Gödel's Incompleteness Theorems

Lawrence C. Paulson

July 22, 2014

Abstract

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

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
    by *default* (*simp-all add*: *permute-hf-def*)
end

instance *hf* :: *pure*
  proof qed (*rule permute-hf-def*)

atom-decl *name*

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

lemma *supp-name* [*simp*]: **fixes** *i*::*name* **shows** *supp* $i = \{atom\ i\}$
  by (*rule supp-at-base*)

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*)
by (*auto simp*: *eqvt-def subst-graph-aux-def*) (*metis tm.strong-exhaust*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma *fresh-subst-if* [*simp*]:
  $j \sharp subst\ i\ x\ t \longleftrightarrow (atom\ i \sharp t \wedge j \sharp t) \vee (j \sharp x \wedge (j \sharp t \vee j = atom\ i))$
  by (*induct t rule*: *tm.induct*) (*auto simp*: *fresh-at-base*)

lemma *forget-subst-tm* [*simp*]: $atom\ a \sharp tm \Longrightarrow subst\ a\ x\ tm = tm$
  by (*induct tm rule*: *tm.induct*) (*simp-all add*: *fresh-at-base*)

lemma *subst-tm-id* [*simp*]: *subst a* (*Var a*) *tm* = *tm*
  by (*induct tm rule*: *tm.induct*) *simp-all*

lemma *subst-tm-commute* [*simp*]:
  $atom\ j \sharp tm \Longrightarrow subst\ j\ u\ (subst\ i\ t\ tm) = subst\ i\ (subst\ j\ u\ t)\ tm$
  by (*induct tm rule*: *tm.induct*) (*auto simp*: *fresh-Pair*)

lemma *subst-tm-commute2* [*simp*]:
  $atom\ j \sharp t \Longrightarrow atom\ i \sharp u \Longrightarrow i \neq j \Longrightarrow subst\ j\ u\ (subst\ i\ t\ tm) = subst\ i\ t\ (subst\ j\ u\ tm)$
  by (*induct tm rule*: *tm.induct*) *auto*

lemma *repeat-subst-tm* [*simp*]: *subst i u* (*subst i t tm*) = *subst i* (*subst i u t*) *tm*
  by (*induct tm rule*: *tm.induct*) *auto*

nominal-function *subst-fm* :: *fm* ⇒ *name* ⇒ *tm* ⇒ *fm* (-′(-::=-′) [*1000*, *0*, *0*] *200*)
  where
  Mem: (*Mem t u*)(*i*::=*x*) = *Mem* (*subst i x t*) (*subst i x u*)
| *Eq*: (*Eq t u*)(*i*::=*x*) = *Eq* (*subst i x t*) (*subst i x u*)
| *Disj*: (*Disj A B*)(*i*::=*x*) = *Disj* (*A*(*i*::=*x*)) (*B*(*i*::=*x*))

| *Neg*: $(Neg\ A)(i{::=}x) \quad = Neg\ \ (A(i{::=}x))$
| *Ex*: $atom\ j\ \sharp\ (i,\ x) \Longrightarrow (Ex\ j\ A)(i{::=}x) = Ex\ j\ \ (A(i{::=}x))$
apply (*simp add*: *eqvt-def subst-fm-graph-aux-def*)
apply *auto* [*16*]
apply (*rule-tac* $y{=}a$ **and** $c{=}(aa,\ b)$ **in** *fm.strong-exhaust*)
apply (*auto simp*: *eqvt-at-def fresh-star-def fresh-Pair fresh-at-base*)
apply (*metis flip-at-base-simps*(*3*) *flip-fresh-fresh*)
done

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma *size-subst-fm* [*simp*]: $size\ (A(i{::=}x)) = size\ A$
  by (*nominal-induct A avoiding*: *i x rule*: *fm.strong-induct*) *auto*

lemma *forget-subst-fm* [*simp*]: $atom\ a\ \sharp\ A \Longrightarrow A(a{::=}x) = A$
 by (*nominal-induct A avoiding*: *a x rule*: *fm.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-fm-id* [*simp*]: $A(a{::=}Var\ a) = A$
 by (*nominal-induct A avoiding*: *a rule*: *fm.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *fresh-subst-fm-if* [*simp*]:
 $j\ \sharp\ (A(i{::=}x)) \longleftrightarrow (atom\ i\ \sharp\ A \wedge j\ \sharp\ A) \vee (j\ \sharp\ x \wedge (j\ \sharp\ A \vee j = atom\ i))$
 by (*nominal-induct A avoiding*: *i x rule*: *fm.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-fm-commute* [*simp*]:
 $atom\ j\ \sharp\ A \Longrightarrow (A(i{::=}t))(j{::=}u) = A(i ::= subst\ j\ u\ t)$
 by (*nominal-induct A avoiding*: *i j t u rule*: *fm.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *repeat-subst-fm* [*simp*]: $(A(i{::=}t))(i{::=}u) = A(i ::= subst\ i\ u\ t)$
  by (*nominal-induct A avoiding*: *i t u rule*: *fm.strong-induct*) *auto*

lemma *subst-fm-Ex-with-renaming*:
 $atom\ i'\ \sharp\ (A,\ i,\ j,\ t) \Longrightarrow (Ex\ i\ A)(j ::= t) = Ex\ i'\ (((i \leftrightarrow i') \cdot A)(j ::= t))$
 by (*rule subst* [*of* $Ex\ i'\ ((i \leftrightarrow i') \cdot A)$ $Ex\ i\ A$])
  (*auto simp*: *Abs1-eq-iff flip-def swap-commute*)

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

simproc-setup *subst-fm-renaming* ((*Ex i A*)(*j ::= t*)) = ⟪ *fn - => fn ctxt => fn ctrm =>*
 let
  val - \$ (- \$ *i* \$ *A*) \$ *j* \$ *t* = *term-of ctrm*

  val atoms = *Simplifier.prems-of ctxt*
   |> *map-filter* (*fn thm => case Thm.prop-of thm of*
     - \$ (*Const* (@{*const-name fresh*}, -) \$ *atm* \$ -) => *SOME* (*atm*) | - => *NONE*)
   |> *distinct* (*op*=)

```
fun get-thm atm =
  let
    val goal = HOLogic.mk-Trueprop (mk-fresh atm (HOLogic.mk-tuple [A, i,
j, t]))
  in
    SOME ((Goal.prove ctxt [] [] goal (K (asm-full-simp-tac ctxt 1)))
      RS @{thm subst-fm-Ex-with-renaming} RS eq-reflection)
    handle ERROR - => NONE
  end
in
  get-first get-thm atoms
end
⟫
```

1.1.4 Semantics

definition *e0* :: (*name*, *hf*) *finfun* — the null environment
  where *e0* ≡ *finfun-const 0*

nominal-function *eval-tm* :: (*name*, *hf*) *finfun* ⇒ *tm* ⇒ *hf*
  where
   eval-tm e Zero = *0*
 | *eval-tm e* (*Var k*) = *finfun-apply e k*
 | *eval-tm e* (*Eats t u*) = *eval-tm e t* ◃ *eval-tm e u*
by (*auto simp*: *eqvt-def eval-tm-graph-aux-def*) (*metis tm.strong-exhaust*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

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

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

nominal-function *eval-fm* :: (*name*, *hf*) *finfun* ⇒ *fm* ⇒ *bool*
  where
   eval-fm e (*t IN u*) ⟷ ⟦*t*⟧*e* ∈ ⟦*u*⟧*e*
 | *eval-fm e* (*t EQ u*) ⟷ ⟦*t*⟧*e* = ⟦*u*⟧*e*
 | *eval-fm e* (*A OR B*) ⟷ *eval-fm e A* ∨ *eval-fm e B*
 | *eval-fm e* (*Neg A*) ⟷ (~ *eval-fm e A*)
 | *atom k* ♯ *e* ⟹ *eval-fm e* (*Ex k A*) ⟷ (∃ *x*. *eval-fm* (*finfun-update e k x*) *A*)
apply(*simp add*: *eqvt-def eval-fm-graph-aux-def*)
apply(*auto del*: *iffI*)[*16*]
apply(*rule-tac y*=*b* **and** *c*=(*a*) **in** *fm.strong-exhaust*)
apply(*auto simp*: *fresh-star-def*)[*5*]
using [[*simproc del*: *alpha-lst*]] **apply** *clarsimp*
apply(*erule-tac c*=(*ea*) **in** *Abs-lst1-fcb2′*)

apply(*rule pure-fresh*)
apply(*simp add*: *fresh-star-def*)
apply (*simp-all add*: *eqvt-at-def*)
apply (*simp-all add*: *perm-supp-eq*)
done

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma *eval-tm-rename*:
  assumes $atom\ k' \sharp t$
  shows $[\![t]\!](finfun\text{-}update\ e\ k\ x) = [\![(k' \leftrightarrow k) \bullet t]\!](finfun\text{-}update\ e\ k'\ x)$
using *assms*
by (*induct t rule*: *tm.induct*) (*auto simp*: *permute-flip-at*)

lemma *eval-fm-rename*:
  assumes $atom\ k' \sharp A$
  shows $eval\text{-}fm\ (finfun\text{-}update\ e\ k\ x)\ A = eval\text{-}fm\ (finfun\text{-}update\ e\ k'\ x)\ ((k' \leftrightarrow k) \bullet A)$
using *assms*
apply (*nominal-induct A avoiding*: $e\ k\ k'\ x$ *rule*: *fm.strong-induct*)
apply (*simp-all add*: *eval-tm-rename*[*symmetric*], *metis*)
apply (*simp add*: *fresh-finfun-update fresh-at-base finfun-update-twist*)
done

lemma *better-ex-eval-fm*[*simp*]:
  $eval\text{-}fm\ e\ (Ex\ k\ A) \longleftrightarrow (\exists x.\ eval\text{-}fm\ (finfun\text{-}update\ e\ k\ x)\ A)$
proof −
  obtain k'::*name* **where** k': $atom\ k' \sharp (k,\ e,\ A)$
    by (*rule obtain-fresh*)
  then have *eq*: $Ex\ k'\ ((k' \leftrightarrow k) \bullet A) = Ex\ k\ A$
    by (*simp add*: *Abs1-eq-iff flip-def*)
  have $eval\text{-}fm\ e\ (Ex\ k'\ ((k' \leftrightarrow k) \bullet A)) = (\exists x.\ eval\text{-}fm\ (finfun\text{-}update\ e\ k'\ x)\ ((k' \leftrightarrow k) \bullet A))$
    using k' **by** *simp*
  also have ... $= (\exists x.\ eval\text{-}fm\ (finfun\text{-}update\ e\ k\ x)\ A)$
    by (*metis eval-fm-rename* k' *fresh-Pair*)
  finally show *?thesis*
    by (*metis eq*)
qed

lemma *forget-eval-tm* [*simp*]: $atom\ i \sharp t \Longrightarrow [\![t]\!](finfun\text{-}update\ e\ i\ x) = [\![t]\!]e$
  by (*induct t rule*: *tm.induct*) (*simp-all add*: *fresh-at-base*)

lemma *forget-eval-fm* [*simp*]:
  $atom\ k \sharp A \Longrightarrow eval\text{-}fm\ (finfun\text{-}update\ e\ k\ x)\ A = eval\text{-}fm\ e\ A$
  by (*nominal-induct A avoiding*: $k\ e$ *rule*: *fm.strong-induct*)
    (*simp-all add*: *fresh-at-base finfun-update-twist*)

lemma *eval-subst-tm*: $[\![subst\ i\ t\ u]\!]e = [\![u]\!](finfun\text{-}update\ e\ i\ [\![t]\!]e)$
by (*induct u rule*: *tm.induct*) (*auto*)

lemma *eval-subst-fm*: $eval\text{-}fm\ e\ (fm(i::=\ t)) = eval\text{-}fm\ (finfun\text{-}update\ e\ i\ [\![t]\!]e)\ fm$
by (*nominal-induct fm avoiding*: *i t e rule*: *fm.strong-induct*)
(*simp-all add*: *eval-subst-tm finfun-update-twist fresh-at-base*)

1.1.5 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*]: $(p \cdot HPair\ a\ b) = HPair\ (p \cdot a)\ (p \cdot b)$
by (*auto simp*: *HPair-def*)

lemma *fresh-HPair* [*simp*]: $x \sharp HPair\ a\ b \longleftrightarrow (x \sharp a \wedge x \sharp b)$
by (*auto simp*: *HPair-def*)

lemma *HPair-injective-iff* [*iff*]: $HPair\ a\ b = HPair\ a'\ b' \longleftrightarrow (a = a' \wedge b = b')$
by (*auto simp*: *HPair-def*)

lemma *subst-tm-HPair* [*simp*]: $subst\ i\ x\ (HPair\ a\ b) = HPair\ (subst\ i\ x\ a)\ (subst\ i\ x\ b)$
by (*auto simp*: *HPair-def*)

lemma *eval-tm-HPair* [*simp*]: $[\![HPair\ a\ b]\!]e = hpair\ [\![a]\!]e\ [\![b]\!]e$
by (*auto simp*: *HPair-def hpair-def*)

Ordinals

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

fun *ORD-OF* :: $nat \Rightarrow tm$
where
$ORD\text{-}OF\ 0 = Zero$
$|\ ORD\text{-}OF\ (Suc\ k) = SUCC\ (ORD\text{-}OF\ k)$

lemma *eval-tm-SUCC* [*simp*]: $[\![SUCC\ t]\!]e = succ\ [\![t]\!]e$
by (*simp add*: *SUCC-def succ-def*)

lemma *SUCC-fresh-iff* [*simp*]: $a \sharp SUCC\ t \longleftrightarrow a \sharp t$
by (*simp add*: *SUCC-def*)

lemma *SUCC-eqvt* [*eqvt*]: $(p \cdot SUCC\ a) = SUCC\ (p \cdot a)$
by (*simp add*: *SUCC-def*)

lemma *SUCC-subst* [*simp*]: *subst i t* (*SUCC k*) = *SUCC* (*subst i t k*)
by (*simp add*: *SUCC-def*)

lemma *eval-tm-ORD-OF* [*simp*]: ⟦*ORD-OF n*⟧*e* = *ord-of n*
by (*induct n*) *auto*

lemma *ORD-OF-fresh* [*simp*]: *a* ♯ *ORD-OF n*
by (*induct n*) (*auto simp*: *SUCC-def*)

lemma *ORD-OF-eqvt* [*eqvt*]: (*p* • *ORD-OF n*) = *ORD-OF* (*p* • *n*)
by (*induct n*) (*auto simp*: *permute-pure SUCC-eqvt*)

1.1.6 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*)
by (*simp add*: *Conj-def*)

lemma *fresh-Conj* [*simp*]: *a* ♯ *A AND B* ⟷ (*a* ♯ *A* ∧ *a* ♯ *B*)
by (*auto simp*: *Conj-def*)

lemma *supp-Conj* [*simp*]: *supp* (*A AND B*) = *supp A* ∪ *supp B*
by (*auto simp*: *Conj-def*)

lemma *size-Conj* [*simp*]: *size* (*A AND B*) = *size A* + *size B* + *4*
by (*simp add*: *Conj-def*)

lemma *Conj-injective-iff* [*iff*]: (*A AND B*) = (*A′ AND B′*) ⟷ (*A* = *A′* ∧ *B* = *B′*)
by (*auto simp*: *Conj-def*)

lemma *subst-fm-Conj* [*simp*]: (*A AND B*)(*i*::=*x*) = (*A*(*i*::=*x*)) *AND* (*B*(*i*::=*x*))
by (*auto simp*: *Conj-def*)

lemma *eval-fm-Conj* [*simp*]: *eval-fm e* (*Conj A B*) $\longleftrightarrow$ (*eval-fm e A* $\wedge$ *eval-fm e B*)
  by (*auto simp*: *Conj-def*)

If and only if

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

lemma *Iff-eqvt* [*eqvt*]: *p* $\cdot$ (*A IFF B*) = (*p* $\cdot$ *A*) *IFF* (*p* $\cdot$ *B*)
  by (*simp add*: *Iff-def*)

lemma *fresh-Iff* [*simp*]: *a* $\sharp$ *A IFF B* $\longleftrightarrow$ (*a* $\sharp$ *A* $\wedge$ *a* $\sharp$ *B*)
  by (*auto simp*: *Conj-def Iff-def*)

lemma *size-Iff* [*simp*]: *size* (*A IFF B*) = *2*$*$(*size A* + *size B*) + *8*
  by (*simp add*: *Iff-def*)

lemma *Iff-injective-iff* [*iff*]: (*A IFF B*) = (*A′ IFF B′*) $\longleftrightarrow$ (*A* = *A′* $\wedge$ *B* = *B′*)
  by (*auto simp*: *Iff-def*)

lemma *subst-fm-Iff* [*simp*]: (*A IFF B*)(*i*::=*x*) = (*A*(*i*::=*x*)) *IFF* (*B*(*i*::=*x*))
  by (*auto simp*: *Iff-def*)

lemma *eval-fm-Iff* [*simp*]: *eval-fm e* (*Iff A B*) $\longleftrightarrow$ (*eval-fm e A* $\longleftrightarrow$ *eval-fm e B*)
  by (*auto simp*: *Iff-def*)

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*

lemma *boolean-axioms-hold*: *A* $\in$ *boolean-axioms* $\Longrightarrow$ *eval-fm e A*
  by (*induct rule*: *boolean-axioms.induct*, *auto*)

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

lemma *special-axioms-hold*: *A* $\in$ *special-axioms* $\Longrightarrow$ *eval-fm e A*
  by (*induct rule*: *special-axioms.induct*, *auto*) (*metis eval-subst-fm*)

inductive-set *induction-axioms* :: *fm set* **where**
ind:
atom (*j*::*name*) ♯ (*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*

lemma *twist-forget-eval-fm* [*simp*]:
atom j ♯ (*i*, *A*)
$\Longrightarrow$ *eval-fm* (*finfun-update* (*finfun-update* (*finfun-update e i x*) *j y*) *i z*) *A* =
eval-fm (*finfun-update e i z*) *A*
by (*metis finfun-update-twice finfun-update-twist forget-eval-fm fresh-Pair*)

lemma *induction-axioms-hold*: *A* $\in$ *induction-axioms* $\Longrightarrow$ *eval-fm e A*
by (*induction rule*: *induction-axioms.induct*) (*auto simp*: *eval-subst-fm intro*: *hf-induct-ax*)

1.2.2 Concrete variables

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

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

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

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

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

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

1.2.3 The HF axioms

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

lemma *HF1-holds*: *eval-fm e HF1*
by (*auto simp*: *HF1-def*)

definition *HF2* :: *fm* **where** — the axiom $(z = x \lhd y) = (\forall u.\ (u \in z) = (u \in x \vee u = y))$

HF2 ≡ *Var X0 EQ Eats* (*Var X1*) (*Var X2*) *IFF*
All X3 (*Var X3 IN Var X0 IFF Var X3 IN Var X1 OR Var X3 EQ Var X2*)

lemma *HF2-holds*: *eval-fm e HF2*
by (*auto simp*: *HF2-def*)

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

lemma *HF-axioms-hold*: *A* ∈ *HF-axioms* ⟹ *eval-fm e A*
by (*auto simp*: *HF-axioms-def HF1-holds HF2-holds*)

1.2.4 Equality axioms

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

lemma *refl-ax-holds*: *eval-fm e refl-ax*
by (*auto simp*: *refl-ax-def*)

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

lemma *eq-cong-ax-holds*: *eval-fm e eq-cong-ax*
by (*auto simp*: *Conj-def eq-cong-ax-def*)

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

lemma *mem-cong-ax-holds*: *eval-fm e mem-cong-ax*
by (*auto simp*: *Conj-def mem-cong-ax-def*)

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

lemma *eats-cong-ax-holds*: *eval-fm e eats-cong-ax*
by (*auto simp*: *Conj-def eats-cong-ax-def*)

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

lemma *equality-axioms-hold*: *A* ∈ *equality-axioms* ⟹ *eval-fm e A*
by (*auto simp*: *equality-axioms-def refl-ax-holds eq-cong-ax-holds mem-cong-ax-holds eats-cong-ax-holds*)

1.2.5 The proof system

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

consts *extra-axiom* :: *fm*

specification (*extra-axiom*)
extra-axiom-holds: *eval-fm e extra-axiom*
by (*rule exI* [**where** x = *Zero IN Eats Zero Zero*], *auto*)

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

Soundness theorem!

theorem *hfthm-sound*: **assumes** $H \vdash A$ **shows** ($\forall B{\in}H.$ *eval-fm e B*) $\Longrightarrow$ *eval-fm e A*
using *assms*
proof (*induct arbitrary*: *e*)
case (*Hyp A H*) **thus** *?case*
by *auto*
next
case (*Extra H*) **thus** *?case*
by (*metis extra-axiom-holds*)
next
case (*Bool A H*) **thus** *?case*
by (*metis boolean-axioms-hold*)
next
case (*Eq A H*) **thus** *?case*
by (*metis equality-axioms-hold*)
next
case (*Spec A H*) **thus** *?case*
by (*metis special-axioms-hold*)
next
case (*HF A H*) **thus** *?case*
by (*metis HF-axioms-hold*)
next

case (*Ind A H*) **thus** *?case*
by (*metis induction-axioms-hold*)
next
case (*MP H A B H'*) **thus** *?case*
by *auto*
next
case (*Exists H A B i e*) **thus** *?case*
by *auto* (*metis forget-eval-fm*)
qed

