)$ *♯ t*
shows *atom i ♯ ssubst t V F*
⟨*proof*⟩

lemma *ssubst_vquot_Ex*:
⟦*finite V*; *atom i ♯* $p \cdot V$⟧
$\Longrightarrow$ *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* = {}⟧ $\Longrightarrow$ *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* ∈ $p \cdot V$; *V* ⊆ *Vs*⟧ $\Longrightarrow$ *i* ∈ $p \cdot$ *Vs*
⟨*proof*⟩

8.3 The Main Theorems of Section 7

lemma *SubstTermP_vquot_dbtm*:
assumes *w*: *w* ∈ *Vs* − *V* **and** *V*: *V* ⊆ *Vs V'* = $p \cdot V$
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'* = $p \cdot V$
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'* = $p \cdot V$
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

Chapter 9

Quotations of the Free Variables

theory *Quote*
imports *Pseudo_Coding*
begin

9.1 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.

9.1.1 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 *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*]

9.1.2 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*⟩

9.2 The “special function” itself

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 *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*]

9.2.1 Correctness properties

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*: $\{QuoteP\ t1\ u1,\ QuoteP\ t2\ u2\} \vdash QuoteP\ (Eats\ t1\ t2)\ (Q\_Eats\ u1\ u2)$
⟨*proof*⟩

lemma *exists_QuoteP*:
assumes *j*: $atom\ j \sharp x$ **shows** $\{\} \vdash Ex\ j\ (QuoteP\ x\ (Var\ j))$
⟨*proof*⟩

lemma *QuoteP_imp_ConstP*: $\{\ QuoteP\ x\ y\ \} \vdash ConstP\ y$
⟨*proof*⟩

lemma *SeqQuoteP_imp_QuoteP*: $\{SeqQuoteP\ t\ u\ s\ k\} \vdash QuoteP\ t\ u$
⟨*proof*⟩

lemmas $QuoteP\_I = SeqQuoteP\_imp\_QuoteP\ [THEN\ cut1]$

9.3 The Operator *quote_all*

9.3.1 Definition and basic properties

definition *quote_all* :: [*perm*, *name set*] ⇒ *fm set*
where $quote\_all\ p\ V = \{QuoteP\ (Var\ i)\ (Var\ (p \cdot i)) \mid i.\ i \in V\}$

lemma *quote_all_empty* [*simp*]: $quote\_all\ p\ \{\} = \{\}$
⟨*proof*⟩

lemma *quote_all_insert* [*simp*]:
$quote\_all\ p\ (insert\ i\ V) = insert\ (QuoteP\ (Var\ i)\ (Var\ (p \cdot i)))\ (quote\_all\ p\ V)$
⟨*proof*⟩

lemma *finite_quote_all* [*simp*]: $finite\ V \Longrightarrow finite\ (quote\_all\ p\ V)$
⟨*proof*⟩

lemma *fresh_quote_all* [*simp*]: $finite\ V \Longrightarrow i \sharp quote\_all\ p\ V \longleftrightarrow i \sharp V \wedge i \sharp p{\cdot}V$
⟨*proof*⟩

lemma *fresh_quote_all_mem*: $[\![A \in quote\_all\ p\ V;\ finite\ V;\ i \sharp V;\ i \sharp p \cdot V]\!] \Longrightarrow i \sharp A$
⟨*proof*⟩

lemma *quote_all_perm_eq*:
assumes $finite\ V\ atom\ i \sharp (p,V)\ atom\ i' \sharp (p,V)$
shows $quote\_all\ ((atom\ i \rightleftharpoons atom\ i') + p)\ V = quote\_all\ p\ V$
⟨*proof*⟩

9.3.2 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** $quote\_all\ p\ Us \vdash A$
⟨*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*: {} ⊢ *α IMP β*
and *V*: *V* ⊆ *Vs*
and *s*: *supp α* ⊆ *atom ‘ Vs supp β* ⊆ *atom ‘ Vs*
shows *quote_all p V* ⊢ *PfP* (*ssubst* ⌊*α*⌋ *V V F*) *IMP PfP* (*ssubst* ⌊*β*⌋ *V V F*)
⟨*proof*⟩

Lemma 8.4b

corollary *quote_all_MonPon2_PfP_ssubst*:
assumes *A*: {} ⊢ *α1 IMP α2 IMP β*
and *V*: *V* ⊆ *Vs*
and *s*: *supp α1* ⊆ *atom ‘ Vs supp α2* ⊆ *atom ‘ Vs supp β* ⊆ *atom ‘ Vs*
shows *quote_all p V* ⊢ *PfP* (*ssubst* ⌊*α1*⌋ *V V F*) *IMP PfP* (*ssubst* ⌊*α2*⌋ *V V F*) *IMP PfP* (*ssubst* ⌊*β*⌋ *V V F*)
⟨*proof*⟩

lemma *quote_all_Disj_I1_PfP_ssubst*:
assumes *V* ⊆ *Vs supp α* ⊆ *atom ‘ Vs supp β* ⊆ *atom ‘ Vs*
and *prems*: *H* ⊢ *PfP* (*ssubst* ⌊*α*⌋ *V V F*) *quote_all p V* ⊆ *H*
shows *H* ⊢ *PfP* (*ssubst* ⌊*α OR β*⌋ *V V F*)
⟨*proof*⟩

lemma *quote_all_Disj_I2_PfP_ssubst*:
assumes *V* ⊆ *Vs supp α* ⊆ *atom ‘ Vs supp β* ⊆ *atom ‘ Vs*
and *prems*: *H* ⊢ *PfP* (*ssubst* ⌊*β*⌋ *V V F*) *quote_all p V* ⊆ *H*
shows *H* ⊢ *PfP* (*ssubst* ⌊*α OR β*⌋ *V V F*)
⟨*proof*⟩

lemma *quote_all_Conj_I_PfP_ssubst*:
assumes *V* ⊆ *Vs supp α* ⊆ *atom ‘ Vs supp β* ⊆ *atom ‘ Vs*
and *prems*: *H* ⊢ *PfP* (*ssubst* ⌊*α*⌋ *V V F*) *H* ⊢ *PfP* (*ssubst* ⌊*β*⌋ *V V F*) *quote_all p V* ⊆ *H*
shows *H* ⊢ *PfP* (*ssubst* ⌊*α AND β*⌋ *V V F*)
⟨*proof*⟩

lemma *quote_all_Contra_PfP_ssubst*:
assumes *V* ⊆ *Vs supp α* ⊆ *atom ‘ Vs*
shows *quote_all p V*
⊢ *PfP* (*ssubst* ⌊*α*⌋ *V V F*) *IMP PfP* (*ssubst* ⌊*Neg α*⌋ *V V F*) *IMP PfP* (*ssubst* ⌊*Fls*⌋ *V V F*)
⟨*proof*⟩

lemma *fresh_ssubst_dbtm*: ⟦*atom i* ♯ *p*•*V*; *V* ⊆ *Vs*⟧ ⟹ *atom i* ♯ *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

9.4 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*⟩

9.5 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*⟩

9.6 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: $atom\ `\ (p \cdot V) \sharp* V - p = p$
and V: $finite\ V\ supp\ \alpha \subseteq atom\ `\ V$
and F: $F = make\_F\ V\ p$
shows $insert\ \alpha\ (quote\_all\ p\ V) \vdash PfP\ (ssubst\ \lfloor \alpha \rfloor\ V\ V\ F)$
⟨*proof*⟩

theorem *Provability*:
assumes $Sigma\_fm\ \alpha\ ground\_fm\ \alpha$
shows $\{\alpha\} \vdash PfP\ «\alpha»$
⟨*proof*⟩