1.2.6 Derived rules of inference

lemma *contraction*: *insert A* (*insert A H*) $\vdash$ *B* $\Longrightarrow$ *insert A H* $\vdash$ *B*
by (*metis insert-absorb2*)

lemma *thin-Un*: *H* $\vdash$ *A* $\Longrightarrow$ *H* $\cup$ *H'* $\vdash$ *A*
by (*metis Bool MP boolean-axioms.Ident sup-commute*)

lemma *thin*: *H* $\vdash$ *A* $\Longrightarrow$ *H* $\subseteq$ *H'* $\Longrightarrow$ *H'* $\vdash$ *A*
by (*metis Un-absorb1 thin-Un*)

lemma *thin0*: {} $\vdash$ *A* $\Longrightarrow$ *H* $\vdash$ *A*
by (*metis sup-bot-left thin-Un*)

lemma *thin1*: *H* $\vdash$ *B* $\Longrightarrow$ *insert A H* $\vdash$ *B*
by (*metis subset-insertI thin*)

lemma *thin2*: *insert A1 H* $\vdash$ *B* $\Longrightarrow$ *insert A1* (*insert A2 H*) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *thin3*: *insert A1* (*insert A2 H*) $\vdash$ *B* $\Longrightarrow$ *insert A1* (*insert A2* (*insert A3 H*)) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *thin4*:
insert A1 (*insert A2* (*insert A3 H*)) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4 H*))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate2*: *insert A2* (*insert A1 H*) $\vdash$ *B* $\Longrightarrow$ *insert A1* (*insert A2 H*) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate3*: *insert A3* (*insert A1* (*insert A2 H*)) $\vdash$ *B* $\Longrightarrow$ *insert A1* (*insert A2* (*insert A3 H*)) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate4*:
insert A4 (*insert A1* (*insert A2* (*insert A3 H*))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4 H*))) $\vdash$ *B*

by (*blast intro*: *thin*)

lemma *rotate5*:
insert A5 (*insert A1* (*insert A2* (*insert A3* (*insert A4 H*)))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5 H*)))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate6*:
insert A6 (*insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5 H*))))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6 H*))))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate7*:
insert A7 (*insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* *H*)))))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert* *A7 H*)))))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate8*:
insert A8 (*insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert A7 H*))))))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert* *A7* (*insert A8 H*))))))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate9*:
insert A9 (*insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert A7* (*insert A8 H*)))))))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert* *A7* (*insert A8* (*insert A9 H*)))))))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *rotate10*:
insert A10 (*insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert A7* (*insert A8* (*insert A9 H*))))))))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert* *A7* (*insert A8* (*insert A9* (*insert A10 H*))))))))) $\vdash$ *B*
by (*blast intro*: *thin*)

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*)))))))))) $\vdash$ *B*
$\Longrightarrow$ *insert A1* (*insert A2* (*insert A3* (*insert A4* (*insert A5* (*insert A6* (*insert* *A7* (*insert A8* (*insert A9* (*insert A10* (*insert A11 H*)))))))))) $\vdash$ *B*
by (*blast intro*: *thin*)

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*))))))))))) $\vdash$ *B*
$\Longrightarrow$ *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*))))))))))) $\vdash$ *B*
by (*blast intro*: *thin*)

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*)))))))))))) $\vdash$ *B*
$\Longrightarrow$ *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*)))))))))))) $\vdash$ *B*
by (*blast intro*: *thin*)

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*))))))))))))) $\vdash$ *B*
$\Longrightarrow$ *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*))))))))))))) $\vdash$ *B*
by (*blast intro*: *thin*)

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*)))))))))))))) $\vdash$ *B*
$\Longrightarrow$ *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*)))))))))))))) $\vdash$ *B*
by (*blast intro*: *thin*)

lemma *MP-same*: $H \vdash A$ *IMP* $B \Longrightarrow H \vdash A \Longrightarrow H \vdash B$
by (*metis MP Un-absorb*)

lemma *MP-thin*: $HA \vdash A$ *IMP* $B \Longrightarrow HB \vdash A \Longrightarrow HA \cup HB \subseteq H \Longrightarrow H \vdash B$
by (*metis MP-same le-sup-iff thin*)

lemma *MP-null*: $\{\} \vdash A$ *IMP* $B \Longrightarrow H \vdash A \Longrightarrow H \vdash B$
by (*metis MP-same thin0*)

lemma *Disj-commute*: $H \vdash B$ *OR* $A \Longrightarrow H \vdash A$ *OR* B
using *DisjConj* [*of B A B*] *Ident* [*of B*]
by (*metis Bool MP-same*)

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

have $H' \cup H \vdash (\mathit{Neg}\ A)\ \mathit{OR}\ (C\ \mathit{OR}\ (\mathit{Neg}\ A))$
by (*metis Bool MP MP-same boolean-axioms.DisjConj Disj-commute DisjAssoc assms*)
thus *?thesis*
by (*metis Bool Disj-commute Un-commute MP-same DisjAssoc DisjCont DisjI1*)
qed

lemma *Assume*: $\mathit{insert}\ A\ H \vdash A$
by (*metis Hyp insertI1*)

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

lemma *Imp-triv-I*: $H \vdash B \Longrightarrow H \vdash A\ \mathit{IMP}\ B$
by (*metis Bool Disj-commute MP-same boolean-axioms.DisjI1*)

lemma *DisjAssoc1*: $H \vdash A\ \mathit{OR}\ (B\ \mathit{OR}\ C) \Longrightarrow H \vdash (A\ \mathit{OR}\ B)\ \mathit{OR}\ C$
by (*metis Bool MP-same boolean-axioms.DisjAssoc*)

lemma *DisjAssoc2*: $H \vdash (A\ \mathit{OR}\ B)\ \mathit{OR}\ C \Longrightarrow H \vdash A\ \mathit{OR}\ (B\ \mathit{OR}\ C)$
by (*metis DisjAssoc1 Disj-commute*)

lemma *Disj-commute-Imp*: $H \vdash (B\ \mathit{OR}\ A)\ \mathit{IMP}\ (A\ \mathit{OR}\ B)$
using *DisjConj* [*of B A B*] *Ident* [*of B*]
by (*metis Bool DisjAssoc2 Disj-commute MP-same*)

lemma *Disj-Semicong-1*: $H \vdash A\ \mathit{OR}\ C \Longrightarrow H \vdash A\ \mathit{IMP}\ B \Longrightarrow H \vdash B\ \mathit{OR}\ C$
using *DisjConj* [*of A C B*]
by (*metis Bool Disj-commute MP-same*)

lemma *Imp-Imp-commute*: $H \vdash B\ \mathit{IMP}\ (A\ \mathit{IMP}\ C) \Longrightarrow H \vdash A\ \mathit{IMP}\ (B\ \mathit{IMP}\ C)$
by (*metis DisjAssoc1 DisjAssoc2 Disj-Semicong-1 Disj-commute-Imp*)

1.2.7 The Deduction Theorem

lemma *deduction-Diff*: **assumes** $H \vdash B$ **shows** $H - \{C\} \vdash C\ \mathit{IMP}\ B$
using *assms*
proof (*induct*)
case (*Hyp A H*) **thus** *?case*
by (*metis Bool Imp-triv-I boolean-axioms.Ident hfthm.Hyp member-remove remove-def*)
next
case (*Extra H*) **thus** *?case*
by (*metis Imp-triv-I hfthm.Extra*)
next

case (*Bool A H*) **thus** *?case*
  by (*metis Imp-triv-I hfthm.Bool*)
next
  case (*Eq A H*) **thus** *?case*
    by (*metis Imp-triv-I hfthm.Eq*)
next
  case (*Spec A H*) **thus** *?case*
    by (*metis Imp-triv-I hfthm.Spec*)
next
  case (*HF A H*) **thus** *?case*
    by (*metis Imp-triv-I hfthm.HF*)
next
  case (*Ind A H*) **thus** *?case*
    by (*metis Imp-triv-I hfthm.Ind*)
next
  case (*MP H A B H'*)
  hence $(H-\{C\}) \cup (H'-\{C\}) \vdash$ *Imp C B*
    by (*simp add*: *S*)
  thus *?case*
    by (*metis Un-Diff*)
next
  case (*Exists H A B i*) **show** *?case*
  proof (*cases* $C \in H$)
    case *True*
    hence *atom i* $\sharp$ *C* **using** *Exists* **by** *auto*
    moreover have $H - \{C\} \vdash$ *A IMP C IMP B* **using** *Exists*
      by (*metis Imp-Imp-commute*)
    ultimately have $H - \{C\} \vdash$ (*Ex i A*) *IMP C IMP B* **using** *Exists*
      by (*metis fm.fresh*(*3*) *fm.fresh*(*4*) *hfthm.Exists member-remove remove-def*)
    thus *?thesis*
      by (*metis Imp-Imp-commute*)
  next
    case *False*
    hence $H - \{C\} = H$ **by** *auto*
    thus *?thesis* **using** *Exists*
      by (*metis Imp-triv-I hfthm.Exists*)
  qed
qed

theorem *Imp-I* [*intro*!]: *insert A H* $\vdash$ *B* $\Longrightarrow$ *H* $\vdash$ *A IMP B*
  by (*metis Diff-insert-absorb Imp-triv-I deduction-Diff insert-absorb*)

lemma *anti-deduction*: *H* $\vdash$ *A IMP B* $\Longrightarrow$ *insert A H* $\vdash$ *B*
   by (*metis Assume MP-same thin1*)

1.2.8 Cut rules

lemma *cut*: $H \vdash A \Longrightarrow$ *insert A H'* $\vdash B \Longrightarrow H \cup H' \vdash B$
  by (*metis MP Un-commute Imp-I*)

lemma *cut-same*: $H \vdash A \Longrightarrow \textit{insert } A\ H \vdash B \Longrightarrow H \vdash B$
by (*metis Un-absorb cut*)

lemma *cut-thin*: $HA \vdash A \Longrightarrow \textit{insert } A\ HB \vdash B \Longrightarrow HA \cup HB \subseteq H \Longrightarrow H \vdash B$
by (*metis thin cut*)

lemma *cut0*: $\{\} \vdash A \Longrightarrow \textit{insert } A\ H \vdash B \Longrightarrow H \vdash B$
by (*metis cut-same thin0*)

lemma *cut1*: $\{A\} \vdash B \Longrightarrow H \vdash A \Longrightarrow H \vdash B$
by (*metis cut sup-bot-right*)

lemma *rcut1*: $\{A\} \vdash B \Longrightarrow \textit{insert } B\ H \vdash C \Longrightarrow \textit{insert } A\ H \vdash C$
by (*metis Assume cut1 cut-same rotate2 thin1*)

lemma *cut2*: $[\![\{A,B\} \vdash C;\ H \vdash A;\ H \vdash B]\!] \Longrightarrow H \vdash C$
by (*metis Un-empty-right Un-insert-right cut cut-same*)

lemma *rcut2*: $\{A,B\} \vdash C \Longrightarrow \textit{insert } C\ H \vdash D \Longrightarrow H \vdash B \Longrightarrow \textit{insert } A\ H \vdash D$
by (*metis Assume cut2 cut-same insert-commute thin1*)

lemma *cut3*: $[\![\{A,B,C\} \vdash D;\ H \vdash A;\ H \vdash B;\ H \vdash C]\!] \Longrightarrow H \vdash D$
by (*metis MP-same cut2 Imp-I*)

lemma *cut4*: $[\![\{A,B,C,D\} \vdash E;\ H \vdash A;\ H \vdash B;\ H \vdash C;\ H \vdash D]\!] \Longrightarrow H \vdash E$
by (*metis MP-same cut3* [*of B C D*] *Imp-I*)

1.3 Miscellaneous logical rules

lemma *Disj-I1*: $H \vdash A \Longrightarrow H \vdash A\ \textit{OR}\ B$
by (*metis Bool MP-same boolean-axioms.DisjI1*)

lemma *Disj-I2*: $H \vdash B \Longrightarrow H \vdash A\ \textit{OR}\ B$
by (*metis Disj-commute Disj-I1*)

lemma *Peirce*: $H \vdash (\textit{Neg } A)\ \textit{IMP}\ A \Longrightarrow H \vdash A$
using *DisjConj* [*of Neg A A A*] *DisjCont* [*of A*]
by (*metis Bool MP-same boolean-axioms.Ident*)

lemma *Contra*: $\textit{insert } (\textit{Neg } A)\ H \vdash A \Longrightarrow H \vdash A$
by (*metis Peirce Imp-I*)

lemma *Imp-Neg-I*: $H \vdash A\ \textit{IMP}\ B \Longrightarrow H \vdash A\ \textit{IMP}\ (\textit{Neg } B) \Longrightarrow H \vdash \textit{Neg } A$
by (*metis DisjConj* [*of B Neg A Neg A*] *DisjCont Bool Disj-commute MP-same*)

lemma *NegNeg-I*: $H \vdash A \Longrightarrow H \vdash \textit{Neg } (\textit{Neg } A)$
using *DisjConj* [*of Neg (Neg A) Neg A Neg (Neg A)*]
by (*metis Bool Ident MP-same*)

lemma *NegNeg-D*: $H \vdash Neg\ (Neg\ A) \Longrightarrow H \vdash A$
by (*metis Disj-I1 Peirce*)

lemma *Neg-D*: $H \vdash Neg\ A \Longrightarrow H \vdash A \Longrightarrow H \vdash B$
by (*metis Imp-Neg-I Imp-triv-I NegNeg-D*)

lemma *Disj-Neg-1*: $H \vdash A\ OR\ B \Longrightarrow H \vdash Neg\ B \Longrightarrow H \vdash A$
by (*metis Disj-I1 Disj-Semicong-1 Disj-commute Peirce*)

lemma *Disj-Neg-2*: $H \vdash A\ OR\ B \Longrightarrow H \vdash Neg\ A \Longrightarrow H \vdash B$
by (*metis Disj-Neg-1 Disj-commute*)

lemma *Neg-Disj-I*: $H \vdash Neg\ A \Longrightarrow H \vdash Neg\ B \Longrightarrow H \vdash Neg\ (A\ OR\ B)$
by (*metis Bool Disj-Neg-1 MP-same boolean-axioms.Ident DisjAssoc*)

lemma *Conj-I* [*intro*!]: $H \vdash A \Longrightarrow H \vdash B \Longrightarrow H \vdash A\ AND\ B$
by (*metis Conj-def NegNeg-I Neg-Disj-I*)

lemma *Conj-E1*: $H \vdash A\ AND\ B \Longrightarrow H \vdash A$
by (*metis Conj-def Bool Disj-Neg-1 NegNeg-D boolean-axioms.DisjI1*)

lemma *Conj-E2*: $H \vdash A\ AND\ B \Longrightarrow H \vdash B$
by (*metis Conj-def Bool Disj-I2 Disj-Neg-2 MP-same DisjAssoc Ident*)

lemma *Conj-commute*: $H \vdash B\ AND\ A \Longrightarrow H \vdash A\ AND\ B$
by (*metis Conj-E1 Conj-E2 Conj-I*)

lemma *Conj-E*: **assumes** *insert A* (*insert B H*) $\vdash C$ **shows** *insert* ($A\ AND\ B$) $H \vdash C$
apply (*rule cut-same* [**where** *A*=*A*], *metis Conj-E1 Hyp insertI1*)
by (*metis* (*full-types*) *AssumeH*(*2*) *Conj-E2 assms cut-same* [**where** *A*=*B*] *insert-commute thin2*)

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** $H \vdash Neg\ A$
by (*rule Imp-Neg-I* [**where** $B = Zero\ IN\ Zero$]) (*auto simp*: *assms*)

lemma *Neg-mono*: $insert\ A\ H \vdash B \Longrightarrow insert\ (Neg\ B)\ H \vdash Neg\ A$
by (*rule Neg-I0*) (*metis Hyp Neg-D insert-commute insertI1 thin1*)

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

by (*metis Conj-E1 Conj-E2 Conj-I Hyp Un-absorb2 cut insertI1 subset-insertI*)

lemma *Disj-mono*:
assumes *insert A H* ⊢ *B insert C H* ⊢ *D* **shows** *insert* (*A OR C*) *H* ⊢ *B OR D*
proof –
{ **fix** *A B C H*
have *insert* (*A OR C*) *H* ⊢ (*A IMP B*) *IMP C OR B*
by (*metis Bool Hyp MP-same boolean-axioms.DisjConj insertI1*)
hence *insert A H* ⊢ *B* ⟹ *insert* (*A OR C*) *H* ⊢ *C OR B*
by (*metis MP-same Un-absorb Un-insert-right Imp-I thin-Un*)
}
thus *?thesis*
by (*metis cut-same assms thin2*)
qed

lemma *Disj-E*:
assumes *A*: *insert A H* ⊢ *C* **and** *B*: *insert B H* ⊢ *C* **shows** *insert* (*A OR B*) *H* ⊢ *C*
by (*metis A B Disj-mono NegNeg-I Peirce*)

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* ⊢ *Neg A* ⟹ *H* ⊢ *Neg A*
by (*metis Contra Neg-mono*)

lemma *NegNeg-E* [*intro*!]: *insert A H* ⊢ *B* ⟹ *insert* (*Neg* (*Neg A*)) *H* ⊢ *B*
by (*metis NegNeg-D Neg-mono*)

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* ⊢ *A* **and** *B*: *insert B H* ⊢ *C* **shows** *insert* (*A IMP B*) *H* ⊢ *C*
proof –
have *insert* (*A IMP B*) *H* ⊢ *B*
by (*metis Hyp A thin1 MP-same insertI1*)
thus *?thesis*
by (*metis cut* [**where** *B*=*C*] *Un-insert-right sup-commute sup-idem B*)
qed

lemma *Imp-cut*:
  assumes *insert* C $H \vdash A$ *IMP* B $\{A\} \vdash C$
    shows $H \vdash A$ *IMP* B
  by (*metis Contra Disj-I1 Neg-mono assms rcut1*)

lemma *Iff-I* [*intro*!]: *insert* A $H \vdash B \Longrightarrow$ *insert* B $H \vdash A \Longrightarrow H \vdash A$ *IFF* B
  by (*metis Iff-def Conj-I Imp-I*)

lemma *Iff-MP-same*: $H \vdash A$ *IFF* $B \Longrightarrow H \vdash A \Longrightarrow H \vdash B$
  by (*metis Iff-def Conj-E1 MP-same*)

lemma *Iff-MP2-same*: $H \vdash A$ *IFF* $B \Longrightarrow H \vdash B \Longrightarrow H \vdash A$
  by (*metis Iff-def Conj-E2 MP-same*)

lemma *Iff-refl* [*intro*!]: $H \vdash A$ *IFF* A
  by (*metis Hyp Iff-I insertI1*)

lemma *Iff-sym*: $H \vdash A$ *IFF* $B \Longrightarrow H \vdash B$ *IFF* A
  by (*metis Iff-def Conj-commute*)

lemma *Iff-trans*: $H \vdash A$ *IFF* $B \Longrightarrow H \vdash B$ *IFF* $C \Longrightarrow H \vdash A$ *IFF* C
  unfolding *Iff-def*
  by (*metis Conj-E1 Conj-E2 Conj-I Disj-Semicong-1 Disj-commute*)

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) $H \vdash C$
  apply (*auto simp*: *Iff-def insert-commute*)
  apply (*metis Disj-I1 Hyp anti-deduction insertCI*)
  apply (*metis Assume Disj-I1 anti-deduction*)
  done

lemma *Iff-E1*:
  assumes A: $H \vdash A$ **and** B: *insert* B $H \vdash C$ **shows** *insert* (A *IFF* B) $H \vdash C$
  by (*metis Iff-def A B Conj-E Imp-E insert-commute thin1*)

lemma *Iff-E2*:
  assumes A: $H \vdash A$ **and** B: *insert* B $H \vdash C$ **shows** *insert* (B *IFF* A) $H \vdash C$
  by (*metis Iff-def A B Bool Conj-E2 Conj-mono Imp-E boolean-axioms.Ident*)

lemma *Iff-MP-left*: $H \vdash A$ *IFF* $B \Longrightarrow$ *insert* A $H \vdash C \Longrightarrow$ *insert* B $H \vdash C$
  by (*metis Hyp Iff-E2 cut-same insertI1 insert-commute thin1*)

lemma *Iff-MP-left'*: $H \vdash A$ *IFF* $B \Longrightarrow$ *insert* B $H \vdash C \Longrightarrow$ *insert* A $H \vdash C$
  by (*metis Iff-MP-left Iff-sym*)

lemma *Swap*: *insert* (*Neg* B) $H \vdash A \Longrightarrow$ *insert* (*Neg* A) $H \vdash B$
  by (*metis NegNeg-D Neg-mono*)

lemma *Cases*: *insert A H* $\vdash$ *B* $\Longrightarrow$ *insert* (*Neg A*) *H* $\vdash$ *B* $\Longrightarrow$ *H* $\vdash$ *B*
by (*metis Contra Neg-D Neg-mono*)

lemma *Neg-Conj-E*: *H* $\vdash$ *B* $\Longrightarrow$ *insert* (*Neg A*) *H* $\vdash$ *C* $\Longrightarrow$ *insert* (*Neg* (*A AND B*)) *H* $\vdash$ *C*
by (*metis Conj-I Swap thin1*)

lemma *Disj-CI*: *insert* (*Neg B*) *H* $\vdash$ *A* $\Longrightarrow$ *H* $\vdash$ *A OR B*
by (*metis Contra Disj-I1 Disj-I2 Swap*)

lemma *Disj-3I*: *insert* (*Neg A*) (*insert* (*Neg C*) *H*) $\vdash$ *B* $\Longrightarrow$ *H* $\vdash$ *A OR B OR C*
by (*metis Disj-CI Disj-commute insert-commute*)

lemma *Contrapos1*: *H* $\vdash$ *A IMP B* $\Longrightarrow$ *H* $\vdash$ *Neg B IMP Neg A*
by (*metis Bool MP-same boolean-axioms.DisjConj boolean-axioms.Ident*)

lemma *Contrapos2*: *H* $\vdash$ (*Neg B*) *IMP* (*Neg A*) $\Longrightarrow$ *H* $\vdash$ *A IMP B*
by (*metis Bool MP-same boolean-axioms.DisjConj boolean-axioms.Ident*)

lemma *ContraAssumeN* [*intro*]: *B* $\in$ *H* $\Longrightarrow$ *insert* (*Neg B*) *H* $\vdash$ *A*
by (*metis Hyp Swap thin1*)

lemma *ContraAssume*: *Neg B* $\in$ *H* $\Longrightarrow$ *insert B H* $\vdash$ *A*
by (*metis Disj-I1 Hyp anti-deduction*)

lemma *ContraProve*: *H* $\vdash$ *B* $\Longrightarrow$ *insert* (*Neg B*) *H* $\vdash$ *A*
by (*metis Swap thin1*)

lemma *Disj-IE1*: *insert B H* $\vdash$ *C* $\Longrightarrow$ *insert* (*A OR B*) *H* $\vdash$ *A OR C*
by (*metis Assume Disj-mono assms*)

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*
by (*metis MP-same Spec special-axioms.intros*)

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*
by (*metis Exists Imp-I anti-deduction assms*)

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 −
have *Ex i A* = *Ex i'* $((i \leftrightarrow i') \cdot A)$ **using** *assms*
apply (*auto simp*: *Abs1-eq-iff fresh-Pair*)
apply (*metis flip-at-simps*(*2*) *fresh-at-base-permute-iff*)+
done
thus *?thesis*
by (*metis Ex-E assms fresh-Pair*)
qed

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*)
by (*auto simp add*: *intro*: *Ex-I* [**where** x=*Var i*])

lemma *All-I* [*intro*!]: $H \vdash A \Longrightarrow \forall C \in H.$ *atom* $i \sharp C \Longrightarrow H \vdash$ *All i A*
by (*auto intro*: *ContraProve Neg-I0*)

lemma *All-D*: $H \vdash$ *All i A* $\Longrightarrow H \vdash A(i::=x)$
by (*metis Assume Ex-I NegNeg-D Neg-mono SyntaxN.Neg cut-same*)

lemma *All-E*: *insert* $(A(i::=x))$ $H \vdash B \Longrightarrow$ *insert* (*All i A*) $H \vdash B$
by (*metis Ex-I NegNeg-D Neg-mono SyntaxN.Neg*)

lemma *All-E'*: $H \vdash$ *All i A* $\Longrightarrow$ *insert* $(A(i::=x))$ $H \vdash B \Longrightarrow H \vdash B$
by (*metis All-D cut-same*)

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*) $H \vdash B$
apply (*rule All-E* [**where** x=x], *auto*)
by (*metis Swap thin1*)

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

1.3.2 Congruence rules

lemma *Neg-cong*: $H \vdash A$ *IFF* $A' \Longrightarrow H \vdash$ *Neg A IFF Neg A'*
by (*metis Iff-def Conj-E1 Conj-E2 Conj-I Contrapos1*)

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

by (*metis Conj-E1 Conj-E2 Disj-mono Iff-I Iff-def anti-deduction*)

lemma *Conj-cong*: $H \vdash A$ *IFF* $A' \Longrightarrow H \vdash B$ *IFF* $B' \Longrightarrow H \vdash A$ *AND* B *IFF* A' *AND* B'
by (*metis Conj-def Disj-cong Neg-cong*)

lemma *Imp-cong*: $H \vdash A$ *IFF* $A' \Longrightarrow H \vdash B$ *IFF* $B' \Longrightarrow H \vdash (A$ *IMP* $B)$ *IFF* $(A'$ *IMP* $B')$
by (*metis Disj-cong Neg-cong*)

lemma *Iff-cong*: $H \vdash A$ *IFF* $A' \Longrightarrow H \vdash B$ *IFF* $B' \Longrightarrow H \vdash (A$ *IFF* $B)$ *IFF* $(A'$ *IFF* $B')$
by (*metis Iff-def Conj-cong Imp-cong*)

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')
apply (*rule Iff-I*)
apply (*metis Ex-mono Hyp Iff-MP-same Un-absorb Un-insert-right insertI1 thin-Un*)
apply (*metis Ex-mono Hyp Iff-MP2-same Un-absorb Un-insert-right insertI1 thin-Un*)
done

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')
by (*metis Ex-cong Neg-cong*)

lemma *Subst*: $H \vdash A \Longrightarrow \forall B \in H.$ *atom* $i \sharp B \Longrightarrow H \vdash A$ $(i::=x)$
by (*metis All-D All-I*)

1.4 Equality reasoning

1.4.1 The congruence property for *op 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 −
obtain *v2::name* **and** *v3::name* **and** *v4::name*
where *v2*: *atom v2* $\sharp$ (*t,X1,X3,X4*)
and *v3*: *atom v3* $\sharp$ (*t,t′,X1,v2,X4*)
and *v4*: *atom v4* $\sharp$ (*t,t′,u,X1,v2,v3*)
by (*metis obtain-fresh*)
have $\{\} \vdash$ (*Var X1 EQ Var X2 AND Var X3 EQ Var X4*) *IMP* (*Var X1 EQ Var X3 IMP Var X2 EQ Var X4*)
by (*rule Eq*) (*simp add: eq-cong-ax-def equality-axioms-def*)
hence $\{\} \vdash$ (*Var X1 EQ Var X2 AND Var X3 EQ Var X4*) *IMP* (*Var X1 EQ Var X3 IMP Var X2 EQ Var X4*)
by (*drule-tac i=X1* **and** *x=Var X1* **in** *Subst*) *simp-all*
hence $\{\} \vdash$ (*Var X1 EQ Var v2 AND Var X3 EQ Var X4*) *IMP* (*Var X1 EQ*

Var X3 IMP Var v2 EQ Var X4)
  by (*drule-tac i=X2* **and** *x=Var v2* **in** *Subst*) *simp-all*
 hence {} ⊢ (*Var X1 EQ Var v2 AND Var v3 EQ Var X4*) *IMP* (*Var X1 EQ Var v3 IMP Var v2 EQ Var X4*)
  using *v2*
  by (*drule-tac i=X3* **and** *x=Var v3* **in** *Subst*) *simp-all*
 hence {} ⊢ (*Var X1 EQ Var v2 AND Var v3 EQ Var v4*) *IMP* (*Var X1 EQ Var v3 IMP Var v2 EQ Var v4*)
  using *v2 v3*
  by (*drule-tac i=X4* **and** *x=Var v4* **in** *Subst*) *simp-all*
 hence {} ⊢ (*t EQ Var v2 AND Var v3 EQ Var v4*) *IMP* (*t EQ Var v3 IMP Var v2 EQ Var v4*)
  using *v2 v3 v4*
  by (*drule-tac i=X1* **and** *x=t* **in** *Subst*) *simp-all*
 hence {} ⊢ (*t EQ t′ AND Var v3 EQ Var v4*) *IMP* (*t EQ Var v3 IMP t′ EQ Var v4*)
  using *v2 v3 v4*
  by (*drule-tac i=v2* **and** *x=t′* **in** *Subst*) *simp-all*
 hence {} ⊢ (*t EQ t′ AND u EQ Var v4*) *IMP* (*t EQ u IMP t′ EQ Var v4*)
  using *v3 v4*
  by (*drule-tac i=v3* **and** *x=u* **in** *Subst*) *simp-all*
 thus *?thesis*
  using *v4*
  by (*drule-tac i=v4* **and** *x=u′* **in** *Subst*) *simp-all*
qed

lemma *Refl* [*iff*]: *H* ⊢ *t EQ t*
proof −
 have {} ⊢ *Var X1 EQ Var X1*
  by (*rule Eq*) (*simp add*: *equality-axioms-def refl-ax-def*)
 hence {} ⊢ *t EQ t*
  by (*drule-tac i=X1* **and** *x=t* **in** *Subst*) *simp-all*
 thus *?thesis*
  by (*metis empty-subsetI thin*)
qed

Apparently necessary in order to prove the congruence property.

lemma *Sym*: **assumes** *H* ⊢ *t EQ u* **shows** *H* ⊢ *u EQ t*
proof −
 have {} ⊢ (*t EQ u AND t EQ t*) *IMP* (*t EQ t IMP u EQ t*)
  by (*rule Eq-cong1*)
 moreover have {*t EQ u*} ⊢ *t EQ u AND t EQ t*
  by (*metis Assume Conj-I Refl*)
 ultimately have {*t EQ u*} ⊢ *u EQ t*
  by (*metis MP-same MP Refl sup-bot-left*)
 thus *H* ⊢ *u EQ t* **by** (*metis assms cut1*)
qed

lemma *Sym-L*: *insert* (*t EQ u*) *H* ⊢ *A* ⟹ *insert* (*u EQ t*) *H* ⊢ *A*

by (*metis Assume Sym Un-empty-left Un-insert-left cut*)

lemma *Trans*: **assumes** $H \vdash x\ EQ\ y$ $H \vdash y\ EQ\ z$ **shows** $H \vdash x\ EQ\ z$
proof −
have $\bigwedge H.\ H \vdash (x\ EQ\ x\ AND\ y\ EQ\ z)\ IMP\ (x\ EQ\ y\ IMP\ x\ EQ\ z)$
by (*metis Eq-cong1 bot-least thin*)
moreover have $\{x\ EQ\ y,\ y\ EQ\ z\} \vdash x\ EQ\ x\ AND\ y\ EQ\ z$
by (*metis Assume Conj-I Refl thin1*)
ultimately have $\{x\ EQ\ y,\ y\ EQ\ z\} \vdash x\ EQ\ z$
by (*metis Hyp MP-same insertI1*)
thus *?thesis*
by (*metis assms cut2*)
qed

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 −
{ fix $t\ t'\ u\ u'$
assume $H \vdash t\ EQ\ t'$ $H \vdash u\ EQ\ u'$
moreover have $\{t\ EQ\ t',\ u\ EQ\ u'\} \vdash t\ EQ\ u\ IMP\ t'\ EQ\ u'$ **using** *Eq-cong1*
by (*metis Assume Conj-I MP-null insert-commute*)
ultimately have $H \vdash t\ EQ\ u\ IMP\ t'\ EQ\ u'$
by (*metis cut2*)
}
thus *?thesis*
by (*metis Iff-def Conj-I assms Sym*)
qed

lemma *Eq-Trans-E*: $H \vdash x\ EQ\ u \Longrightarrow insert\ (t\ EQ\ u)\ H \vdash A \Longrightarrow insert\ (x\ EQ\ t)\ H \vdash A$
by (*metis Assume Sym-L Trans cut-same thin1 thin2*)

1.4.2 The congruence property for *op IN*

lemma *Mem-cong1*: $\{\} \vdash (t\ EQ\ t'\ AND\ u\ EQ\ u')\ IMP\ (t\ IN\ u\ IMP\ t'\ IN\ u')$
proof −
obtain $v2$::*name* **and** $v3$::*name* **and** $v4$::*name*
where $v2$: *atom* $v2 \sharp (t,X1,X3,X4)$
and $v3$: *atom* $v3 \sharp (t,t',X1,v2,X4)$
and $v4$: *atom* $v4 \sharp (t,t',u,X1,v2,v3)$
by (*metis obtain-fresh*)
have $\{\} \vdash (Var\ X1\ EQ\ Var\ X2\ AND\ Var\ X3\ EQ\ Var\ X4)\ IMP\ (Var\ X1\ IN\ Var\ X3\ IMP\ Var\ X2\ IN\ Var\ X4)$
by (*metis mem-cong-ax-def equality-axioms-def insert-iff Eq*)
hence $\{\} \vdash (Var\ X1\ EQ\ Var\ v2\ AND\ Var\ X3\ EQ\ Var\ X4)\ IMP\ (Var\ X1\ IN\ Var\ X3\ IMP\ Var\ v2\ IN\ Var\ X4)$
by (*drule-tac* i=$X2$ **and** x=$Var\ v2$ **in** *Subst*) *simp-all*
hence $\{\} \vdash (Var\ X1\ EQ\ Var\ v2\ AND\ Var\ v3\ EQ\ Var\ X4)\ IMP\ (Var\ X1\ IN\ Var\ v3\ IMP\ Var\ v2\ IN\ Var\ X4)$

using *v2*
by (*drule-tac i=X3* **and** *x=Var v3* **in** *Subst*) *simp-all*
hence {} ⊢ (*Var X1 EQ Var v2 AND Var v3 EQ Var v4*) *IMP* (*Var X1 IN Var v3 IMP Var v2 IN Var v4*)
using *v2 v3*
by (*drule-tac i=X4* **and** *x=Var v4* **in** *Subst*) *simp-all*
hence {} ⊢ (*t EQ Var v2 AND Var v3 EQ Var v4*) *IMP* (*t IN Var v3 IMP Var v2 IN Var v4*)
using *v2 v3 v4*
by (*drule-tac i=X1* **and** *x=t* **in** *Subst*) *simp-all*
hence {} ⊢ (*t EQ t′ AND Var v3 EQ Var v4*) *IMP* (*t IN Var v3 IMP t′ IN Var v4*)
using *v2 v3 v4*
by (*drule-tac i=v2* **and** *x=t′* **in** *Subst*) *simp-all*
hence {} ⊢ (*t EQ t′ AND u EQ Var v4*) *IMP* (*t IN u IMP t′ IN Var v4*)
using *v3 v4*
by (*drule-tac i=v3* **and** *x=u* **in** *Subst*) *simp-all*
thus *?thesis*
using *v4*
by (*drule-tac i=v4* **and** *x=u′* **in** *Subst*) *simp-all*
qed

lemma *Mem-cong*:
assumes *H* ⊢ *t EQ t′ H* ⊢ *u EQ u′* **shows** *H* ⊢ *t IN u IFF t′ IN u′*
proof −
{ fix *t t′ u u′*
have *cong*: {*t EQ t′*, *u EQ u′*} ⊢ *t IN u IMP t′ IN u′*
by (*metis AssumeH*(*2*) *Conj-I MP-null Mem-cong1 insert-commute*)
}
thus *?thesis*
by (*metis Iff-def Conj-I cut2 assms Sym*)
qed

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

lemma *Eats-cong1*: {} ⊢ (*t EQ t′ AND u EQ u′*) *IMP* (*Eats t u EQ Eats t′ u′*)
proof −
obtain *v2*::*name* **and** *v3*::*name* **and** *v4*::*name*
where *v2*: *atom v2* ♯ (*t*,*X1*,*X3*,*X4*)
and *v3*: *atom v3* ♯ (*t*,*t′*,*X1*,*v2*,*X4*)
and *v4*: *atom v4* ♯ (*t*,*t′*,*u*,*X1*,*v2*,*v3*)
by (*metis obtain-fresh*)
have {} ⊢ (*Var X1 EQ Var X2 AND Var X3 EQ Var X4*) *IMP* (*Eats* (*Var X1*) (*Var X3*) *EQ Eats* (*Var X2*) (*Var X4*))
by (*metis eats-cong-ax-def equality-axioms-def insert-iff Eq*)
hence {} ⊢ (*Var X1 EQ Var v2 AND Var X3 EQ Var X4*) *IMP* (*Eats* (*Var X1*) (*Var X3*) *EQ Eats* (*Var v2*) (*Var X4*))
by (*drule-tac i=X2* **and** *x=Var v2* **in** *Subst*) *simp-all*
hence {} ⊢ (*Var X1 EQ Var v2 AND Var v3 EQ Var X4*) *IMP* (*Eats* (*Var X1*)

(*Var v3*) *EQ Eats* (*Var v2*) (*Var X4*))
  using *v2*
  by (*drule-tac i=X3* **and** *x=Var v3* **in** *Subst*) *simp-all*
 hence {} ⊢ (*Var X1 EQ Var v2 AND Var v3 EQ Var v4*) *IMP* (*Eats* (*Var X1*) (*Var v3*) *EQ Eats* (*Var v2*) (*Var v4*))
  using *v2 v3*
  by (*drule-tac i=X4* **and** *x=Var v4* **in** *Subst*) *simp-all*
 hence {} ⊢ (*t EQ Var v2 AND Var v3 EQ Var v4*) *IMP* (*Eats t* (*Var v3*) *EQ Eats* (*Var v2*) (*Var v4*))
  using *v2 v3 v4*
  by (*drule-tac i=X1* **and** *x=t* **in** *Subst*) *simp-all*
 hence {} ⊢ (*t EQ t′ AND Var v3 EQ Var v4*) *IMP* (*Eats t* (*Var v3*) *EQ Eats t′* (*Var v4*))
  using *v2 v3 v4*
  by (*drule-tac i=v2* **and** *x=t′* **in** *Subst*) *simp-all*
 hence {} ⊢ (*t EQ t′ AND u EQ Var v4*) *IMP* (*Eats t u EQ Eats t′* (*Var v4*))
  using *v3 v4*
  by (*drule-tac i=v3* **and** *x=u* **in** *Subst*) *simp-all*
 thus *?thesis*
  using *v4*
  by (*drule-tac i=v4* **and** *x=u′* **in** *Subst*) *simp-all*
qed

lemma *Eats-cong*: ⟦*H* ⊢ *t EQ t′*; *H* ⊢ *u EQ u′*⟧ ⟹ *H* ⊢ *Eats t u EQ Eats t′ u′*
 by (*metis Conj-I anti-deduction Eats-cong1 cut1*)

lemma *HPair-cong*: ⟦*H* ⊢ *t EQ t′*; *H* ⊢ *u EQ u′*⟧ ⟹ *H* ⊢ *HPair t u EQ HPair t′ u′*
 by (*metis HPair-def Eats-cong Refl assms*)

lemma *SUCC-cong*: *H* ⊢ *t EQ t′* ⟹ *H* ⊢ *SUCC t EQ SUCC t′*
 by (*metis Eats-cong SUCC-def assms*)

1.4.4 Substitution for Equalities

lemma *Eq-subst-tm-Iff*: {*t EQ u*} ⊢ *subst i t tm EQ subst i u tm*
 by (*induct tm rule*: *tm.induct*) (*auto simp*: *Eats-cong*)

lemma *Eq-subst-fm-Iff*: *insert* (*t EQ u*) *H* ⊢ *A*(*i*::=*t*) *IFF A*(*i*::=*u*)
proof −
 have { *t EQ u* } ⊢ *A*(*i*::=*t*) *IFF A*(*i*::=*u*)
  by (*nominal-induct A avoiding*: *i t u rule*: *fm.strong-induct*)
   (*auto simp*: *Disj-cong Neg-cong Ex-cong Mem-cong Eq-cong Eq-subst-tm-Iff*)
 thus *?thesis*
  by (*metis Assume cut1*)
qed

lemma *Var-Eq-subst-Iff*: *insert* (*Var i EQ t*) *H* ⊢ *A*(*i*::=*t*) *IFF A*
 by (*metis Eq-subst-fm-Iff Iff-sym subst-fm-id*)

lemma *Var-Eq-imp-subst-Iff*: $H \vdash$ *Var i EQ t* $\Longrightarrow H \vdash A(i{::}{=}t)$ *IFF A*
  by (*metis Var-Eq-subst-Iff cut-same*)