end

Chapter 10

Uniqueness Results: Syntactic Relations are Functions

theory *Functions*
imports *Coding_Predicates*
begin

10.0.1 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 $atom\ m \sharp (v,i,t,u,s,k,n,sm,sm',sn,sn')\ atom\ n \sharp (v,i,t,u,s,k,sm,sm',sn,sn')$
$atom\ sm \sharp (v,i,t,u,s,k,sm',sn,sn')\ atom\ sm' \sharp (v,i,t,u,s,k,sn,sn')$
$atom\ sn \sharp (v,i,t,u,s,k,sn')\ atom\ sn' \sharp (v,i,t,u,s,k)$
shows $\{\ SeqStTermP\ v\ i\ t\ u\ s\ k\ \}$
$\vdash ((t\ EQ\ v\ AND\ u\ EQ\ i)\ OR$
$((IndP\ t\ OR\ t\ NEQ\ v)\ AND\ u\ EQ\ t))\ OR$
$Ex\ m\ (Ex\ n\ (Ex\ sm\ (Ex\ sm'\ (Ex\ sn\ (Ex\ sn'\ (Var\ m\ IN\ k\ AND\ Var\ n\ IN\ k\ AND$
$SeqStTermP\ v\ i\ (Var\ sm)\ (Var\ sm')\ s\ (Var\ m)\ AND$
$SeqStTermP\ v\ i\ (Var\ sn)\ (Var\ sn')\ s\ (Var\ n)\ AND$
$t\ EQ\ Q\_Eats\ (Var\ sm)\ (Var\ sn)\ AND$
$u\ EQ\ Q\_Eats\ (Var\ sm')\ (Var\ sn')))))))$
⟨*proof*⟩

lemma *SeqStTermP_unique*: $\{SeqStTermP\ v\ a\ t\ u\ s\ kk,\ SeqStTermP\ v\ a\ t\ u'\ s'\ kk'\} \vdash u'\ EQ\ u$
⟨*proof*⟩

theorem *SubstTermP_unique*: $\{SubstTermP\ v\ tm\ t\ u,\ SubstTermP\ v\ tm\ t\ u'\} \vdash u'\ EQ\ u$
⟨*proof*⟩

10.0.2 *SubstAtomicP*

lemma *SubstTermP_eq*:
$[\![H \vdash SubstTermP\ v\ tm\ x\ z;\ insert\ (SubstTermP\ v\ tm\ y\ z)\ H \vdash A]\!] \Longrightarrow insert\ (x\ EQ\ y)\ H \vdash A$
⟨*proof*⟩

lemma *SubstAtomicP_unique*: $\{SubstAtomicP\ v\ tm\ x\ y,\ SubstAtomicP\ v\ tm\ x\ y'\} \vdash y'\ EQ\ y$
⟨*proof*⟩

10.0.3 *SeqSubstFormP*

lemma *SeqSubstFormP_lemma*:
assumes $atom\ m \sharp (v,u,x,y,s,k,n,sm,sm',sn,sn')\ atom\ n \sharp (v,u,x,y,s,k,sm,sm',sn,sn')$
$atom\ sm \sharp (v,u,x,y,s,k,sm',sn,sn')\ \ atom\ sm' \sharp (v,u,x,y,s,k,sn,sn')$
$atom\ sn \sharp (v,u,x,y,s,k,sn')\qquad atom\ sn' \sharp (v,u,x,y,s,k)$
shows { $SeqSubstFormP\ v\ u\ x\ y\ s\ k$ }
$\vdash SubstAtomicP\ v\ u\ x\ y\ \ OR$
$Ex\ m\ (Ex\ n\ (Ex\ sm\ (Ex\ sm'\ (Ex\ sn\ (Ex\ sn'\ (Var\ m\ IN\ k\ AND\ Var\ n\ IN\ k\ AND$
$SeqSubstFormP\ v\ u\ (Var\ sm)\ (Var\ sm')\ s\ (Var\ m)\ AND$
$SeqSubstFormP\ v\ u\ (Var\ sn)\ (Var\ sn')\ s\ (Var\ n)\ AND$
$(((x\ EQ\ Q\_Disj\ (Var\ sm)\ (Var\ sn)\ AND\ y\ EQ\ Q\_Disj\ (Var\ sm')\ (Var\ sn'))\ OR$
$(x\ EQ\ Q\_Neg\ (Var\ sm)\ AND\ y\ EQ\ Q\_Neg\ (Var\ sm'))\ OR$
$(x\ EQ\ Q\_Ex\ (Var\ sm)\ AND\ y\ EQ\ Q\_Ex\ (Var\ sm')))))))))$
⟨*proof*⟩

lemma
shows *Neg_SubstAtomicP_Fls*: $\{y\ EQ\ Q\_Neg\ z,\ \ SubstAtomicP\ v\ tm\ y\ y'\} \vdash Fls$ (**is** *?thesis1*)
and *Disj_SubstAtomicP_Fls*: $\{y\ EQ\ Q\_Disj\ z\ w,\ SubstAtomicP\ v\ tm\ y\ y'\} \vdash Fls$ (**is** *?thesis2*)
and *Ex_SubstAtomicP_Fls*: $\{y\ EQ\ Q\_Ex\ z,\ \ SubstAtomicP\ v\ tm\ y\ y'\} \vdash Fls$ (**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*⟩