1.4.5 Congruence Rules for Predicates

lemma *P1-cong*:
  fixes *tms* :: *tm list*
  assumes $\bigwedge i\ t\ x.$ *atom i* $\sharp$ *tms* $\Longrightarrow (P\ t)(i{::}{=}x) = P$ (*subst i x t*) **and** $H \vdash x$ *EQ* x'
  shows $H \vdash P\ x$ *IFF* $P\ x'$
proof −
  obtain *i*::*name* **where** *i*: *atom i* $\sharp$ *tms*
    by (*metis obtain-fresh*)
  have *insert* (x *EQ* x') $H \vdash (P\ (Var\ i))(i{::}{=}x)$ *IFF* $(P(Var\ i))(i{::}{=}x')$
    by (*rule Eq-subst-fm-Iff*)
  thus *?thesis* **using** *assms i*
    by (*metis cut-same subst.simps*(*2*))
qed

lemma *P2-cong*:
  fixes *tms* :: *tm list*
  assumes *sub*: $\bigwedge i\ t\ u\ x.$ *atom i* $\sharp$ *tms* $\Longrightarrow (P\ t\ u)(i{::}{=}x) = P$ (*subst i x t*) (*subst i x u*)
      and *eq*: $H \vdash x$ *EQ* x' $H \vdash y$ *EQ* y'
  shows $H \vdash P\ x\ y$ *IFF* $P\ x'\ y'$
proof −
  have *yy'*: { y *EQ* y' } $\vdash P\ x'\ y$ *IFF* $P\ x'\ y'$
    by (*rule P1-cong* [**where** *tms*=[y,x']@*tms*]) (*auto simp*: *fresh-Cons sub*)
  have { x *EQ* x' } $\vdash P\ x\ y$ *IFF* $P\ x'\ y$
    by (*rule P1-cong* [**where** *tms*=[y,x']@*tms*]) (*auto simp*: *fresh-Cons sub*)
  hence {x *EQ* x', y *EQ* y'} $\vdash P\ x\ y$ *IFF* $P\ x'\ y'$
    by (*metis Assume Iff-trans cut1 rotate2 yy'*)
  thus *?thesis*
    by (*metis cut2 eq*)
 qed

lemma *P3-cong*:
  fixes *tms* :: *tm list*
  assumes *sub*: $\bigwedge i\ t\ u\ v\ x.$ *atom i* $\sharp$ *tms* $\Longrightarrow$
                 $(P\ t\ u\ v)(i{::}{=}x) = P$ (*subst i x t*) (*subst i x u*) (*subst i x v*)
      and *eq*: $H \vdash x$ *EQ* x' $H \vdash y$ *EQ* y' $H \vdash z$ *EQ* z'
  shows $H \vdash P\ x\ y\ z$ *IFF* $P\ x'\ y'\ z'$
proof −
  obtain *i*::*name* **where** *i*: *atom i* $\sharp$ (z,z',y,y',x,x')
    by (*metis obtain-fresh*)
  have *tl*: { y *EQ* y', z *EQ* z' } $\vdash P\ x'\ y\ z$ *IFF* $P\ x'\ y'\ z'$
     by (*rule P2-cong* [**where** *tms*=[z,z',y,y',x,x']@*tms*]) (*auto simp*: *fresh-Cons sub*)

have *hd*: { *x EQ x′* } ⊢ *P x y z IFF P x′ y z*
  by (*rule P1-cong* [**where** *tms*=[*z*,*y*,*x′*]@*tms*]) (*auto simp*: *fresh-Cons sub*)
have {*x EQ x′*, *y EQ y′*, *z EQ z′*} ⊢ *P x y z IFF P x′ y′ z′*
  by (*metis Assume thin1 hd* [*THEN cut1*] *tl Iff-trans*)
thus *?thesis*
  by (*rule cut3*) (*rule eq*)+
qed

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 −
  obtain *i*::*name* **where** *i*: *atom i* ♯ (*x4*,*x4′*,*x3*,*x3′*,*x2*,*x2′*,*x1*,*x1′*)
    by (*metis obtain-fresh*)
  have *tl*: { *x2 EQ x2′*, *x3 EQ x3′*, *x4 EQ x4′* } ⊢ *P x1′ x2 x3 x4 IFF P x1′ x2′ x3′ x4′*
    by (*rule P3-cong* [**where** *tms*=[*x4*,*x4′*,*x3*,*x3′*,*x2*,*x2′*,*x1*,*x1′*]@*tms*]) (*auto simp*: *fresh-Cons sub*)
  have *hd*: { *x1 EQ x1′* } ⊢ *P x1 x2 x3 x4 IFF P x1′ x2 x3 x4*
    by (*auto simp*: *fresh-Cons sub intro*!: *P1-cong* [**where** *tms*=[*x4*,*x3*,*x2*,*x1′*]@*tms*])
  have {*x1 EQ x1′*, *x2 EQ x2′*, *x3 EQ x3′*, *x4 EQ x4′*} ⊢ *P x1 x2 x3 x4 IFF P x1′ x2′ x3′ x4′*
    by (*metis Assume thin1 hd* [*THEN cut1*] *tl Iff-trans*)
  thus *?thesis*
    by (*rule cut4*) (*rule eq*)+
qed

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 −
  obtain *i′*::*name* **where** *i′*: *atom i′* ♯ (*t*, *X0*, *X1*, *i*)
    by (*rule obtain-fresh*)
  have {} ⊢ ((*Var X0*) *EQ Zero*) *IFF* (*All X1* (*Neg* ((*Var X1*) *IN* (*Var X0*))))
    by (*simp add*: *HF HF-axioms-def HF1-def*)
  then have {} ⊢ (((*Var X0*) *EQ Zero*) *IFF* (*All X1* (*Neg* ((*Var X1*) *IN* (*Var X0*)))))(*X0* ::= *t*)
    by (*rule Subst*) *simp*
  hence {} ⊢ (*t EQ Zero*) *IFF* (*All i′* (*Neg* ((*Var i′*) *IN t*))) **using** *i′*
    by *simp*
  also have ... = (*FRESH i′*. (*t EQ Zero*) *IFF* (*All i′* (*Neg* ((*Var i′*) *IN t*))))
    using *i′* **by** *simp*
  also have ... = (*t EQ Zero*) *IFF* (*All i* (*Neg* ((*Var i*) *IN t*)))
    using *assms* **by** *simp*

finally show *?thesis*
  by (*metis empty-subsetI thin*)
qed

lemma *Mem-Zero-E* [*intro*!]: *insert* (*x IN Zero*) *H* $\vdash$ *A*
proof −
  obtain *i*::*name* **where** *atom i* $\sharp$ *Zero*
    by (*rule obtain-fresh*)
  hence {} $\vdash$ *All i* (*Neg* ((*Var i*) *IN Zero*))
    by (*metis Mem-Zero-iff Iff-MP-same Refl*)
  hence {} $\vdash$ *Neg* (*x IN Zero*)
    by (*drule-tac x=x* **in** *All-D*) *simp*
  thus *?thesis*
    by (*metis Contrapos2 Hyp Imp-triv-I MP-same empty-subsetI insertI1 thin*)
qed

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* $\equiv$ *Zero IN Zero*

lemma *Fls-eqvt* [*eqvt*]: (*p* $\cdot$ *Fls*) = *Fls*
  by (*simp add*: *Fls-def*)

lemma *Fls-fresh* [*simp*]: *a* $\sharp$ *Fls*
  by (*simp add*: *Fls-def*)

lemma *Neg-I* [*intro*!]: *insert A H* $\vdash$ *Fls* $\Longrightarrow$ *H* $\vdash$ *Neg A*
  unfolding *Fls-def*
  by (*rule Neg-I0*) (*metis Mem-Zero-E cut-same*)

lemma *Neg-E* [*intro*!]: *H* $\vdash$ *A* $\Longrightarrow$ *insert* (*Neg A*) *H* $\vdash$ *Fls*
  by (*rule ContraProve*)

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

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

lemma *Neg-Imp-I* [*intro*!]: $H \vdash A \Longrightarrow$ *insert B H* $\vdash$ *Fls* $\Longrightarrow H \vdash$ *Neg* (*A IMP B*)
  by (*metis NegNeg-I Neg-Disj-I Neg-I*)

lemma *Neg-Imp-E* [*intro*!]: *insert* (*Neg B*) (*insert A H*) $\vdash C \Longrightarrow$ *insert* (*Neg* (*A IMP B*)) *H* $\vdash C$
apply (*rule cut-same* [**where** *A*=*A*])
 apply (*metis Assume Disj-I1 NegNeg-D Neg-mono*)
apply (*metis Swap Imp-I rotate2 thin1*)
done

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* $\vdash A$
  by (*metis Mem-Zero-E Fls-def*)

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*: $H \vdash$ (*Neg Fls*)
  by (*metis Fls-E Neg-I*)

lemma *ExFalso*: $H \vdash$ *Fls* $\Longrightarrow H \vdash A$
  by (*metis Neg-D truth-provable*)

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 −
  obtain *i*::*name* **where** *i*: *atom i* $\sharp$ *t*
    by (*rule obtain-fresh*)
  with *assms* **have** *an*: $H \vdash$ (*All i* (*Neg* ((*Var i*) *IN t*)))
    by (*metis Iff-MP-same Mem-Zero-iff*)
  have $H \vdash$ *Neg* (*u IN t*) **using** *All-D* [*OF an*, *of u*] *i*
    by *simp*
  thus *?thesis* **using** *assms*
    by (*metis Neg-D*)

qed

lemma *Eq-Zero-thm*:
  assumes *atom i* $\sharp$ *t* **shows** {*All i* (*Neg* ((*Var i*) *IN t*))} $\vdash$ *t EQ Zero*
by (*metis Assume Iff-MP2-same Mem-Zero-iff assms*)

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 −
  have *H* $\vdash$ *All i* (*Neg* ((*Var i*) *IN t*))
    by (*metis All-I Neg-I i2 insi*)
  thus *?thesis*
    by (*metis cut-same cut* [*OF Eq-Zero-thm* [*OF i1*] *Hyp*] *insertCI insert-is-Un*)
qed

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 −
  obtain *v1*::*name* **and** *v2*::*name* **and** *i'*::*name*
    where *v1*: *atom v1* $\sharp$ (*z*,*X0*,*X2*,*X3*)
      and *v2*: *atom v2* $\sharp$ (*t*,*z*,*X0*,*v1*,*X3*)
      and *i'*: *atom i'* $\sharp$ (*t*,*u*,*z*,*X0*,*v1*,*v2*,*X3*)
    by (*metis obtain-fresh*)
  have {} $\vdash$ ((*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*))
    by (*simp add*: *HF HF-axioms-def HF2-def*)
  hence {} $\vdash$ ((*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*))
    by (*drule-tac i=X0* **and** *x=Var X0* **in** *Subst*) *simp-all*
  hence {} $\vdash$ ((*Var X0*) *EQ* (*Eats* (*Var v1*) (*Var X2*))) *IFF*
            (*All X3* (*Var X3 IN Var X0 IFF Var X3 IN Var v1 OR Var X3 EQ Var X2*))
    using *v1* **by** (*drule-tac i=X1* **and** *x=Var v1* **in** *Subst*) *simp-all*
  hence {} $\vdash$ ((*Var X0*) *EQ* (*Eats* (*Var v1*) (*Var v2*))) *IFF*
            (*All X3* (*Var X3 IN Var X0 IFF Var X3 IN Var v1 OR Var X3 EQ Var v2*))
    using *v1 v2* **by** (*drule-tac i=X2* **and** *x=Var v2* **in** *Subst*) *simp-all*
  hence {} $\vdash$ (((*Var X0*) *EQ* (*Eats* (*Var v1*) (*Var v2*))) *IFF*
            (*All X3* (*Var X3 IN Var X0 IFF Var X3 IN Var v1 OR Var X3 EQ Var v2*)))(*X0* ::= *z*)
     by (*rule Subst*) *simp*

hence {} ⊢ ((*z EQ* (*Eats* (*Var v1*) (*Var v2*))) *IFF*
(*All i′* (*Var i′ IN z IFF Var i′ IN Var v1 OR Var i′ EQ Var v2*)))
using *v1 v2 i′* **by** (*simp add*: *Conj-def Iff-def*)
hence {} ⊢ (*z EQ* (*Eats t* (*Var v2*))) *IFF*
(*All i′* (*Var i′ IN z IFF Var i′ IN t OR Var i′ EQ Var v2*))
using *v1 v2 i′* **by** (*drule-tac i=v1* **and** *x=t* **in** *Subst*) *simp-all*
hence {} ⊢ (*z EQ Eats t u*) *IFF*
(*All i′* (*Var i′ IN z IFF Var i′ IN t OR Var i′ EQ u*))
using *v1 v2 i′* **by** (*drule-tac i=v2* **and** *x=u* **in** *Subst*) *simp-all*
also have ... = (*FRESH i′*. (*z EQ Eats t u*) *IFF* (*All i′* (*Var i′ IN z IFF Var i′ IN t OR Var i′ EQ u*)))
using *i′* **by** *simp*
also have ... = (*z EQ Eats t u*) *IFF* (*All i* (*Var i IN z IFF Var i IN t OR Var i EQ u*))
using *assms i′* **by** *simp*
finally show *?thesis*
by (*rule thin0*)
qed

lemma *Eq-Eats-I*:
H ⊢ *All i* (*Var i IN z IFF Var i IN t OR Var i EQ u*) ⟹ *atom i* ♯ (*z*,*t*,*u*) ⟹ *H* ⊢ *z EQ Eats t u*
by (*metis Iff-MP2-same Eq-Eats-iff*)

lemma *Mem-Eats-Iff*:
H ⊢ *x IN* (*Eats t u*) *IFF x IN t OR x EQ u*
proof −
obtain *i*::*name* **where** *atom i* ♯ (*Eats t u*, *t*, *u*)
by (*rule obtain-fresh*)
thus *?thesis*
using *Iff-MP-same* [*OF Eq-Eats-iff*, *THEN All-D*]
by *auto*
qed

lemma *Mem-Eats-I1*: *H* ⊢ *u IN t* ⟹ *H* ⊢ *u IN Eats t z*
by (*metis Disj-I1 Iff-MP2-same Mem-Eats-Iff*)

lemma *Mem-Eats-I2*: *H* ⊢ *u EQ z* ⟹ *H* ⊢ *u IN Eats t z*
by (*metis Disj-I2 Iff-MP2-same Mem-Eats-Iff*)

lemma *Mem-Eats-E*:
assumes *A*: *insert* (*u IN t*) *H* ⊢ *C* **and** *B*: *insert* (*u EQ z*) *H* ⊢ *C*
shows *insert* (*u IN Eats t z*) *H* ⊢ *C*
by (*rule Mem-Eats-Iff* [*of - u t z*, *THEN Iff-MP-left′*]) (*metis A B Disj-E*)

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*: $H \vdash u\ IN\ t \Longrightarrow H \vdash u\ IN\ SUCC\ t$
by (*metis Mem-Eats-I1 SUCC-def*)

lemma *Mem-SUCC-I2*: $H \vdash u\ EQ\ t \Longrightarrow H \vdash u\ IN\ SUCC\ t$
by (*metis Mem-Eats-I2 SUCC-def*)

lemma *Mem-SUCC-Refl* [*simp*]: $H \vdash k\ IN\ SUCC\ k$
by (*metis Mem-SUCC-I2 Refl*)

lemma *Mem-SUCC-E*:
assumes *insert* $(u\ IN\ t)\ H \vdash C$ *insert* $(u\ EQ\ t)\ H \vdash C$ **shows** *insert* $(u\ IN\ SUCC\ t)\ H \vdash C$
by (*metis assms Mem-Eats-E SUCC-def*)

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

lemma *Eats-EQ-Zero-E*: *insert* $(Eats\ t\ u\ EQ\ Zero)\ H \vdash A$
by (*metis Assume Eq-Zero-D Mem-Eats-I2 Refl*)

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

lemma *Eats-EQ-Zero-E2*: *insert* $(Zero\ EQ\ Eats\ t\ u)\ H \vdash A$
by (*metis Eats-EQ-Zero-E Sym-L*)

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

1.6 Bounded Quantification involving *Eats*

lemma *All2-cong*: $H \vdash t\ EQ\ t' \Longrightarrow H \vdash A\ IFF\ A' \Longrightarrow \forall\ C \in H.\ atom\ i \sharp\ C \Longrightarrow H \vdash (All2\ i\ t\ A)\ IFF\ (All2\ i\ t'\ A')$
by (*metis All-cong Imp-cong Mem-cong Refl*)

lemma *All2-Zero-E* [*intro*!]: $H \vdash B \Longrightarrow$ *insert* $(All2\ i\ Zero\ A)\ H \vdash B$
by (*rule thin1*)

lemma *All2-Eats-I-D*:
atom i ♯ (*t*,*u*) $\Longrightarrow$ { *All2 i t A*, *A*(*i*::=*u*)} $\vdash$ (*All2 i* (*Eats t u*) *A*)
apply (*auto*, *auto intro*!: *Ex-I* [**where** *x*=*Var i*])
apply (*metis Assume thin1 Var-Eq-subst-Iff* [*THEN Iff-MP-same*])
done

lemma *All2-Eats-I*:
⟦*atom i* ♯ (*t*,*u*); *H* $\vdash$ *All2 i t A*; *H* $\vdash$ *A*(*i*::=*u*)⟧ $\Longrightarrow$ *H* $\vdash$ (*All2 i* (*Eats t u*) *A*)
by (*rule cut2* [*OF All2-Eats-I-D*], *auto*)

lemma *All2-Eats-E1*:
⟦*atom i* ♯ (*t*,*u*); $\forall$ *C* $\in$ *H*. *atom i* ♯ *C*⟧ $\Longrightarrow$ *insert* (*All2 i* (*Eats t u*) *A*) *H* $\vdash$ *All2 i t A*
by *auto* (*metis Assume Ex-I Imp-E Mem-Eats-I1 Neg-mono subst-fm-id*)

lemma *All2-Eats-E2*:
⟦*atom i* ♯ (*t*,*u*); $\forall$ *C* $\in$ *H*. *atom i* ♯ *C*⟧ $\Longrightarrow$ *insert* (*All2 i* (*Eats t u*) *A*) *H* $\vdash$ *A*(*i*::=*u*)
by (*rule All-E* [**where** *x*=*u*]) (*auto intro*: *ContraProve Mem-Eats-I2*)

lemma *All2-Eats-E*:
assumes *i*: *atom i* ♯ (*t*,*u*)
and *B*: *insert* (*All2 i t A*) (*insert* (*A*(*i*::=*u*)) *H*) $\vdash$ *B*
shows *insert* (*All2 i* (*Eats t u*) *A*) *H* $\vdash$ *B*
using *i*
apply (*rule cut-thin* [*OF All2-Eats-E2*, **where** *HB* = *insert* (*All2 i* (*Eats t u*) *A*) *H*], *auto*)
apply (*rule cut-thin* [*OF All2-Eats-E1 B*], *auto*)
done

lemma *All2-SUCC-I*:
atom i ♯ *t* $\Longrightarrow$ *H* $\vdash$ *All2 i t A* $\Longrightarrow$ *H* $\vdash$ *A*(*i*::=*t*) $\Longrightarrow$ *H* $\vdash$ (*All2 i* (*SUCC t*) *A*)
by (*simp add*: *SUCC-def All2-Eats-I*)

lemma *All2-SUCC-E*:
assumes *atom i* ♯ *t*
and *insert* (*All2 i t A*) (*insert* (*A*(*i*::=*t*)) *H*) $\vdash$ *B*
shows *insert* (*All2 i* (*SUCC t*) *A*) *H* $\vdash$ *B*
by (*simp add*: *SUCC-def All2-Eats-E assms*)

lemma *All2-SUCC-E'*:
assumes *H* $\vdash$ *u EQ SUCC t*
and *atom i* ♯ *t* $\forall$ *C* $\in$ *H*. *atom i* ♯ *C*
and *insert* (*All2 i t A*) (*insert* (*A*(*i*::=*t*)) *H*) $\vdash$ *B*
shows *insert* (*All2 i u A*) *H* $\vdash$ *B*
by (*metis All2-SUCC-E Iff-MP-left' Iff-refl All2-cong assms*)

1.7 Induction

lemma *Ind*:
  assumes j: *atom* $(j::name)$ ♯ (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 −
  have $\{A(i::=Zero)$, *All* i (*All* j (A *IMP* ($A(i::=$ *Var* $j)$ *IMP* $A(i::=$ *Eats*(*Var* i)(*Var* j)))))$\} \vdash$ *All* i A
    by (*metis j hfthm.Ind ind anti-deduction insert-commute*)
  hence $H \vdash$ (*All* i A)
    by (*metis cut2 prems*)
  thus *?thesis*
    by (*metis All-E′ Assume subst-fm-id*)
qed

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*
  by (*metis List.finite-set fresh-finite-set-at-base fresh-set*)

lemma *lookup-notin*: *x* ∉ *set e* ⟹ *lookup e n x* = *DBVar x*
  by (*induct e arbitrary*: *n*) *auto*

lemma *lookup-in*:
  $x \in set\ e \Longrightarrow \exists k.\ lookup\ e\ n\ x = DBInd\ k \wedge n \leq k \wedge k < n + length\ e$
apply (*induct e arbitrary*: *n*)
apply (*auto intro*: *Suc-leD*)
apply (*metis Suc-leD add-Suc-right add-Suc-shift*)
done

lemma *lookup-fresh*: $x \sharp lookup\ e\ n\ y \longleftrightarrow y \in set\ e \vee x \neq atom\ y$
  by (*induct arbitrary*: *n rule*: *lookup.induct*) (*auto simp*: *pure-fresh fresh-at-base*)

lemma *lookup-eqvt*[*eqvt*]: $(p \cdot lookup\ xs\ n\ x) = lookup\ (p \cdot xs)\ (p \cdot n)\ (p \cdot x)$
  by (*induct xs arbitrary*: *n*) (*simp-all add*: *permute-pure*)

lemma *lookup-inject* [*iff*]: $(lookup\ e\ n\ x = lookup\ e\ n\ y) \longleftrightarrow x = y$
apply (*induct e n x arbitrary*: *y rule*: *lookup.induct*, *force*, *simp*)
by (*metis Suc-n-not-le-n dbtm.distinct(7) dbtm.eq-iff(3) lookup-in lookup-notin*)

nominal-function *trans-tm* :: *name list* ⇒ *tm* ⇒ *dbtm*
  where
   $trans\text{-}tm\ e\ Zero = DBZero$
 | $trans\text{-}tm\ e\ (Var\ k) = lookup\ e\ 0\ k$
 | $trans\text{-}tm\ e\ (Eats\ t\ u) = DBEats\ (trans\text{-}tm\ e\ t)\ (trans\text{-}tm\ e\ u)$
by (*auto simp*: *eqvt-def trans-tm-graph-aux-def*) (*metis tm.strong-exhaust*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma *fresh-trans-tm-iff* [*simp*]: $i \sharp trans\text{-}tm\ e\ t \longleftrightarrow i \sharp t \vee i \in atom\ `\ set\ e$
  by (*induct t rule*: *tm.induct*, *auto simp*: *lookup-fresh fresh-at-base*)

lemma *trans-tm-forget*: $atom\ i \sharp t \Longrightarrow trans\text{-}tm\ [i]\ t = trans\text{-}tm\ []\ t$
  by (*induct t rule*: *tm.induct*, *auto simp*: *fresh-Pair*)

nominal-function (*invariant* $\lambda(xs, \text{-})\ y.\ atom\ `\ set\ xs \sharp* y$)
  trans-fm :: *name list* ⇒ *fm* ⇒ *dbfm*
  where
   $trans\text{-}fm\ e\ (Mem\ t\ u) = DBMem\ (trans\text{-}tm\ e\ t)\ (trans\text{-}tm\ e\ u)$
 | $trans\text{-}fm\ e\ (Eq\ t\ u)\ = DBEq\ (trans\text{-}tm\ e\ t)\ (trans\text{-}tm\ e\ u)$
 | $trans\text{-}fm\ e\ (Disj\ A\ B) = DBDisj\ (trans\text{-}fm\ e\ A)\ (trans\text{-}fm\ e\ B)$
 | $trans\text{-}fm\ e\ (Neg\ A)\ \ = DBNeg\ (trans\text{-}fm\ e\ A)$
 | $atom\ k \sharp e \Longrightarrow trans\text{-}fm\ e\ (Ex\ k\ A) = DBEx\ (trans\text{-}fm\ (k\#e)\ A)$
apply(*simp add*: *eqvt-def trans-fm-graph-aux-def*)
apply(*erule trans-fm-graph.induct*)
using [[*simproc del*: *alpha-lst*]]
apply(*auto simp*: *fresh-star-def*)
apply(*rule-tac y=b* **and** *c=a* **in** *fm.strong-exhaust*)
apply(*auto simp*: *fresh-star-def*)
apply(*erule-tac c=ea* **in** *Abs-lst1-fcb2'*)

apply (*simp-all add*: *eqvt-at-def*)
apply (*simp-all add*: *fresh-star-Pair perm-supp-eq*)
apply (*simp add*: *fresh-star-def*)
done