10.0.4 *SubstFormP*

theorem *SubstFormP_unique*: $\{SubstFormP\ v\ tm\ x\ y,\ SubstFormP\ v\ tm\ x\ y'\} \vdash y'\ EQ\ y$
⟨*proof*⟩

end

Chapter 11

Section 6 Material and Gödel's First Incompleteness Theorem

theory *Goedel_I*
imports *Pf_Predicates Functions II_Prelims*
begin

11.1 The Function W and Lemma 6.1

11.1.1 Predicate form, defined on sequences

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 *SeqWRP_sf* [*iff*]: *Sigma_fm* (*SeqWRP s k y*) (**is** *?thsf*)
    and *SeqWRP_imp_OrdP*: {*SeqWRP s k t*} ⊢ *OrdP k* (**is** *?thOrd*)
    and *SeqWRP_LstSeqP*: {*SeqWRP s k t*} ⊢ *LstSeqP s k t* (**is** *?thlstseq*)
⟨*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* ⊢ *s EQ s′* **and** *H* ⊢ *k EQ k′* **and** *H* ⊢ *y EQ y′*
  shows *H* ⊢ *SeqWRP s k y IFF SeqWRP s′ k′ y′*
  ⟨*proof*⟩

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

11.1.2 Predicate form of W

nominal_function *WRP* :: *tm* ⇒ *tm* ⇒ *fm*
  where ⟦*atom s* ♯ (*x*,*y*)⟧ ⟹
    WRP x y = *Ex s* (*SeqWRP* (*Var s*) *x y*)
  ⟨*proof*⟩

nominal_termination (*eqvt*)
  ⟨*proof*⟩

lemma
  shows *WRP_fresh_iff* [*simp*]: *a* ♯ *WRP x y* ⟷ *a* ♯ *x* ∧ *a* ♯ *y* (**is** *?thesis1*)
    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* ⊢ *t EQ t′* ⟹ *H* ⊢ *u EQ u′* ⟹ *H* ⊢ *WRP t u IFF WRP t′ u′*
  ⟨*proof*⟩

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

lemma *ground_WRP* [*simp*]: *ground_fm* (*WRP x y*) ⟷ *ground x* ∧ *ground y*
  ⟨*proof*⟩

lemma *SeqWRP_Zero*: {} ⊢ *SyntaxN.Ex s* (*SeqWRP* (*Var s*) *Zero Zero*)
⟨*proof*⟩

lemma *WRP_Zero*: {} ⊢ *WRP Zero Zero*
  ⟨*proof*⟩

lemma *SeqWRP_HPair_Zero_Zero*: {*SeqWRP s k y*} ⊢ *HPair Zero Zero IN s*
⟨*proof*⟩

lemma *SeqWRP_Succ*:
  assumes *atom s* ♯ (*s1*,*k1*,*y*)
  shows {*SeqWRP s1 k1 y*} ⊢ *SyntaxN.Ex s* (*SeqWRP* (*Var s*) (*SUCC k1*) (*Q_Succ y*))
⟨*proof*⟩

lemma *WRP_Succ*: {*OrdP i*, *WRP i y*} ⊢ *WRP* (*SUCC i*) (*Q_Succ y*)
⟨*proof*⟩

lemma *WRP*: {} ⊢ *WRP* (*ORD_OF i*) «*ORD_OF i*»
  ⟨*proof*⟩

lemma *prove_WRP*: {} ⊢ *WRP* «*Var x*» ««*Var x*»»
  ⟨*proof*⟩

11.1.3 Proving that these relations are functions

lemma *SeqWRP_Zero_E*:
  assumes *insert* (*y EQ Zero*) *H* ⊢ *A* *H* ⊢ *k EQ Zero*
  shows *insert* (*SeqWRP s k y*) *H* ⊢ *A*
⟨*proof*⟩

lemma *SeqWRP_SUCC_lemma*:
  assumes *y′*: *atom y′* ♯ (*s*,*k*,*y*)
  shows {*SeqWRP s* (*SUCC k*) *y*} ⊢ *Ex y′* (*SeqWRP s k* (*Var y′*) *AND y EQ Q_Succ* (*Var y′*))

⟨*proof*⟩

lemma *SeqWRP_SUCC_E*:
assumes *y′*: *atom y′* ♯ (*s*,*k*,*y*) **and** *k′*: *H* ⊢ *k′ EQ* (*SUCC k*)
shows *insert* (*SeqWRP s k′ y*) *H* ⊢ *Ex y′* (*SeqWRP s k* (*Var y′*) *AND y EQ Q_Succ* (*Var y′*))
⟨*proof*⟩

lemma *SeqWRP_unique*: {*OrdP x*, *SeqWRP s x y*, *SeqWRP s′ x y′*} ⊢ *y′ EQ y*
⟨*proof*⟩

theorem *WRP_unique*: {*OrdP x*, *WRP x y*, *WRP x y′*} ⊢ *y′ EQ y*
⟨*proof*⟩

11.2 The Function HF and Lemma 6.2

11.2.1 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 ♯ *SeqHRP x x′ s k* ⟷ *a* ♯ *x* ∧ *a* ♯ *x′* ∧ *a* ♯ *s* ∧ *a* ♯ *k* (**is** *?thesis1*)
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*)
and *SeqHRP_imp_LstSeqP*: { *SeqHRP x y s k* } ⊢ *LstSeqP s k* (*HPair x y*) (**is** *?thlstseq*)
⟨*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* ⊢ *x EQ x′* **and** *H* ⊢ *y EQ y′ H* ⊢ *s EQ s′* **and** *H* ⊢ *k EQ k′*
shows *H* ⊢ *SeqHRP x y s k IFF SeqHRP x′ y′ s′ k′*
⟨*proof*⟩

11.2.2 Defining the syntax: main predicate

nominal_function *HRP* :: *tm* ⇒ *tm* ⇒ *fm*
  where ⟦*atom s* ♯ (*x*,*x*′,*k*); *atom k* ♯ (*x*,*x*′)⟧ ⟹
      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* ♯ *HRP x x*′ ⟷ *a* ♯ *x* ∧ *a* ♯ *x*′ (**is** *?thesis1*)
   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*⟩

11.2.3 Proving that these relations are functions

lemma *SeqHRP_lemma*:
  assumes *atom m* ♯ (*x*,*x*′,*s*,*k*,*n*,*sm*,*sm*′,*sn*,*sn*′) *atom n* ♯ (*x*,*x*′,*s*,*k*,*sm*,*sm*′,*sn*,*sn*′)
      atom sm ♯ (*x*,*x*′,*s*,*k*,*sm*′,*sn*,*sn*′) *atom sm*′ ♯ (*x*,*x*′,*s*,*k*,*sn*,*sn*′)
      atom sn ♯ (*x*,*x*′,*s*,*k*,*sn*′) *atom sn*′ ♯ (*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*⟩

lemma *HRP_ORD_OF*: {} ⊢ *HRP* (*ORD_OF i*) «*ORD_OF i*»
⟨*proof*⟩

lemma *SeqHRP_HPair*:
  assumes *atom s* ♯ (*k*,*s1*,*s2*,*k1*,*k2*,*x*,*y*,*x*′,*y*′) *atom k* ♯ (*s1*,*s2*,*k1*,*k2*,*x*,*y*,*x*′,*y*′)
  shows {*SeqHRP x x*′ *s1 k1*,
       SeqHRP y y′ *s2 k2*}
      ⊢ *Ex s* (*Ex k* (*SeqHRP* (*HPair x y*) (*Q_HPair x*′ *y*′) (*Var s*) (*Var k*))) ⟨*proof*⟩
lemma *HRP_HPair*: {*HRP x x*′, *HRP y y*′} ⊢ *HRP* (*HPair x y*) (*Q_HPair x*′ *y*′)
⟨*proof*⟩

lemma *HRP_HPair_quot*: {*HRP x* «*x*», *HRP y* «*y*»} ⊢ *HRP* (*HPair x y*) «*HPair x y*»
  ⟨*proof*⟩

lemma *prove_HRP_coding_tm*: **fixes** *t*::*tm* **shows** *coding_tm t* ⟹ {} ⊢ *HRP t* «*t*»
  ⟨*proof*⟩

lemmas *prove_HRP* = *prove_HRP_coding_tm*[*OF quot_fm_coding*]