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma *fresh-trans-fm* [*simp*]: $i \sharp$ *trans-fm e A* $\longleftrightarrow i \sharp A \vee i \in$ *atom* ‘ *set e*
 by (*nominal-induct A avoiding*: *e rule*: *fm.strong-induct*, *auto simp*: *fresh-at-base*)

abbreviation *DBConj* :: *dbfm* $\Rightarrow$ *dbfm* $\Rightarrow$ *dbfm*
  where *DBConj t u* $\equiv$ *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*)
  by (*simp add*: *Conj-def*)

lemma *trans-tm-inject* [*iff*]: (*trans-tm e t* = *trans-tm e u*) $\longleftrightarrow$ *t* = *u*
proof (*induct t arbitrary*: *e u rule*: *tm.induct*)
  case *Zero* **show** *?case*
    apply (*cases u rule*: *tm.exhaust*, *auto*)
    apply (*metis dbtm.distinct*(*1*) *dbtm.distinct*(*3*) *lookup-in lookup-notin*)
    done
next
  case (*Var i*) **show** *?case*
    apply (*cases u rule*: *tm.exhaust*, *auto*)
    apply (*metis dbtm.distinct*(*1*) *dbtm.distinct*(*3*) *lookup-in lookup-notin*)
    apply (*metis dbtm.distinct*(*10*) *dbtm.distinct*(*11*) *lookup-in lookup-notin*)
    done
next
  case (*Eats tm1 tm2*) **thus** *?case*
    apply (*cases u rule*: *tm.exhaust*, *auto*)
    apply (*metis dbtm.distinct*(*12*) *dbtm.distinct*(*9*) *lookup-in lookup-notin*)
    done
qed

lemma *trans-fm-inject* [*iff*]: (*trans-fm e A* = *trans-fm e B*) $\longleftrightarrow$ *A* = *B*
proof (*nominal-induct A avoiding*: *e B rule*: *fm.strong-induct*)
  case (*Mem tm1 tm2*) **thus** *?case*
    by (*rule fm.strong-exhaust* [**where** *y*=*B* **and** *c*=*e*]) (*auto simp*: *fresh-star-def*)
next
  case (*Eq tm1 tm2*) **thus** *?case*
    by (*rule fm.strong-exhaust* [**where** *y*=*B* **and** *c*=*e*]) (*auto simp*: *fresh-star-def*)
next
  case (*Disj fm1 fm2*) **show** *?case*
   by (*rule fm.strong-exhaust* [**where** *y*=*B* **and** *c*=*e*]) (*auto simp*: *Disj fresh-star-def*)
next
  case (*Neg fm*) **show** *?case*

by (*rule fm.strong-exhaust* [**where** $y{=}B$ **and** $c{=}e$]) (*auto simp*: *Neg fresh-star-def*)
next
case (*Ex name fm*)
thus *?case* **using** [[*simproc del*: *alpha-lst*]]
proof (*cases rule*: *fm.strong-exhaust* [**where** $y{=}B$ **and** $c{=}(e,\ name)$], *simp-all add*: *fresh-star-def*)
fix *name*′::*name* **and** *fm*′::*fm*
assume *name*′: *atom name*′ ♯ (*e*, *name*)
assume *atom name* ♯ *fm*′ ∨ *name* = *name*′
thus (*trans-fm* (*name* # *e*) *fm* = *trans-fm* (*name*′ # *e*) *fm*′) = ([[*atom name*]]*lst*. *fm* = [[*atom name*′]]*lst*. *fm*′)
(**is** *?lhs* = *?rhs*)
proof (*rule disjE*)
assume *name* = *name*′
thus *?lhs* = *?rhs*
by (*metis fresh-Pair fresh-at-base*(*2*) *name*′)
next
assume *name*: *atom name* ♯ *fm*′
have *eq1*: (*name* ↔ *name*′) • *trans-fm* (*name*′ # *e*) *fm*′ = *trans-fm* (*name*′ # *e*) *fm*′
by (*simp add*: *flip-fresh-fresh name*)
have *eq2*: (*name* ↔ *name*′) • ([[*atom name*′]]*lst*. *fm*′) = [[*atom name*′]]*lst*. *fm*′
by (*rule flip-fresh-fresh*) (*auto simp*: *Abs-fresh-iff name*)
show *?lhs* = *?rhs* **using** *name*′ *eq1 eq2 Ex*(*1*) *Ex*(*3*) [*of name*#*e* (*name* ↔ *name*′) • *fm*′]
by (*simp add*: *flip-fresh-fresh*) (*metis Abs1-eq*(*3*))
qed
qed
qed

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 −
have *c-fresh1*: *atom c* ♯ *trans-fm* [*i*] *A*
using *c* **by** (*auto simp*: *supp-Pair*)
moreover
have *i-fresh*: *atom i* ♯ *trans-fm* [*i*] *A*
by *auto*
moreover
have *c-fresh2*: *atom c* ♯ *trans-fm* [*j*] *B*
using *c* **by** (*auto simp*: *supp-Pair*)
moreover
have *j-fresh*: *atom j* ♯ *trans-fm* [*j*] *B*
by *auto*
ultimately have ((*i* ↔ *c*) • (*trans-fm* [*i*] *A*)) = ((*j* ↔ *c*) • *trans-fm* [*j*] *B*)
by (*simp only*: *flip-fresh-fresh t*)

then have *trans-fm* $[c]$ $((i \leftrightarrow c) \cdot A) = $ *trans-fm* $[c]$ $((j \leftrightarrow c) \cdot B)$
  by *simp*
then show $(i \leftrightarrow c) \cdot A = (j \leftrightarrow c) \cdot B$ **by** *simp*
qed

2.2 Characterising the Well-Formed de Bruijn Formulas

2.2.1 Well-Formed Terms

inductive *wf-dbtm* :: *dbtm* $\Rightarrow$ *bool*
  where
    Zero: *wf-dbtm DBZero*
  | *Var*: *wf-dbtm* (*DBVar name*)
  | *Eats*: *wf-dbtm t1* $\Longrightarrow$ *wf-dbtm t2* $\Longrightarrow$ *wf-dbtm* (*DBEats t1 t2*)

equivariance *wf-dbtm*

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

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

lemma *wf-dbtm-imp-is-tm*:
  assumes *wf-dbtm x*
  shows $\exists$ *t*::*tm*. $x =$ *trans-tm* [] *t*
using *assms*
proof (*induct rule*: *wf-dbtm.induct*)
  case *Zero* **thus** *?case*
    by (*metis trans-tm.simps*(*1*))
next
  case (*Var i*) **thus** *?case*
    by (*metis lookup.simps*(*1*) *trans-tm.simps*(*2*))
next
  case (*Eats dt1 dt2*) **thus** *?case*
    by (*metis trans-tm.simps*(*3*))
qed

lemma *wf-dbtm-trans-tm*: *wf-dbtm* (*trans-tm* [] *t*)
  by (*induct t rule*: *tm.induct*) *auto*

theorem *wf-dbtm-iff-is-tm*: *wf-dbtm x* $\longleftrightarrow$ ($\exists$ *t*::*tm*. $x =$ *trans-tm* [] *t*)
  by (*metis wf-dbtm-imp-is-tm wf-dbtm-trans-tm*)

nominal-function *abst-dbtm* :: *name* $\Rightarrow$ *nat* $\Rightarrow$ *dbtm* $\Rightarrow$ *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*)
apply (*simp add*: *eqvt-def abst-dbtm-graph-aux-def*, *auto*)
apply (*metis dbtm.exhaust*)
done

nominal-termination (*eqvt*)
 by *lexicographic-order*

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*)
by (*auto simp*: *eqvt-def subst-dbtm-graph-aux-def*) (*metis dbtm.exhaust*)

nominal-termination (*eqvt*)
 by *lexicographic-order*

lemma *fresh-iff-non-subst-dbtm*: *subst-dbtm DBZero i t* = *t* ⟷ *atom i* ♯ *t*
 by (*induct t rule*: *dbtm.induct*) (*auto simp*: *pure-fresh fresh-at-base*(*2*))

lemma *lookup-append*: *lookup* (*e* @ [*i*]) *n j* = *abst-dbtm i* (*length e* + *n*) (*lookup e n j*)
 by (*induct e arbitrary*: *n*) (*auto simp*: *fresh-Cons*)

lemma *trans-tm-abs*: *trans-tm* (*e*@[*name*]) *t* = *abst-dbtm name* (*length e*) (*trans-tm e t*)
 by (*induct t rule*: *tm.induct*) (*auto simp*: *lookup-notin lookup-append*)

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*)
apply (*simp add*: *eqvt-def abst-dbfm-graph-aux-def*, *auto*)
apply (*metis dbfm.exhaust*)
done

nominal-termination (*eqvt*)
by *lexicographic-order*

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*)
by (*auto simp*: *eqvt-def subst-dbfm-graph-aux-def*) (*metis dbfm.exhaust*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma *fresh-iff-non-subst-dbfm*: *subst-dbfm DBZero i t* = *t* ⟷ *atom i* ♯ *t*
by (*induct t rule*: *dbfm.induct*) (*auto simp*: *fresh-iff-non-subst-dbtm*)

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*
by (*induct t rule*: *dbtm.induct*) (*auto simp*: *pure-fresh*)

lemma *atom-fresh-abst-dbfm* [*simp*]: *atom i* ♯ *abst-dbfm i n A*
by (*nominal-induct A arbitrary*: *n rule*: *dbfm.strong-induct*) *auto*

Nexessary to allow some proofs to go through

nominal-inductive *wf-dbfm*
avoids *Ex*: *name*
by (*auto simp*: *fresh-star-def*)

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)
  apply (nominal-induct A avoiding: name e rule: fm.strong-induct)
  apply (auto simp: trans-tm-abs fresh-Cons fresh-append)
  apply (metis One-nat-def Suc-eq-plus1 append-Cons list.size(4))
  done

lemma abst-trans-fm: abst-dbfm name 0 (trans-fm [] A) = trans-fm [name] A
  by (metis append-Nil list.size(3) trans-fm-abs)

lemma abst-trans-fm2: i ≠ j ⟹ abst-dbfm i (Suc 0) (trans-fm [j] A) = trans-fm
[j,i] A
  using trans-fm-abs [where e=[j] and name=i]
  by auto

lemma wf-dbfm-imp-is-fm:
  assumes wf-dbfm x shows ∃ A::fm. x = trans-fm [] A
using assms
proof (induct rule: wf-dbfm.induct)
  case (Mem t1 t2) thus ?case
    by (metis trans-fm.simps(1) wf-dbtm-imp-is-tm)
next
  case (Eq t1 t2) thus ?case
    by (metis trans-fm.simps(2) wf-dbtm-imp-is-tm)
next
  case (Disj fm1 fm2) thus ?case
    by (metis trans-fm.simps(3))
next
  case (Neg fm) thus ?case
    by (metis trans-fm.simps(4))
next
  case (Ex fm name) thus ?case
    apply auto
    apply (rule-tac x=Ex name A in exI)
    apply (auto simp: abst-trans-fm)
    done
qed

lemma wf-dbfm-trans-fm: wf-dbfm (trans-fm [] A)
  apply (nominal-induct A rule: fm.strong-induct)
  apply (auto simp: wf-dbtm-trans-tm abst-trans-fm)
  apply (metis abst-trans-fm wf-dbfm.Ex)
  done
```

lemma *wf-dbfm-iff-is-fm*: *wf-dbfm x* $\longleftrightarrow$ ($\exists$ *A*::*fm*. *x* = *trans-fm* [] *A*)
by (*metis wf-dbfm-imp-is-fm wf-dbfm-trans-fm*)

lemma *dbtm-abst-ignore* [*simp*]:
abst-dbtm name i (*abst-dbtm name j t*) = *abst-dbtm name j t*
by (*induct t rule*: *dbtm.induct*) *auto*

lemma *abst-dbtm-fresh-ignore* [*simp*]: *atom name* $\sharp$ *u* $\Longrightarrow$ *abst-dbtm name j u* = *u*
by (*induct u rule*: *dbtm.induct*) *auto*

lemma *dbtm-subst-ignore* [*simp*]:
subst-dbtm u name (*abst-dbtm name j t*) = *abst-dbtm name j t*
by (*induct t rule*: *dbtm.induct*) *auto*

lemma *dbtm-abst-swap-subst*:
name $\neq$ *name'* $\Longrightarrow$ *atom name'* $\sharp$ *u* $\Longrightarrow$
subst-dbtm u name (*abst-dbtm name' j t*) = *abst-dbtm name' j* (*subst-dbtm u name t*)
by (*induct t rule*: *dbtm.induct*) *auto*

lemma *dbfm-abst-swap-subst*:
name $\neq$ *name'* $\Longrightarrow$ *atom name'* $\sharp$ *u* $\Longrightarrow$
subst-dbfm u name (*abst-dbfm name' j A*) = *abst-dbfm name' j* (*subst-dbfm u name A*)
by (*induct A arbitrary*: *j rule*: *dbfm.induct*) (*auto simp*: *dbtm-abst-swap-subst*)

lemma *subst-trans-commute* [*simp*]:
atom i $\sharp$ *e* $\Longrightarrow$ *subst-dbtm* (*trans-tm e u*) *i* (*trans-tm e t*) = *trans-tm e* (*subst i u t*)
apply (*induct t rule*: *tm.induct*)
apply (*auto simp*: *lookup-notin fresh-imp-notin-env*)
apply (*metis abst-dbtm-fresh-ignore dbtm-subst-ignore lookup-fresh lookup-notin subst-dbtm.simps*(*2*))
done

lemma *subst-fm-trans-commute* [*simp*]:
subst-dbfm (*trans-tm* [] *u*) *name* (*trans-fm* [] *A*) = *trans-fm* [] (*A* (*name*::= *u*))
apply (*nominal-induct A avoiding*: *name u rule*: *fm.strong-induct*)
apply (*auto simp*: *lookup-notin abst-trans-fm* [*symmetric*])
apply (*metis dbfm-abst-swap-subst fresh-at-base*(*2*) *fresh-trans-tm-iff*)
done

lemma *subst-fm-trans-commute-eq*:
du = *trans-tm* [] *u* $\Longrightarrow$ *subst-dbfm du i* (*trans-fm* [] *A*) = *trans-fm* [] (*A*(*i*::=*u*))
by (*metis subst-fm-trans-commute*)

2.4 Quotations

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

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

lemma *eval-tm-HTuple* [*simp*]: ⟦*HTuple n*⟧*e* = *htuple n*
 by (*induct n*) *auto*

lemma *fresh-HTuple* [*simp*]: *x* ♯ *HTuple n*
 by (*induct n*) *auto*

lemma *HTuple-eqvt*[*eqvt*]: (*p* • *HTuple n*) = *HTuple* (*p* • *n*)
 by (*induct n*, *auto simp*: *HPair-eqvt permute-pure*)

lemma *htuple-nonzero* [*simp*]: *htuple k* ≠ *0*
 by (*induct k*) *auto*

lemma *htuple-inject* [*iff*]: *htuple i* = *htuple j* ⟷ *i*=*j*
proof (*induct i arbitrary*: *j*)
 case *0* **show** *?case*
  by (*cases j*) *auto*
next
 case (*Suc i*) **show** *?case*
  by (*cases j*) (*auto simp*: *Suc*)
qed

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*
 by (*metis nat-of-name-def atom-components-eq-iff atom-eq-iff sort-of-atom-eq*)

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*
 by (*auto simp*: *nat-of-name-def atom-name-def Abs-name-inverse*)

lemma *nat-of-name-name-eq* [*simp*]: *nat-of-name* (*name-of-nat n*) = *n*
 by (*simp add*: *name-of-nat-def*)

lemma *name-of-nat-nat-of-name* [*simp*]: *name-of-nat* (*nat-of-name i*) = *i*
by (*metis nat-of-name-inject nat-of-name-name-eq*)

lemma *HPair-neq-ORD-OF* [*simp*]: *HPair x y* ≠ *ORD-OF i*
by (*metis Not-Ord-hpair Ord-ord-of eval-tm-HPair eval-tm-ORD-OF*)

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*))
by (*rule dbtm.exhaust*) *auto*

termination
by *lexicographic-order*

lemma *quot-dbtm-inject-lemma* [*simp*]: ⟦*quot-dbtm t*⟧*e* = ⟦*quot-dbtm u*⟧*e* ⟷ *t*=*u*
proof (*induct t arbitrary*: *u rule*: *dbtm.induct*)
case *DBZero* **show** *?case*
by (*induct u rule*: *dbtm.induct*) *auto*
next
case (*DBVar name*) **show** *?case*
by (*induct u rule*: *dbtm.induct*) (*auto simp*: *hpair-neq-Ord′*)
next
case (*DBInd k*) **show** *?case*
by (*induct u rule*: *dbtm.induct*) (*auto simp*: *hpair-neq-Ord hpair-neq-Ord′*)
next
case (*DBEats t1 t2*) **thus** *?case*
by (*induct u rule*: *dbtm.induct*) (*simp-all add*: *hpair-neq-Ord*)
qed

lemma *quot-dbtm-inject* [*iff*]: *quot-dbtm t* = *quot-dbtm u* ⟷ *t*=*u*
by (*metis quot-dbtm-inject-lemma*)

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*)
by (*rule-tac y=x* **in** *dbfm.exhaust*, *auto*)

termination
  by *lexicographic-order*

lemma *htuple-minus-1*: $n > 0 \Longrightarrow$ *htuple n* = ⟨*0*, *htuple* (*n* − *1*)⟩
  by (*metis Suc-diff-1 htuple.simps*(*2*))

lemma *HTuple-minus-1*: $n > 0 \Longrightarrow$ *HTuple n* = *HPair Zero* (*HTuple* (*n* − *1*))
  by (*metis Suc-diff-1 HTuple.simps*(*2*))

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

lemma *quot-dbfm-inject-lemma* [*simp*]: ⟦*quot-dbfm A*⟧*e* = ⟦*quot-dbfm B*⟧*e* ⟷ *A*=*B*
proof (*induct A arbitrary*: *B rule*: *dbfm.induct*)
  case (*DBMem t u*) **show** *?case*
    by (*induct B rule*: *dbfm.induct*) (*simp-all add*: *htuple-minus-1*)
next
  case (*DBEq t u*) **show** *?case*
    by (*induct B rule*: *dbfm.induct*) (*auto simp*: *htuple-minus-1*)
next
  case (*DBDisj A B′*) **thus** *?case*
    by (*induct B rule*: *dbfm.induct*) (*simp-all add*: *htuple-minus-1*)
next
  case (*DBNeg A*) **thus** *?case*
    by (*induct B rule*: *dbfm.induct*) (*simp-all add*: *htuple-minus-1*)
next
  case (*DBEx A*) **thus** *?case*
    by (*induct B rule*: *dbfm.induct*) (*simp-all add*: *htuple-minus-1*)
qed

class *quot* =
  fixes *quot* :: *′a* ⇒ *tm* (⌈-⌉)

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

  instance ..
end

lemma *quot-dbtm-fresh* [*simp*]: *s* ♯ (*quot-dbtm t*)
  by (*induct t rule*: *dbtm.induct*) *auto*

lemma *quot-tm-fresh* [*simp*]: **fixes** t::*tm* **shows** $s \sharp \lceil t \rceil$
  by (*simp add*: *quot-tm-def*)

lemma *quot-Zero* [*simp*]: $\lceil Zero \rceil = Zero$
  by (*simp add*: *quot-tm-def*)

lemma *quot-Var*: $\lceil Var\ x \rceil = SUCC\ (ORD\text{-}OF\ (nat\text{-}of\text{-}name\ x))$
  by (*simp add*: *quot-tm-def*)

lemma *quot-Eats*: $\lceil Eats\ x\ y \rceil = HPair\ (HTuple\ 1)\ (HPair\ \lceil x \rceil\ \lceil y \rceil)$
  by (*simp add*: *quot-tm-def*)

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

lemma *eval-quot-dbtm-ignore*:
    $[\![quot\text{-}dbtm\ t]\!]e = [\![quot\text{-}dbtm\ t]\!]e'$
  by (*induct t rule*: *dbtm.induct*) *auto*

lemma *eval-quot-dbfm-ignore*:
    $[\![quot\text{-}dbfm\ A]\!]e = [\![quot\text{-}dbfm\ A]\!]e'$
  by (*induct A rule*: *dbfm.induct*) (*auto intro*: *eval-quot-dbtm-ignore*)

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

  instance ..
end

lemma *quot-dbfm-fresh* [*simp*]: $s \sharp (quot\text{-}dbfm\ A)$
  by (*induct A rule*: *dbfm.induct*) *auto*

lemma *quot-fm-fresh* [*simp*]: **fixes** A::*fm* **shows** $s \sharp \lceil A \rceil$
  by (*simp add*: *quot-fm-def*)

lemma *quot-fm-permute* [*simp*]: **fixes** A:: *fm* **shows** $p \cdot \lceil A \rceil = \lceil A \rceil$
  by (*metis fresh-star-def perm-supp-eq quot-fm-fresh*)

lemma *quot-Mem*: $\lceil x\ IN\ y \rceil = HPair\ (HTuple\ 0)\ (HPair\ (\lceil x \rceil)\ (\lceil y \rceil))$
  by (*simp add*: *quot-fm-def quot-tm-def*)

lemma *quot-Eq*: $\lceil x\ EQ\ y \rceil = HPair\ (HTuple\ 2)\ (HPair\ (\lceil x \rceil)\ (\lceil y \rceil))$
  by (*simp add*: *quot-fm-def quot-tm-def*)

lemma *quot-Disj*: $\lceil A\ OR\ B \rceil = HPair\ (HTuple\ 3)\ (HPair\ (\lceil A \rceil)\ (\lceil B \rceil))$
  by (*simp add*: *quot-fm-def*)

lemma *quot-Neg*: $\lceil Neg\ A\rceil = HPair\ (HTuple\ 4)\ (\lceil A\rceil)$
by (*simp add*: *quot-fm-def*)

lemma *quot-Ex*: $\lceil Ex\ i\ A\rceil = HPair\ (HTuple\ 5)\ (quot\text{-}dbfm\ (trans\text{-}fm\ [i]\ A))$
by (*simp add*: *quot-fm-def*)

lemma *eval-quot-fm-ignore*: **fixes** A:: *fm* **shows** $[\![\lceil A\rceil]\!]e = [\![\lceil A\rceil]\!]e'$
by (*metis eval-quot-dbfm-ignore quot-fm-def*)

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

2.5 Definitions Involving Coding

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

definition *q-Ind* :: *hf* ⇒ *hf*
where *q-Ind k* ≡ $\langle$*htuple 6*, *k*$\rangle$

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

definition *q-Eats* :: *hf* ⇒ *hf* ⇒ *hf*
where *q-Eats x y* ≡ $\langle$*htuple 1*, *x*, *y*$\rangle$

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

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

lemma *quot-Succ*: $\lceil SUCC\ x\rceil = Q\text{-}Succ\ \lceil x\rceil$
by (*auto simp*: *SUCC-def quot-Eats*)

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

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

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

definition *q-Mem* :: *hf* ⇒ *hf* ⇒ *hf*

where $q\text{-}Mem\ x\ y \equiv \langle htuple\ 0,\ x,\ y\rangle$

abbreviation $Q\text{-}Eq :: tm \Rightarrow tm \Rightarrow tm$
where $Q\text{-}Eq\ t\ u \equiv HPair\ (HTuple\ 2)\ (HPair\ t\ u)$

definition $q\text{-}Eq :: hf \Rightarrow hf \Rightarrow hf$
where $q\text{-}Eq\ x\ y \equiv \langle htuple\ 2,\ x,\ y\rangle$

abbreviation $Q\text{-}Disj :: tm \Rightarrow tm \Rightarrow tm$
where $Q\text{-}Disj\ t\ u \equiv HPair\ (HTuple\ 3)\ (HPair\ t\ u)$

definition $q\text{-}Disj :: hf \Rightarrow hf \Rightarrow hf$
where $q\text{-}Disj\ x\ y \equiv \langle htuple\ 3,\ x,\ y\rangle$

abbreviation $Q\text{-}Neg :: tm \Rightarrow tm$
where $Q\text{-}Neg\ t \equiv HPair\ (HTuple\ 4)\ t$

definition $q\text{-}Neg :: hf \Rightarrow hf$
where $q\text{-}Neg\ x \equiv \langle htuple\ 4,\ x\rangle$

abbreviation $Q\text{-}Conj :: tm \Rightarrow tm \Rightarrow tm$
where $Q\text{-}Conj\ t\ u \equiv Q\text{-}Neg\ (Q\text{-}Disj\ (Q\text{-}Neg\ t)\ (Q\text{-}Neg\ u))$

definition $q\text{-}Conj :: hf \Rightarrow hf \Rightarrow hf$
where $q\text{-}Conj\ t\ u \equiv q\text{-}Neg\ (q\text{-}Disj\ (q\text{-}Neg\ t)\ (q\text{-}Neg\ u))$

abbreviation $Q\text{-}Imp :: tm \Rightarrow tm \Rightarrow tm$
where $Q\text{-}Imp\ t\ u \equiv Q\text{-}Disj\ (Q\text{-}Neg\ t)\ u$

definition $q\text{-}Imp :: hf \Rightarrow hf \Rightarrow hf$
where $q\text{-}Imp\ t\ u \equiv q\text{-}Disj\ (q\text{-}Neg\ t)\ u$

abbreviation $Q\text{-}Ex :: tm \Rightarrow tm$
where $Q\text{-}Ex\ t \equiv HPair\ (HTuple\ 5)\ t$

definition $q\text{-}Ex :: hf \Rightarrow hf$
where $q\text{-}Ex\ x \equiv \langle htuple\ 5,\ x\rangle$

abbreviation $Q\text{-}All :: tm \Rightarrow tm$
where $Q\text{-}All\ t \equiv Q\text{-}Neg\ (Q\text{-}Ex\ (Q\text{-}Neg\ t))$

definition $q\text{-}All :: hf \Rightarrow hf$
where $q\text{-}All\ x \equiv q\text{-}Neg\ (q\text{-}Ex\ (q\text{-}Neg\ x))$

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

lemma *q-Eats-iff* [*iff*]: $q\text{-}Eats\ x\ y = q\text{-}Eats\ x'\ y' \longleftrightarrow x{=}x' \land y{=}y'$
by (*metis hpair-iff q-Eats-def*)

lemma *quot-subst-eq*: $\lceil A(i::=t)\rceil$ = *quot-dbfm* (*subst-dbfm* (*trans-tm* [] *t*) *i* (*trans-fm* [] *A*))
  by (*metis quot-fm-def subst-fm-trans-commute*)

lemma *Q-Succ-cong*: $H \vdash x$ *EQ* $x' \Longrightarrow H \vdash$ *Q-Succ* x *EQ* *Q-Succ* x'
  by (*metis HPair-cong Refl*)

2.6 Quotations are Injective

2.6.1 Terms

lemma *eval-tm-inject* [*simp*]: **fixes** *t*::*tm* **shows** $[\![\lceil t\rceil]\!]\ e = [\![\lceil u\rceil]\!]\ e \longleftrightarrow t{=}u$
proof (*induct t arbitrary*: *u rule*: *tm.induct*)
  case *Zero* **thus** *?case*
    by (*cases u rule*: *tm.exhaust*) (*auto simp*: *quot-Var quot-Eats*)
next
  case (*Var i*) **thus** *?case*
    apply (*cases u rule*: *tm.exhaust*, *auto*)
    apply (*auto simp*: *quot-Var quot-Eats*)
    done
next
  case (*Eats t1 t2*) **thus** *?case*
    apply (*cases u rule*: *tm.exhaust*, *auto*)
    apply (*auto simp*: *quot-Eats quot-Var*)
    done
qed

2.6.2 Formulas

lemma *eval-fm-inject* [*simp*]: **fixes** *A*::*fm* **shows** $[\![\lceil A\rceil]\!]\ e = [\![\lceil B\rceil]\!]\ e \longleftrightarrow A{=}B$
proof (*nominal-induct B arbitrary*: *A rule*: *fm.strong-induct*)
  case (*Mem tm1 tm2*) **thus** *?case*
    by (*cases A rule*: *fm.exhaust*, *auto simp*: *quot-simps htuple-minus-1*)
next
  case (*Eq tm1 tm2*) **thus** *?case*
    by (*cases A rule*: *fm.exhaust*, *auto simp*: *quot-simps htuple-minus-1*)
next
  case (*Neg* α) **thus** *?case*
    by (*cases A rule*: *fm.exhaust*, *auto simp*: *quot-simps htuple-minus-1*)
next
  case (*Disj fm1 fm2*)
  thus *?case*
    by (*cases A rule*: *fm.exhaust*, *auto simp*: *quot-simps htuple-minus-1*)
next
  case (*Ex i* α)
  thus *?case*
    apply (*induct A arbitrary*: *i rule*: *fm.induct*)
    apply (*auto simp*: *trans-fm-perm quot-simps htuple-minus-1 Abs1-eq-iff-all*)

by (*metis* (*no-types*) *Abs1-eq-iff-all*(*3*) *dbfm.eq-iff*(*5*) *fm.eq-iff*(*5*) *fresh-Nil* *trans-fm.simps*(*5*))
qed

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

inductive *coding-tm* :: *tm* ⇒ *bool*
  where
    Ord: $\exists\, i.\ x = ORD\text{-}OF\ i \Longrightarrow coding\text{-}tm\ x$
  | *HPair*: $coding\text{-}tm\ x \Longrightarrow coding\text{-}tm\ y \Longrightarrow coding\text{-}tm\ (HPair\ x\ y)$

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

lemma *coding-tm-Zero* [*intro*]: *coding-tm Zero*
  by (*metis ORD-OF.simps*(*1*) *Ord*)

lemma *coding-tm-HTuple* [*intro*]: *coding-tm* (*HTuple k*)
  by (*induct k*, *auto*)

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

lemma *quot-dbtm-coding* [*simp*]: *coding-tm* (*quot-dbtm t*)
  apply (*induct t rule*: *dbtm.induct*, *auto*)
  apply (*metis ORD-OF.simps*(*2*) *Ord*)
  done

lemma *quot-dbfm-coding* [*simp*]: *coding-tm* (*quot-dbfm fm*)
  by (*induct fm rule*: *dbfm.induct*, *auto*)

lemma *quot-fm-coding*: **fixes** *A*::*fm* **shows** *coding-tm* $\lceil A \rceil$
  by (*metis quot-dbfm-coding quot-fm-def*)

inductive *coding-hf* :: *hf* ⇒ *bool*
  where
    Ord: $\exists\, i.\ x = ord\text{-}of\ i \Longrightarrow coding\text{-}hf\ x$
  | *HPair*: $coding\text{-}hf\ x \Longrightarrow coding\text{-}hf\ y \Longrightarrow coding\text{-}hf\ (\langle x,y\rangle)$

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

lemma *coding-hf-0* [*intro*]: *coding-hf 0*
  by (*metis coding-hf.Ord ord-of.simps*(*1*))

inductive-simps *coding-hf-hpair* [*simp*]: *coding-hf* ($\langle x,y\rangle$)

lemma *coding-tm-hf* [*simp*]: $coding\text{-}tm\ t \Longrightarrow coding\text{-}hf\ [\![t]\!]e$
  by (*induct t rule*: *coding-tm.induct*) *auto*

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

Infinite support, so we cannot use nominal primrec.

function *vquot-dbtm* :: *name set* ⇒ *dbtm* ⇒ *tm*
  where
   vquot-dbtm V DBZero = *Zero*
 | *vquot-dbtm V* (*DBVar name*) = (*if name* ∈ *V then Var name*
                               else ORD-OF (*Suc* (*nat-of-name name*)))
 | *vquot-dbtm V* (*DBInd k*) = *HPair* (*HTuple 6*) (*ORD-OF k*)
 | *vquot-dbtm V* (*DBEats t u*) = *HPair* (*HTuple 1*) (*HPair* (*vquot-dbtm V t*) (*vquot-dbtm V u*))
by (*auto*, *rule-tac y=b* **in** *dbtm.exhaust*, *auto*)

termination
  by *lexicographic-order*

lemma *fresh-vquot-dbtm* [*simp*]: *i* ♯ *vquot-dbtm V tm* ⟷ *i* ♯ *tm* ∨ *i* ∉ *atom* ‘ *V*
  by (*induct tm rule*: *dbtm.induct*) (*auto simp*: *fresh-at-base pure-fresh*)

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*)
by (*auto*, *rule-tac y=b* **in** *dbfm.exhaust*, *auto*)

termination
  by *lexicographic-order*

lemma *fresh-vquot-dbfm* [*simp*]: *i* ♯ *vquot-dbfm V fm* ⟷ *i* ♯ *fm* ∨ *i* ∉ *atom* ‘ *V*
  by (*induct fm rule*: *dbfm.induct*) (*auto simp*: *HPair-def HTuple-minus-1*)

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

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

lemma *vquot-dbtm-empty* [*simp*]: *vquot-dbtm* {} *t* = *quot-dbtm t*
  by (*induct t rule*: *dbtm.induct*) *auto*

lemma *vquot-tm-empty* [*simp*]: **fixes** *t*::*tm* **shows** $\lfloor t \rfloor \{\} = \lceil t \rceil$
  by (*simp add*: *vquot-tm-def quot-tm-def*)

lemma *vquot-dbtm-eq*: *atom* ‘ *V* ∩ *supp t* = *atom* ‘ *W* ∩ *supp t* ⟹ *vquot-dbtm V t* = *vquot-dbtm W t*
  by (*induct t rule*: *dbtm.induct*) (*auto simp*: *image-iff*, *blast*+)

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

lemma *vquot-fm-fresh* [*simp*]: **fixes** *A*::*fm* **shows** $i \sharp \lfloor A \rfloor V \longleftrightarrow i \sharp A \vee i \notin atom$ ‘ *V*
  by (*simp add*: *vquot-fm-def*)

lemma *vquot-dbfm-empty* [*simp*]: *vquot-dbfm* {} *A* = *quot-dbfm A*
  by (*induct A rule*: *dbfm.induct*) *auto*

lemma *vquot-fm-empty* [*simp*]: **fixes** *A*::*fm* **shows** $\lfloor A \rfloor \{\} = \lceil A \rceil$
  by (*simp add*: *vquot-fm-def quot-fm-def*)

lemma *vquot-dbfm-eq*: *atom* ‘ *V* ∩ *supp A* = *atom* ‘ *W* ∩ *supp A* ⟹ *vquot-dbfm V A* = *vquot-dbfm W A*
  by (*induct A rule*: *dbfm.induct*) (*auto simp*: *intro*!: *vquot-dbtm-eq*, *blast*+)

lemma *vquot-fm-insert*:
  fixes *A*::*fm* **shows** *atom i* ∉ *supp A* ⟹ $\lfloor A \rfloor (insert\ i\ V) = \lfloor A \rfloor V$
  by (*auto simp*: *vquot-fm-def supp-conv-fresh intro*: *vquot-dbfm-eq*)

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*)
  by (*auto simp*: *eqvt-def Subset-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

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

lemma *Subset-fresh-iff* [*simp*]: *a* ♯ *t SUBS u* ⟷ *a* ♯ *t* ∧ *a* ♯ *u*
apply (*rule obtain-fresh* [**where** *x*=(*t*, *u*)])
apply (*subst Subset.simps*, *auto*)
done

lemma *eval-fm-Subset* [*simp*]: *eval-fm e* (*Subset t u*) ⟷ (⟦*t*⟧*e* ≤ ⟦*u*⟧*e*)
apply (*rule obtain-fresh* [**where** *x*=(*t*, *u*)])
apply (*subst Subset.simps*, *auto*)
done

lemma *subst-fm-Subset* [*simp*]: (*t SUBS u*)(*i*::=*x*) = (*subst i x t*) *SUBS* (*subst i x u*)
proof −
  obtain *j*::*name* **where** *atom j* ♯ (*i*,*x*,*t*,*u*)
    by (*rule obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *Subset.simps* [*of j*])
qed

lemma *Subset-I*:
  assumes *insert* ((*Var i*) *IN t*) $H \vdash$ (*Var i*) *IN u atom i* $\sharp$ (*t*,*u*) $\forall B \in H$. *atom i* $\sharp$ *B*
  shows $H \vdash$ *t SUBS u*
by (*subst Subset.simps* [*of i*]) (*auto simp*: *assms*)

lemma *Subset-D*:
  assumes *major*: $H \vdash$ *t SUBS u* **and** *minor*: $H \vdash$ *a IN t* **shows** $H \vdash$ *a IN u*
proof −
  obtain *i*::*name* **where** *i*: *atom i* $\sharp$ (*t*, *u*)
    by (*rule obtain-fresh*)
  hence $H \vdash$ (*Var i IN t IMP Var i IN u*) (*i*::=*a*)
    by (*metis Subset.simps major All-D*)
  thus *?thesis*
    using *i* **by** *simp* (*metis MP-same minor*)
qed

lemma *Subset-E*: $H \vdash$ *t SUBS u* $\Longrightarrow$ $H \vdash$ *a IN t* $\Longrightarrow$ *insert* (*a IN u*) $H \vdash A \Longrightarrow H \vdash A$
  by (*metis Subset-D cut-same*)

lemma *Subset-cong*: $H \vdash$ *t EQ t'* $\Longrightarrow$ $H \vdash$ *u EQ u'* $\Longrightarrow$ $H \vdash$ *t SUBS u IFF t' SUBS u'*
  by (*rule P2-cong*) *auto*

lemma *Set-MP*: *x SUBS y* $\in H \Longrightarrow$ *z IN x* $\in H \Longrightarrow$ *insert* (*z IN y*) $H \vdash A \Longrightarrow H \vdash A$
  by (*metis Assume Subset-D cut-same insert-absorb*)

lemma *Zero-Subset-I* [*intro*!]: $H \vdash$ *Zero SUBS t*
proof −
  have {} $\vdash$ *Zero SUBS t*
    by (*rule obtain-fresh* [**where** *x*=(*Zero*,*t*)]) (*auto intro*: *Subset-I*)
  thus *?thesis*
    by (*auto intro*: *thin*)
qed

lemma *Zero-SubsetE*: $H \vdash A \Longrightarrow$ *insert* (*Zero SUBS X*) $H \vdash A$
  by (*rule thin1*)

lemma *Subset-Zero-D*:
   assumes $H \vdash$ *t SUBS Zero* **shows** $H \vdash$ *t EQ Zero*
proof −
  obtain *i*::*name* **where** *i* [*iff*]: *atom i* $\sharp$ *t*
    by (*rule obtain-fresh*)
  have {*t SUBS Zero*} $\vdash$ *t EQ Zero*
  proof (*rule Eq-Zero-I*)
    fix *A*
    show {*Var i IN t*, *t SUBS Zero*} $\vdash A$

by (*metis Hyp Subset-D insertI1 thin1 Mem-Zero-E cut1*)
qed *auto*
thus *?thesis*
by (*metis assms cut1*)
qed

lemma *Subset-refl*: $H \vdash t\ SUBS\ t$
proof −
obtain *i::name* **where** *atom i* ♯ *t*
by (*rule obtain-fresh*)
thus *?thesis*
by (*metis Assume Subset-I empty-iff fresh-Pair thin0*)
qed

lemma *Eats-Subset-Iff*: $H \vdash$ *Eats x y SUBS z IFF* (*x SUBS z*) *AND* (*y IN z*)
proof −
obtain *i::name* **where** *i*: *atom i* ♯ (*x,y,z*)
by (*rule obtain-fresh*)
have $\{\} \vdash$ (*Eats x y SUBS z*) *IFF* (*x SUBS z AND y IN z*)
proof (*rule Iff-I*)
show {*Eats x y SUBS z*} $\vdash$ *x SUBS z AND y IN z*
proof (*rule Conj-I*)
show {*Eats x y SUBS z*} $\vdash$ *x SUBS z*
apply (*rule Subset-I* [**where** *i=i*]) **using** *i*
apply (*auto intro*: *Subset-D Mem-Eats-I1*)
done
next
show {*Eats x y SUBS z*} $\vdash$ *y IN z*
by (*metis Subset-D Assume Mem-Eats-I2 Refl*)
qed
next
show {*x SUBS z AND y IN z*} $\vdash$ *Eats x y SUBS z* **using** *i*
by (*auto intro!*: *Subset-I* [**where** *i=i*] *intro*: *Subset-D Mem-cong* [*THEN Iff-MP2-same*])
qed
thus *?thesis*
by (*rule thin0*)
qed

lemma *Eats-Subset-I* [*intro!*]: $H \vdash$ *x SUBS z* $\Longrightarrow H \vdash$ *y IN z* $\Longrightarrow H \vdash$ *Eats x y SUBS z*
by (*metis Conj-I Eats-Subset-Iff Iff-MP2-same*)

lemma *Eats-Subset-E* [*intro!*]:
insert (*x SUBS z*) (*insert* (*y IN z*) *H*) $\vdash C \Longrightarrow$ *insert* (*Eats x y SUBS z*) *H* $\vdash C$
by (*metis Conj-E Eats-Subset-Iff Iff-MP-left'*)

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

lemma *Subset-Eats-I* [*intro*!]: $H \vdash x$ *SUBS Eats x y*
  by (*metis Conj-E1 Eats-Subset-Iff Iff-MP-same Subset-refl*)

lemma *SUCC-Subset-I* [*intro*!]: $H \vdash x$ *SUBS z* $\Longrightarrow H \vdash x$ *IN z* $\Longrightarrow H \vdash$ *SUCC x SUBS z*
  by (*metis Eats-Subset-I SUCC-def*)

lemma *SUCC-Subset-E* [*intro*!]:
  insert (*x SUBS z*) (*insert* (*x IN z*) *H*) $\vdash C \Longrightarrow$ *insert* (*SUCC x SUBS z*) $H \vdash C$
  by (*metis Eats-Subset-E SUCC-def*)

lemma *Subset-trans0*: { *a SUBS b*, *b SUBS c* } $\vdash$ *a SUBS c*
proof −
  obtain *i*::*name* **where** [*simp*]: *atom i* ♯ (*a*,*b*,*c*)
    by (*rule obtain-fresh*)
  show *?thesis*
    by (*rule Subset-I* [*of i*]) (*auto intro*: *Subset-D*)
qed

lemma *Subset-trans*: $H \vdash$ *a SUBS b* $\Longrightarrow H \vdash$ *b SUBS c* $\Longrightarrow H \vdash$ *a SUBS c*
  by (*metis Subset-trans0 cut2*)

lemma *Subset-SUCC*: $H \vdash$ *a SUBS* (*SUCC a*)
  by (*metis SUCC-def Subset-Eats-I*)

lemma *All2-Subset-lemma*: *atom l* ♯ (k',*k*) $\Longrightarrow$ {*P*} $\vdash P' \Longrightarrow$ {*All2 l k P*, k' *SUBS k*} $\vdash$ *All2 l* k' P'
  apply *auto*
  apply (*rule Ex-I* [**where** *x* = *Var l*])
  apply (*auto intro*: *ContraProve Set-MP cut1*)
  done

lemma *All2-Subset*: ⟦$H \vdash$ *All2 l k P*; $H \vdash k'$ *SUBS k*; {*P*} $\vdash P'$; *atom l* ♯ (k', *k*)⟧ $\Longrightarrow H \vdash$ *All2 l* k' P'
  by (*rule cut2* [*OF All2-Subset-lemma*]) *auto*

3.2 Extensionality

lemma *Extensionality*: $H \vdash$ *x EQ y IFF x SUBS y AND y SUBS x*
proof −
  obtain *i*::*name* **and** *j*::*name* **and** *k*::*name*
    where *atoms*: *atom i* ♯ (*x*,*y*) *atom j* ♯ (*i*,*x*,*y*) *atom k* ♯ (*i*,*j*,*y*)
    by (*metis obtain-fresh*)
  have {} $\vdash$ (*Var i EQ y IFF Var i SUBS y AND y SUBS Var i*) (**is** {} $\vdash$ *?scheme*)
  proof (*rule Ind* [*of j*])
    show *atom j* ♯ (*i*, *?scheme*) **using** *atoms*
      by *simp*
  next
    show {} $\vdash$ *?scheme*(*i*::=*Zero*) **using** *atoms*

proof *auto*
show $\{Zero\ EQ\ y\} \vdash y\ SUBS\ Zero$
by (*rule Subset-cong* [*OF Assume Refl*, *THEN Iff-MP-same*]) (*rule Subset-refl*)
next
show $\{Zero\ SUBS\ y,\ y\ SUBS\ Zero\} \vdash Zero\ EQ\ y$
by (*metis AssumeH*(*2*) *Subset-Zero-D Sym*)
qed
next
show $\{\} \vdash All\ i\ (All\ j\ (?scheme\ IMP\ ?scheme(i::=Var\ j)\ IMP\ ?scheme(i::=Eats\ (Var\ i)\ (Var\ j))))$
using *atoms*
apply *auto*
apply (*metis Subset-cong* [*OF Refl Assume*, *THEN Iff-MP-same*] *Subset-Eats-I*)
apply (*metis Mem-cong* [*OF Refl Assume*, *THEN Iff-MP-same*] *Mem-Eats-I2 Refl*)
apply (*metis Subset-cong* [*OF Assume Refl*, *THEN Iff-MP-same*] *Subset-refl*)
apply (*rule Eq-Eats-I* [*of - k*, *THEN Sym*])
apply (*auto intro*: *Set-MP* [**where** $x=y$] *Subset-D* [**where** $t = Var\ i$] *Disj-I1 Disj-I2*)
apply (*rule Var-Eq-subst-Iff* [*THEN Iff-MP-same*], *auto*)
done
qed
hence $\{\} \vdash (Var\ i\ EQ\ y\ IFF\ Var\ i\ SUBS\ y\ AND\ y\ SUBS\ Var\ i)(i::=x)$
by (*metis Subst emptyE*)
thus *?thesis* **using** *atoms*
by (*simp add*: *thin0*)
qed

lemma *Equality-I*: $H \vdash y\ SUBS\ x \Longrightarrow H \vdash x\ SUBS\ y \Longrightarrow H \vdash x\ EQ\ y$
by (*metis Conj-I Extensionality Iff-MP2-same*)

lemma *EQ-imp-SUBS*: *insert* $(t\ EQ\ u)\ H \vdash (t\ SUBS\ u)$
proof –
have $\{t\ EQ\ u\} \vdash (t\ SUBS\ u)$
by (*metis Assume Conj-E Extensionality Iff-MP-left'*)
thus *?thesis*
by (*metis Assume cut1*)
qed

lemma *EQ-imp-SUBS2*: *insert* $(u\ EQ\ t)\ H \vdash (t\ SUBS\ u)$
by (*metis EQ-imp-SUBS Sym-L*)

lemma *Equality-E*: *insert* $(t\ SUBS\ u)$ (*insert* $(u\ SUBS\ t)\ H) \vdash A \Longrightarrow$ *insert* $(t\ EQ\ u)\ H \vdash A$
by (*metis Conj-E Extensionality Iff-MP-left'*)

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*))
 by (*auto simp*: *eqvt-def Disjoint-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

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

lemma *Disjoint-fresh-iff* [*simp*]: *a* ♯ *Disjoint t u* ⟷ *a* ♯ *t* ∧ *a* ♯ *u*
proof −
  obtain *j*::*name* **where** *j*: *atom j* ♯ (*a*,*t*,*u*)
    by (*rule obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *Disjoint.simps* [*of j*])
qed

lemma *subst-fm-Disjoint* [*simp*]:
  (*Disjoint t u*)(*i*::=*x*) = *Disjoint* (*subst i x t*) (*subst i x u*)
proof −
  obtain *j*::*name* **where** *j*: *atom j* ♯ (*i*,*x*,*t*,*u*)
    by (*rule obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *Disjoint.simps* [*of j*])
qed

lemma *Disjoint-cong*: *H* ⊢ *t EQ t′* ⟹ *H* ⊢ *u EQ u′* ⟹ *H* ⊢ *Disjoint t u IFF Disjoint t′ u′*
  by (*rule P2-cong*) *auto*

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*
by (*subst Disjoint.simps* [*of i*]) (*auto simp*: *assms insert-commute*)

lemma *Disjoint-E*:
  assumes *major*: *H* ⊢ *Disjoint t u* **and** *minor*: *H* ⊢ *a IN t H* ⊢ *a IN u* **shows** *H* ⊢ *A*
proof −
  obtain *i*::*name* **where** *i*: *atom i* ♯ (*t*, *u*)
    by (*rule obtain-fresh*)
  hence *H* ⊢ (*Var i IN t IMP Neg* (*Var i IN u*)) (*i*::=*a*)
    by (*metis Disjoint.simps major All-D*)
  thus *?thesis* **using** *i*
    by *simp* (*metis MP-same Neg-D minor*)
qed

lemma *Disjoint-commute*: { *Disjoint t u* } ⊢ *Disjoint u t*

proof −
  obtain *i*::*name* **where** *atom i* ♯ (*t*,*u*)
    by (*rule obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *fresh-Pair intro*: *Disjoint-I Disjoint-E*)
qed

lemma *Disjoint-commute-I*: $H \vdash$ *Disjoint t u* $\Longrightarrow$ $H \vdash$ *Disjoint u t*
  by (*metis Disjoint-commute cut1*)

lemma *Disjoint-commute-D*: *insert* (*Disjoint t u*) $H \vdash A \Longrightarrow$ *insert* (*Disjoint u t*) $H \vdash A$
  by (*metis Assume Disjoint-commute-I cut-same insert-commute thin1*)

lemma *Zero-Disjoint-I1* [*iff*]: $H \vdash$ *Disjoint Zero t*
proof −
  obtain *i*::*name* **where** *i*: *atom i* ♯ *t*
    by (*rule obtain-fresh*)
  hence {} ⊢ *Disjoint Zero t*
    by (*auto intro*: *Disjoint-I* [*of i*])
  thus *?thesis*
    by (*metis thin0*)
qed

lemma *Zero-Disjoint-I2* [*iff*]: $H \vdash$ *Disjoint t Zero*
  by (*metis Disjoint-commute Zero-Disjoint-I1 cut1*)

lemma *Disjoint-Eats-D1*: { *Disjoint* (*Eats x y*) *z* } ⊢ *Disjoint x z*
proof −
  obtain *i*::*name* **where** *i*: *atom i* ♯ (*x*,*y*,*z*)
    by (*rule obtain-fresh*)
  show *?thesis*
    apply (*rule Disjoint-I* [*of i*])
    apply (*blast intro*: *Disjoint-E Mem-Eats-I1*)
    using *i* **apply** *auto*
    done
qed

lemma *Disjoint-Eats-D2*: { *Disjoint* (*Eats x y*) *z* } ⊢ *Neg*(*y IN z*)
proof −
  obtain *i*::*name* **where** *i*: *atom i* ♯ (*x*,*y*,*z*)
    by (*rule obtain-fresh*)
  show *?thesis*
    by (*force intro*: *Disjoint-E* [*THEN rotate2*] *Mem-Eats-I2*)
qed

lemma *Disjoint-Eats-E*:
  insert (*Disjoint x z*) (*insert* (*Neg*(*y IN z*)) *H*) ⊢ *A* $\Longrightarrow$ *insert* (*Disjoint* (*Eats x y*) *z*) *H* ⊢ *A*

apply (*rule cut-same* [*OF cut1* [*OF Disjoint-Eats-D2*, *OF Assume*]])
apply (*rule cut-same* [*OF cut1* [*OF Disjoint-Eats-D1*, *OF Hyp*]])
apply (*auto intro*: *thin*)
done

lemma *Disjoint-Eats-E2*:
insert (*Disjoint z x*) (*insert* (*Neg*(*y IN z*)) *H*) ⊢ *A* ⟹ *insert* (*Disjoint z* (*Eats x y*)) *H* ⊢ *A*
by (*metis Disjoint-Eats-E Disjoint-commute-D*)

lemma *Disjoint-Eats-Imp*: { *Disjoint x z*, *Neg*(*y IN z*) } ⊢ *Disjoint* (*Eats x y*) *z*
proof −
obtain *i*::*name* **where***atom i* ♯ (*x*,*y*,*z*)
by (*rule obtain-fresh*)
then show *?thesis*
by (*auto intro*: *Disjoint-I* [*of i*] *Disjoint-E* [*THEN rotate3*]
Mem-cong [*OF Assume Refl*, *THEN Iff-MP-same*])
qed

lemma *Disjoint-Eats-I* [*intro*!]: *H* ⊢ *Disjoint x z* ⟹ *insert* (*y IN z*) *H* ⊢ *Fls* ⟹ *H* ⊢ *Disjoint* (*Eats x y*) *z*
by (*metis Neg-I cut2* [*OF Disjoint-Eats-Imp*])

lemma *Disjoint-Eats-I2* [*intro*!]: *H* ⊢ *Disjoint z x* ⟹ *insert* (*y IN z*) *H* ⊢ *Fls* ⟹ *H* ⊢ *Disjoint z* (*Eats x y*)
by (*metis Disjoint-Eats-I Disjoint-commute cut1*)

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 −
obtain *j*::*name* **where** *j*: *atom j* ♯ (*z*,*i*)
by (*metis obtain-fresh*)
show *?thesis*
apply (*rule Ind* [*of j*]) **using** *i j*
apply *auto*
apply (*rule Ex-I* [**where** *x*=*Zero*], *auto*)
apply (*rule Ex-I* [**where** *x*=*Eats* (*Var i*) (*Var j*)], *auto*)
apply (*metis ContraAssume insertI1 insert-commute*)
apply (*metis ContraProve Disjoint-Eats-Imp rotate2 thin1*)
apply (*metis Assume Disj-I1 anti-deduction rotate3*)
done
qed

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