11.3 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* ♯ *KRP v x x'* ⟷ *a* ♯ *v* ∧ *a* ♯ *x* ∧ *a* ♯ *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*: {} ⊢ *SubstFormP* «*Var i*» ««*A*»» «*A*» «*A*(*i*::=«*A*»)»
  ⟨*proof*⟩

lemma *prove_KRP*: {} ⊢ *KRP* «*Var i*» «*A*» «*A*(*i*::=«*A*»)»
  ⟨*proof*⟩

lemma *KRP_unique*: {*KRP v x y*, *KRP v x y'*} ⊢ *y' EQ y*
⟨*proof*⟩

lemma *KRP_subst_fm*: {*KRP* «*Var i*» «*β*» (*Var j*)} ⊢ *Var j EQ* «*β*(*i*::=«*β*»)»
  ⟨*proof*⟩

end

Chapter 12

The Instantiation

definition *Fvars t* = {*a* :: *name*. ¬ *atom a* ♯ *t*}

lemma *Fvars_tm_simps*[*simp*]:
Fvars Zero = {}
Fvars (*Var a*) = {*a*}
Fvars (*Eats x y*) = *Fvars x* ∪ *Fvars y*
⟨*proof*⟩

lemma *finite_Fvars_tm*[*simp*]:
fixes *t* :: *tm*
shows *finite* (*Fvars t*)
⟨*proof*⟩

lemma *Fvars_fm_simps*[*simp*]:
Fvars (*x IN y*) = *Fvars x* ∪ *Fvars y*
Fvars (*x EQ y*) = *Fvars x* ∪ *Fvars y*
Fvars (*A OR B*) = *Fvars A* ∪ *Fvars B*
Fvars (*A AND B*) = *Fvars A* ∪ *Fvars B*
Fvars (*A IMP B*) = *Fvars A* ∪ *Fvars B*
Fvars Fls = {}
Fvars (*Neg A*) = *Fvars A*
Fvars (*Ex a A*) = *Fvars A* − {*a*}
Fvars (*All a A*) = *Fvars A* − {*a*}
⟨*proof*⟩

lemma *finite_Fvars_fm*[*simp*]:
fixes *A* :: *fm*
shows *finite* (*Fvars A*)
⟨*proof*⟩

lemma *subst_tm_subst_tm*[*simp*]:
x ≠ *y* ⟹ *atom x* ♯ *u* ⟹ *subst y u* (*subst x t v*) = *subst x* (*subst y u t*) (*subst y u v*)
⟨*proof*⟩

lemma *subst_fm_subst_fm*[*simp*]:
x ≠ *y* ⟹ *atom x* ♯ *u* ⟹ (*A*(*x*::=*t*))(*y*::=*u*) = (*A*(*y*::=*u*))(*x*::=*subst y u t*)
⟨*proof*⟩

lemma *Fvars_ground_aux*: *Fvars t* ⊆ *B* ⟹ *ground_aux t* (*atom* ‘ *B*)
⟨*proof*⟩

lemma *ground_Fvars*: *ground t* $\longleftrightarrow$ *Fvars t* = {}
⟨*proof*⟩

lemma *Fvars_ground_fm_aux*: *Fvars A* $\subseteq$ *B* $\Longrightarrow$ *ground_fm_aux A* (*atom* ‘ *B*)
⟨*proof*⟩

lemma *ground_fm_Fvars*: *ground_fm A* $\longleftrightarrow$ *Fvars A* = {}
⟨*proof*⟩

interpretation *Generic_Syntax* **where**
var = *UNIV* :: *name set*
and *trm* = *UNIV* :: *tm set*
and *fmla* = *UNIV* :: *fm set*
and *Var* = *Var*
and *FvarsT* = *Fvars*
and *substT* = λ*t u x*. *subst x u t*
and *Fvars* = *Fvars*
and *subst* = λ*A u x*. *subst_fm A x u*
⟨*proof*⟩

lemma *coding_tm_Fvars_empty*[*simp*]: *coding_tm t* $\Longrightarrow$ *Fvars t* = {}
⟨*proof*⟩