```
  apply auto
  apply (rule Eq-Zero-I)
  apply (rule cut-same [where A = (Neg ((Var i) IN z) AND Disjoint (Var i)
z)])
  apply (rule Foundation-lemma [THEN cut1], auto)
  done

lemma Mem-Neg-refl: {} ⊢ Neg (x IN x)
proof −
  obtain i::name where i: atom i ♯ x
    by (metis obtain-fresh)
  have {} ⊢ Disjoint x (Eats Zero x)
    apply (rule cut-same [OF Foundation [where z = Eats Zero x]]) using i
    apply auto
    apply (rule cut-same [where A = Disjoint x (Eats Zero x)])
    apply (metis Assume thin1 Disjoint-cong [OF Assume Refl, THEN Iff-MP-same])
    apply (metis Assume AssumeH(4) Disjoint-E Mem-Eats-I2 Refl)
    done
  thus ?thesis
    by (metis Disjoint-Eats-D2 Disjoint-commute cut-same)
qed

lemma Mem-refl-E [intro!]: insert (x IN x) H ⊢ A
  by (metis Disj-I1 Mem-Neg-refl anti-deduction thin0)

lemma Mem-non-refl: assumes H ⊢ x IN x shows H ⊢ A
  by (metis Mem-refl-E assms cut-same)

lemma Mem-Neg-sym: { x IN y, y IN x } ⊢ Fls
proof −
  obtain i::name where i: atom i ♯ (x,y)
    by (metis obtain-fresh)
  have {} ⊢ Disjoint x (Eats Zero y) OR Disjoint y (Eats Zero x)
    apply (rule cut-same [OF Foundation [where i=i and z = Eats (Eats Zero
y) x]]) using i
    apply (auto intro!: Disjoint-Eats-E2 [THEN rotate2])
    apply (rule Disj-I2, auto)
    apply (metis Assume EQ-imp-SUBS2 Subset-D insert-commute)
    apply (blast intro!: Disj-I1 Disjoint-cong [OF Hyp Refl, THEN Iff-MP-same])
    done
  thus ?thesis
    by (auto intro: cut0 Disjoint-Eats-E2)
qed