lemma *Fvars_empty_ground*[*simp*]: *Fvars t* = {} $\Longrightarrow$ *ground t*
⟨*proof*⟩

interpretation *Syntax_with_Numerals* **where**
var = *UNIV* :: *name set*
and *trm* = *UNIV* :: *tm set*
and *fmla* = *UNIV* :: *fm set*
and *num* = {*t*. *ground t*}
and *Var* = *Var*
and *FvarsT* = *Fvars*
and *substT* = λ*t u x*. *subst x u t*
and *Fvars* = *Fvars*
and *subst* = λ*A u x*. *subst_fm A x u*
⟨*proof*⟩

declare *FvarsT_num*[*simp del*]

interpretation *Deduct2_with_False* **where**
var = *UNIV* :: *name set*
and *trm* = *UNIV* :: *tm set*
and *fmla* = *UNIV* :: *fm set*
and *num* = {*t*. *ground t*}
and *Var* = *Var*
and *FvarsT* = *Fvars*
and *substT* = λ*t u x*. *subst x u t*
and *Fvars* = *Fvars*
and *subst* = λ*A u x*. *subst_fm A x u*
and *eql* = (*EQ*)
and *cnj* = (*AND*)
and *imp* = (*IMP*)
and *all* = *All*
and *exi* = *Ex*
and *fls* = *Fls*
and *prv* = ($\vdash$) {}
and *bprv* = ($\vdash$) {}

⟨*proof*⟩

interpretation *HBL1* **where**
  var = *UNIV* :: *name set*
and *trm* = *UNIV* :: *tm set*
and *fmla* = *UNIV* :: *fm set*
and *num* = {*t*. *ground t*}
and *Var* = *Var*
and *FvarsT* = *Fvars*
and *substT* = λ*t u x*. *subst x u t*
and *Fvars* = *Fvars*
and *subst* = λ*A u x*. *subst_fm A x u*
and *eql* = (*EQ*)
and *cnj* = (*AND*)
and *imp* = (*IMP*)
and *all* = *All*
and *exi* = *Ex*
and *prv* = (⊢) {}
and *bprv* = (⊢) {}
and *enc* = *quot*
and *P* = *PfP* (*Var xx*)
⟨*proof*⟩

interpretation *Goedel_Form* **where**
  var = *UNIV* :: *name set*
and *trm* = *UNIV* :: *tm set*
and *fmla* = *UNIV* :: *fm set*
and *num* = {*t*. *ground t*}
and *Var* = *Var*
and *FvarsT* = *Fvars*
and *substT* = λ*t u x*. *subst x u t*
and *Fvars* = *Fvars*
and *subst* = λ*A u x*. *subst_fm A x u*
and *eql* = (*EQ*)
and *cnj* = (*AND*)
and *imp* = (*IMP*)
and *all* = *All*
and *exi* = *Ex*
and *fls* = *Fls*
and *prv* = (⊢) {}
and *bprv* = (⊢) {}
and *enc* = *quot*
and *S* = *KRP* (*quot* (*Var xx*)) (*Var xx*) (*Var yy*)
and *P* = *PfP* (*Var xx*)
⟨*proof*⟩

interpretation *g2*: *Goedel_Second_Assumptions* **where**
  var = *UNIV* :: *name set*
and *trm* = *UNIV* :: *tm set*
and *fmla* = *UNIV* :: *fm set*
and *num* = {*t*. *ground t*}
and *Var* = *Var*
and *FvarsT* = *Fvars*
and *substT* = λ*t u x*. *subst x u t*
and *Fvars* = *Fvars*
and *subst* = λ*A u x*. *subst_fm A x u*
and *eql* = (*EQ*)
and *cnj* = (*AND*)

and $imp = (IMP)$
and $all = All$
and $exi = Ex$
and $fls = Fls$
and $prv = (\vdash)\ \{\}$
and $bprv = (\vdash)\ \{\}$
and $enc = quot$
and $S = KRP\ (quot\ (Var\ xx))\ (Var\ xx)\ (Var\ yy)$
and $P = PfP\ (Var\ xx)$
⟨*proof*⟩

theorem $\neg\ \{\} \vdash Fls \Longrightarrow \neg\ \{\} \vdash neg\ (PfP\ (quot\ Fls))$
⟨*proof*⟩