lemma Mem-not-sym: insert (x IN y) (insert (y IN x) H) ⊢ A
  by (rule cut-thin [OF Mem-Neg-sym]) auto
```

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*)))
  by (*auto simp*: *eqvt-def OrdP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma
 shows *OrdP-fresh-iff* [*simp*]: *a* ♯ *OrdP x* ⟷ *a* ♯ *x*   (**is** *?thesis1*)
  and *eval-fm-OrdP* [*simp*]: *eval-fm e* (*OrdP x*) ⟷ *Ord* $[\![x]\!]$*e* (**is** *?thesis2*)
proof −
  obtain *z*::*name* **and** *y*::*name* **where** *atom z* ♯ *x atom y* ♯ (*x*, *z*)
    by (*metis obtain-fresh*)
  thus *?thesis1 ?thesis2*
    by (*auto simp*: *OrdP.simps* [*of y - z*] *Ord-def Transset-def*)
qed

lemma *subst-fm-OrdP* [*simp*]: (*OrdP t*)(*i*::=*x*) = *OrdP* (*subst i x t*)
proof −
  obtain *z*::*name* **and** *y*::*name* **where** *atom z* ♯ (*t*,*i*,*x*) *atom y* ♯ (*t*,*i*,*x*,*z*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *OrdP.simps* [*of y - z*])
qed

lemma *OrdP-cong*: *H* ⊢ *x EQ x′* ⟹ *H* ⊢ *OrdP x IFF OrdP x′*
  by (*rule P1-cong*) *auto*

lemma *OrdP-Mem-lemma*:
  assumes *z*: *atom z* ♯ (*k*,*l*) **and** *l*: *insert* (*OrdP k*) *H* ⊢ *l IN k*
  shows *insert* (*OrdP k*) *H* ⊢ *l SUBS k AND All2 z l* (*Var z SUBS l*)
proof −
  obtain *y*::*name* **where** *y*: *atom y* ♯ (*k*,*l*,*z*)
    by (*metis obtain-fresh*)
  have *insert* (*OrdP k*) *H*
      ⊢ (*Var y IN k IMP* (*Var y SUBS k AND All2 z* (*Var y*) (*Var z SUBS Var y*)))(*y*::=*l*)
    by (*rule All-D*) (*simp add*: *OrdP.simps* [*of y - z*] *y z Assume*)
  also have ... = *l IN k IMP* (*l SUBS k AND All2 z l* (*Var z SUBS l*))
    using *y z* **by** *simp*
  finally show *?thesis*
    by (*metis MP-same l*)
qed

lemma *OrdP-Mem-E*:

assumes *atom z ♯ (k,l)*
insert (OrdP k) H ⊢ l IN k
insert (l SUBS k) (insert (All2 z l (Var z SUBS l)) H) ⊢ A
shows *insert (OrdP k) H ⊢ A*
apply (*rule OrdP-Mem-lemma* [*THEN cut-same*])
apply (*auto simp*: *insert-commute*)
apply (*blast intro*: *assms thin1*)+
done

lemma *OrdP-Mem-imp-Subset*:
assumes *k*: *H ⊢ k IN l* **and** *l*: *H ⊢ OrdP l* **shows** *H ⊢ k SUBS l*
apply (*rule obtain-fresh* [*of* (*l*,*k*)])
apply (*rule cut-same* [*OF l*])
using *k* **apply** (*auto intro*: *OrdP-Mem-E thin1*)
done

lemma *SUCC-Subset-Ord-lemma*: { *k′ IN k*, *OrdP k* } ⊢ *SUCC k′ SUBS k*
by *auto* (*metis Assume thin1 OrdP-Mem-imp-Subset*)

lemma *SUCC-Subset-Ord*: *H ⊢ k′ IN k ⟹ H ⊢ OrdP k ⟹ H ⊢ SUCC k′ SUBS k*
by (*blast intro*!: *cut2* [*OF SUCC-Subset-Ord-lemma*])

lemma *OrdP-Trans-lemma*: { *OrdP k*, *i IN j*, *j IN k* } ⊢ *i IN k*
proof −
obtain *m*::*name* **where** *atom m ♯ (i,j,k)*
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto intro*: *OrdP-Mem-E* [*of m k j*] *Subset-D* [*THEN rotate3*])
qed

lemma *OrdP-Trans*: *H ⊢ OrdP k ⟹ H ⊢ i IN j ⟹ H ⊢ j IN k ⟹ H ⊢ i IN k*
by (*blast intro*: *cut3* [*OF OrdP-Trans-lemma*])

lemma *Ord-IN-Ord0*:
assumes *l*: *H ⊢ l IN k*
shows *insert (OrdP k) H ⊢ OrdP l*
proof −
obtain *z*::*name* **and** *y*::*name* **where** *z*: *atom z ♯ (k,l)* **and** *y*: *atom y ♯ (k,l,z)*
by (*metis obtain-fresh*)
have {*Var y IN l*, *OrdP k*, *l IN k*} ⊢ *All2 z (Var y) (Var z SUBS Var y)* **using** *y z*
apply (*simp add*: *insert-commute* [*of - OrdP k*])
apply (*auto intro*: *OrdP-Mem-E* [*of z k Var y*] *OrdP-Trans-lemma del*: *All-I Neg-I*)
done
hence {*OrdP k*, *l IN k*} ⊢ *OrdP l* **using** *z y*
apply (*auto simp*: *OrdP.simps* [*of y l z*])

apply (*simp add*: *insert-commute* [*of* - *OrdP k*])
apply (*rule OrdP-Mem-E* [*of y k l*], *simp-all*)
apply (*metis Assume thin1*)
apply (*rule All-E* [**where** *x*= *Var y*, *THEN thin1*], *simp*)
apply (*metis Assume anti-deduction insert-commute*)
done
thus *?thesis*
by (*metis* (*full-types*) *Assume l cut2 thin1*)
qed

lemma *Ord-IN-Ord*: $H \vdash l\ IN\ k \Longrightarrow H \vdash OrdP\ k \Longrightarrow H \vdash OrdP\ l$
by (*metis Ord-IN-Ord0 cut-same*)

lemma *OrdP-I*:
assumes *insert* (*Var y IN x*) $H \vdash$ (*Var y*) *SUBS x*
and *insert* (*Var z IN Var y*) (*insert* (*Var y IN x*) *H*) $\vdash$ (*Var z*) *SUBS* (*Var y*)
and *atom y* ♯ (*x*, *z*) $\forall B \in H$. *atom y* ♯ *B* *atom z* ♯ *x* $\forall B \in H$. *atom z* ♯ *B*
shows $H \vdash$ *OrdP x*
using *assms* **by** *auto*

lemma *OrdP-Zero* [*simp*]: $H \vdash$ *OrdP Zero*
proof −
obtain *y*::*name* **and** *z*::*name* **where** *atom y* ♯ *z*
by (*rule obtain-fresh*)
hence {} $\vdash$ *OrdP Zero*
by (*auto intro*: *OrdP-I* [*of y* - - *z*])
thus *?thesis*
by (*metis thin0*)
qed

lemma *OrdP-SUCC-I0*: { *OrdP k* } $\vdash$ *OrdP* (*SUCC k*)
proof −
obtain *w*::*name* **and** *y*::*name* **and** *z*::*name* **where** *atoms*: *atom w* ♯ (*k*,*y*,*z*) *atom y* ♯ (*k*,*z*) *atom z* ♯ *k*
by (*metis obtain-fresh*)
have *1*: {*Var y IN SUCC k*, *OrdP k*} $\vdash$ *Var y SUBS SUCC k*
apply (*rule Mem-SUCC-E*)
apply (*rule OrdP-Mem-E* [*of w* - *Var y*, *THEN rotate2*]) **using** *atoms*
apply *auto*
apply (*metis Assume Subset-SUCC Subset-trans*)
apply (*metis EQ-imp-SUBS Subset-SUCC Subset-trans*)
done
have *in-case*: {*Var y IN k*, *Var z IN Var y*, *OrdP k*} $\vdash$ *Var z SUBS Var y*
apply (*rule OrdP-Mem-E* [*of w* - *Var y*, *THEN rotate3*]) **using** *atoms*
apply (*auto intro*: *All2-E* [*THEN thin1*])
done
have {*Var y EQ k*, *Var z IN k*, *OrdP k*} $\vdash$ *Var z SUBS Var y*
by (*metis AssumeH*(*2*) *AssumeH*(*3*) *EQ-imp-SUBS2 OrdP-Mem-imp-Subset*

Subset-trans)
  hence *eq-case*: {*Var y EQ k*, *Var z IN Var y*, *OrdP k*} ⊢ *Var z SUBS Var y*
    by (*rule cut3*) (*auto intro*: *EQ-imp-SUBS* [*THEN cut1*] *Subset-D*)
  have *2*: {*Var z IN Var y*, *Var y IN SUCC k*, *OrdP k*} ⊢ *Var z SUBS Var y*
    by (*metis rotate2 Mem-SUCC-E in-case eq-case*)
  show *?thesis*
    apply (*rule OrdP-I* [*OF 1 2*])
    using *atoms* **apply** *auto*
    done
qed

lemma *OrdP-SUCC-I*: *H* ⊢ *OrdP k* ⟹ *H* ⊢ *OrdP* (*SUCC k*)
  by (*metis OrdP-SUCC-I0 cut1*)

lemma *Zero-In-OrdP*: { *OrdP x* } ⊢ *x EQ Zero OR Zero IN x*
proof −
  obtain *i*::*name* **and** *j*::*name*
    where *i*: *atom i* ♯ *x* **and** *j*: *atom j* ♯ (*x*,*i*)
    by (*metis obtain-fresh*)
  show *?thesis*
   apply (*rule cut-thin* [**where** *HB* = {*OrdP x*}, *OF Foundation* [**where** *i*=*i* **and**
z = *x*]])
    using *i j* **apply** *auto*
    prefer *2* **apply** (*metis Assume Disj-I1*)
    apply (*rule Disj-I2*)
    apply (*rule cut-same* [**where** *A* = *Var i EQ Zero*])
    prefer *2* **apply** (*blast intro*: *Iff-MP-same* [*OF Mem-cong* [*OF Assume Refl*]])
    apply (*auto intro*!: *Eq-Zero-I* [**where** *i*=*j*] *Ex-I* [**where** *x*=*Var i*])
    apply (*blast intro*: *Disjoint-E Subset-D*)
    done
qed

lemma *OrdP-HPairE*: *insert* (*OrdP* (*HPair x y*)) *H* ⊢ *A*
proof −
  have { *OrdP* (*HPair x y*) } ⊢ *A*
    by (*rule cut-same* [*OF Zero-In-OrdP*]) (*auto simp*: *HPair-def*)
  thus *?thesis*
    by (*metis Assume cut1*)
qed

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

lemma *Zero-Eq-HPairE*: *insert* (*Zero EQ HPair x y*) *H* ⊢ *A*
  by (*metis Eats-EQ-Zero-E2 HPair-def*)

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

lemma *HPair-Eq-ZeroE*: *insert* (*HPair x y EQ Zero*) *H* ⊢ *A*
by (*metis Sym-L Zero-Eq-HPairE*)

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

3.6 Induction on Ordinals

lemma *OrdInd-lemma*:
assumes *j*: *atom* (*j*::*name*) ♯ (*i*,*A*)
shows { *OrdP* (*Var i*) } ⊢ (*All i* (*OrdP* (*Var i*) *IMP* ((*All2 j* (*Var i*) (*A*(*i*::= *Var j*))) *IMP A*))) *IMP A*
proof −
obtain *l*::*name* **and** *k*::*name*
where *l*: *atom l* ♯ (*i*,*j*,*A*) **and** *k*: *atom k* ♯ (*i*,*j*,*l*,*A*)
by (*metis obtain-fresh*)
have { (*All i* (*OrdP* (*Var i*) *IMP* ((*All2 j* (*Var i*) (*A*(*i*::= *Var j*))) *IMP A*))) }
⊢ (*All2 l* (*Var i*) (*OrdP* (*Var l*) *IMP A*(*i*::= *Var l*)))
apply (*rule Ind* [*of k*])
using *j k l* **apply** *auto*
apply (*rule All-E* [**where** *x*=*Var l*, *THEN rotate5*], *auto*)
apply (*metis Assume Disj-I1 anti-deduction thin1*)
apply (*rule Ex-I* [**where** *x*=*Var l*], *auto*)
apply (*rule All-E* [**where** *x*=*Var j*, *THEN rotate6*], *auto*)
apply (*blast intro*: *ContraProve Iff-MP-same* [*OF Mem-cong* [*OF Refl*]])
apply (*metis Assume Ord-IN-Ord0 ContraProve insert-commute*)
apply (*metis Assume Neg-D thin1*)+
done
hence { (*All i* (*OrdP* (*Var i*) *IMP* ((*All2 j* (*Var i*) (*A*(*i*::= *Var j*))) *IMP A*))) }
⊢ (*All2 l* (*Var i*) (*OrdP* (*Var l*) *IMP A*(*i*::= *Var l*)))(*i*::= *Eats Zero* (*Var i*))
by (*rule Subst*, *auto*)
hence *indlem*: { *All i* (*OrdP* (*Var i*) *IMP* ((*All2 j* (*Var i*) (*A*(*i*::= *Var j*))) *IMP A*)) }

⊢ *All2 l* (*Eats Zero* (*Var i*)) (*OrdP* (*Var l*) *IMP A*(*i*::=*Var l*))
using *j l* **by** *simp*
show *?thesis*
apply (*rule Imp-I*)
apply (*rule cut-thin* [*OF indlem*, **where** *HB* = {*OrdP* (*Var i*)}])
apply (*rule All2-Eats-E*) **using** *j l*
apply *auto*
done
qed

lemma *OrdInd*:
assumes *j*: *atom* (*j*::*name*) ♯ (*i*,*A*)
and *x*: *H* ⊢ *OrdP* (*Var i*) **and** *step*: *H* ⊢ *All i* (*OrdP* (*Var i*) *IMP* (*All2 j* (*Var i*) (*A*(*i*::= *Var j*)) *IMP A*))
shows *H* ⊢ *A*
apply (*rule cut-thin* [*OF x*, **where** *HB*=*H*])
apply (*rule MP-thin* [*OF OrdInd-lemma step*])
apply (*auto simp*: *j*)
done

lemma *OrdIndH*:
assumes *atom* (*j*::*name*) ♯ (*i*,*A*)
and *H* ⊢ *All i* (*OrdP* (*Var i*) *IMP* (*All2 j* (*Var i*) (*A*(*i*::= *Var j*)) *IMP A*))
shows *insert* (*OrdP* (*Var i*)) *H* ⊢ *A*
by (*metis assms thin1 Assume OrdInd*)

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 −
obtain *k*::*name* **and** *l*::*name* **and** *m*::*name*
where *k*: *atom k* ♯ (*i*,*j*) **and** *l*: *atom l* ♯ (*i*,*j*,*k*) **and** *m*: *atom m* ♯ (*i*,*j*)
by (*metis obtain-fresh*)
show *?thesis*
proof (*rule OrdIndH* [**where** *i*=*i* **and** *j*=*k*])
show *atom k* ♯ (*i*, *?scheme*)
using *k* **by** (*force simp add*: *fresh-Pair*)
next
show {} ⊢ *All i* (*OrdP* (*Var i*) *IMP* (*All2 k* (*Var i*) (*?scheme*(*i*::= *Var k*)) *IMP ?scheme*))
using *j k*
apply *simp*
apply (*rule All-I Imp-I*)+
defer *1*
apply *auto* [*2*]

apply (*rule OrdIndH* [**where** *i*=*j* **and** *j*=*l*]) **using** *l*
— nested induction
apply (*force simp add*: *fresh-Pair*)
apply *simp*
apply (*rule All-I Imp-I*)+
prefer *2* **apply** *force*
apply (*rule Disj-3I*)
apply (*rule Equality-I*)
— Now the opposite inclusion, *Var j SUBS Var i*
apply (*rule Subset-I* [**where** *i*=*m*])
apply (*rule All2-E* [*THEN rotate4*]) **using** *l m*
apply *auto*
apply (*blast intro*: *ContraProve* [*THEN rotate3*] *OrdP-Trans*)
apply (*blast intro*: *ContraProve* [*THEN rotate3*] *Mem-cong* [*OF Hyp Refl*, *THEN Iff-MP2-same*])
— Now the opposite inclusion, *Var i SUBS Var j*
apply (*rule Subset-I* [**where** *i*=*m*])
apply (*rule All2-E* [*THEN rotate6*], *auto*)
apply (*rule All-E* [**where** *x* = *Var j*], *auto*)
apply (*blast intro*: *ContraProve* [*THEN rotate4*] *Mem-cong* [*OF Hyp Refl*, *THEN Iff-MP-same*])
apply (*blast intro*: *ContraProve* [*THEN rotate4*] *OrdP-Trans*)
done
qed
qed

lemma *OrdP-linear-imp*: {} ⊢ *OrdP x IMP OrdP y IMP x IN y OR x EQ y OR y IN x*
proof −
obtain *i*::*name* **and** *j*::*name*
where *atoms*: *atom i* ♯ (*x*,*y*) *atom j* ♯ (*x*,*y*,*i*)
by (*metis obtain-fresh*)
have { *OrdP* (*Var i*) } ⊢ (*OrdP* (*Var j*) *IMP* (*Var i IN Var j OR Var i EQ Var j OR Var j IN Var i*))(*j*::=*y*)
using *atoms* **by** (*metis All-D OrdP-linear-lemma fresh-Pair*)
hence {} ⊢ *OrdP* (*Var i*) *IMP OrdP y IMP* (*Var i IN y OR Var i EQ y OR y IN Var i*)
using *atoms* **by** *auto*
hence {} ⊢ (*OrdP* (*Var i*) *IMP OrdP y IMP* (*Var i IN y OR Var i EQ y OR y IN Var i*))(*i*::=*x*)
by (*metis Subst empty-iff*)
thus *?thesis*
using *atoms* **by** *auto*
qed

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 −
  have { *OrdP x*, *OrdP y* } ⊢ *x IN y OR x EQ y OR y IN x*
    by (*metis OrdP-linear-imp Imp-Imp-commute anti-deduction*)
  thus *?thesis*
    using *assms* **by** (*metis cut2 Disj-E cut-same*)
qed

lemma *Zero-In-SUCC*: {*OrdP k*} ⊢ *Zero IN SUCC k*
  by (*rule OrdP-linear* [*OF OrdP-Zero OrdP-SUCC-I*]) (*force simp*: *SUCC-def*)+

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*)
  by (*auto simp*: *eqvt-def OrdNotEqP-graph-aux-def*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma *OrdNotEqP-fresh-iff* [*simp*]: $a \sharp$ *OrdNotEqP x y* ⟷ $a \sharp x \wedge a \sharp y$
  by *auto*

lemma *eval-fm-OrdNotEqP* [*simp*]: *eval-fm e* (*OrdNotEqP x y*) ⟷ *Ord* $[\![x]\!]e \wedge$ *Ord* $[\![y]\!]e \wedge [\![x]\!]e \neq [\![y]\!]e$
  by (*auto simp*: *hmem-not-refl*) (*metis Ord-linear*)

lemma *OrdNotEqP-subst* [*simp*]: (*OrdNotEqP x y*)(*i*::=*t*) = *OrdNotEqP* (*subst i t x*) (*subst i t y*)
  by *simp*

lemma *OrdNotEqP-cong*: *H* ⊢ *x EQ x'* ⟹ *H* ⊢ *y EQ y'* ⟹ *H* ⊢ *OrdNotEqP x y IFF OrdNotEqP x' y'*
  by (*rule P2-cong*) *auto*

lemma *OrdNotEqP-self-contra*: {*x NEQ x*} ⊢ *Fls*
  by *auto*

lemma *OrdNotEqP-OrdP-E*: *insert* (*OrdP x*) (*insert* (*OrdP y*) *H*) ⊢ *A* ⟹ *insert* (*x NEQ y*) *H* ⊢ *A*
  by (*auto intro*: *thin1 rotate2*)

lemma *OrdNotEqP-I*: *insert* (*x EQ y*) *H* ⊢ *Fls* ⟹ *H* ⊢ *OrdP x* ⟹ *H* ⊢ *OrdP y* ⟹ *H* ⊢ *x NEQ y*
  by (*rule OrdP-linear* [*of - x y*]) (*auto intro*: *ExFalso thin1 Disj-I1 Disj-I2*)

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

lemma *OrdNotEqP-imp-Neg-Eq*: {*x NEQ y*} ⊢ *Neg* (*x EQ y*)
  by (*blast intro*: *OrdNotEqP-cong* [*THEN Iff-MP2-same*] *OrdNotEqP-self-contra*

[*of x*, *THEN cut1*])

lemma *OrdNotEqP-E*: *H* ⊢ *x EQ y* ⟹ *insert* (*x NEQ y*) *H* ⊢ *A*
  by (*metis ContraProve OrdNotEqP-imp-Neg-Eq rcut1*)

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 −
  obtain *l*::*name* **where** *l*: *atom l* ♯ (*i*,*j*,*k*)
    by (*metis obtain-fresh*)
  show *?thesis*
    apply (*rule Ind* [*of l i*]) **using** *j k l*
      apply *simp-all*
     apply (*metis Conj-E Refl Swap Imp-I*)
    apply (*rule All-I Imp-I*)+
      apply *simp-all*
    apply *clarify*
    apply (*rule thin1*)
    apply (*rule thin1* [*THEN rotate2*])
    apply (*rule Disj-EH*)
     apply (*rule Neg-Conj-E*)
      apply (*auto simp*: *All2-Eats-E1*)
     apply (*rule Ex-I* [**where** *x*=*Var l*], *auto intro*: *Mem-Eats-I2*)
      apply (*metis Assume Eq-Zero-D rotate3*)
     apply (*metis Assume EQ-imp-SUBS Neg-D thin1*)
    apply (*rule Cases* [**where** *A* = *Var j IN Var l*])
    apply (*rule Ex-I* [**where** *x*=*Var l*], *auto intro*: *Mem-Eats-I2*)
    apply (*rule Ex-I* [**where** *x*=*Var l*], *auto intro*: *Mem-Eats-I2 ContraProve*)
    apply (*rule Ex-I* [**where** *x*=*Var k*], *auto*)
    apply (*metis Assume Subset-trans OrdP-Mem-imp-Subset thin1*)
    apply (*rule Ex-I* [**where** *x*=*Var l*], *auto intro*: *Mem-Eats-I2 ContraProve*)
    apply (*metis ContraProve EQ-imp-SUBS rotate3*)
    — final case
    apply (*rule All2-Eats-E* [*THEN rotate4*], *simp-all*)
    apply (*rule Ex-I* [**where** *x*=*Var j*], *auto intro*: *Mem-Eats-I1*)
    apply (*rule All2-E* [**where** *x* = *Var k*, *THEN rotate3*], *auto*)
    apply (*rule Ex-I* [**where** *x*=*Var k*], *simp*)
    apply (*metis Assume NegNeg-I Neg-Disj-I rotate3*)
    apply (*rule cut-same* [**where** *A* = *OrdP* (*Var j*)])
    apply (*rule All2-E* [**where** *x* = *Var j*, *THEN rotate3*], *auto*)
    apply (*rule cut-same* [**where** *A* = *Var l EQ Var j OR Var l IN Var j*])
    apply (*rule OrdP-linear* [*of - Var l Var j*], *auto intro*: *Disj-CI*)
    apply (*metis Assume ContraProve rotate7*)
   apply (*metis ContraProve* [*THEN rotate4*] *EQ-imp-SUBS Subset-trans rotate3*)
   apply (*blast intro*: *ContraProve* [*THEN rotate4*] *OrdP-Mem-imp-Subset Iff-MP2-same*

[*OF Mem-cong*])
  done
qed

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 −
  obtain *i*::*name* **where** *i*: *atom i* ♯ (*x*,*j*,*k*)
    by (*metis obtain-fresh*)
  have {} ⊢ ((*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*)))))(*i*::=*x*)
    apply (*rule Subst* [*OF OrdP-set-max-lemma*])
    using *i k* **apply** *auto*
    done
  hence { *Neg* (*x EQ Zero*) *AND* (*All2 j x* (*OrdP* (*Var j*))) }
      ⊢ *Ex j* (*Var j IN x AND* (*All2 k x* (*Var k SUBS Var j*)))
   using *i j k* **by** *simp* (*metis anti-deduction*)
  hence { *All2 j x* (*OrdP* (*Var j*)), *Neg* (*x EQ Zero*) }
        ⊢ *Ex j* (*Var j IN x AND* (*All2 k x* (*Var k SUBS Var j*)))
    by (*rule cut1*) (*metis Assume Conj-I thin1*)
  moreover have { *OrdP x* } ⊢ *All2 j x* (*OrdP* (*Var j*)) **using** *j*
    by *auto* (*metis Assume Ord-IN-Ord thin1*)
  ultimately show *?thesis*
   by (*metis rcut1*)
qed

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 −
  obtain *j*::*name* **and** *k*::*name* **where** *j*: *atom j* ♯ (*x*,*p*) **and** *k*: *atom k* ♯ (*x*,*j*,*p*)
    by (*metis obtain-fresh*)
  show *?thesis*
    apply (*rule cut-same* [*OF OrdP-max-imp* [*of j x k*]])
    using *p j k* **apply** *auto*
    apply (*rule Ex-I* [**where** *x*=*Var j*], *auto*)
     apply (*metis Assume Ord-IN-Ord thin1*)
    apply (*rule cut-same* [**where** *A* = *OrdP* (*SUCC* (*Var j*))])
    apply (*metis Assume Ord-IN-Ord0 OrdP-SUCC-I rotate2 thin1*)
   apply (*rule OrdP-linear* [**where** *x* = *x*, *OF* - *Assume*], *auto intro*!: *Mem-SUCC-EH*)
    apply (*metis Mem-not-sym rotate3*)
     apply (*rule Mem-non-refl*, *blast intro*: *Mem-cong* [*OF Assume Refl*, *THEN*

Iff-MP2-same])
  apply (*force intro*: *thin1 All2-E* [**where** x = *SUCC* (*Var j*), *THEN rotate4*])
  done
qed

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*))
 by (*metis Disj-CI Assume cut2* [*OF OrdP-cases-lemma* [*OF p*]] *Swap thin1*)

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*
 by (*rule cut-same* [*OF OrdP-cases-disj* [*of k*]]) (*auto simp*: *insert-commute intro*: *thin1*)

lemma *OrdInd2-lemma*:
 { *OrdP* (*Var i*), *A*(*i*::= *Zero*), (*All i* (*OrdP* (*Var i*) *IMP A IMP* (*A*(*i*::= *SUCC* (*Var i*))))) } ⊢ *A*
proof −
 obtain *j*::*name* **and** *k*::*name* **where** *atoms*: *atom j* ♯ (*i*,*A*) *atom k* ♯ (*i*,*j*,*A*)
  by (*metis obtain-fresh*)
 show *?thesis*
 apply (*rule OrdIndH* [**where** *i*=*i* **and** *j*=*j*])
 using *atoms* **apply** *auto*
 apply (*rule OrdP-cases-E* [**where** *k*=*k*, *THEN rotate3*])
 apply (*rule ContraProve* [*THEN rotate2*]) **using** *Var-Eq-imp-subst-Iff*
 apply (*metis Assume AssumeH*(*3*) *Iff-MP-same*)
 apply (*rule Ex-I* [**where** *x*=*Var k*], *simp*)
 apply (*rule Neg-Imp-I*, *blast*)
 apply (*rule cut-same* [**where** *A* = *A*(*i*::=*Var k*)])
 apply (*rule All2-E* [**where** *x* = *Var k*, *THEN rotate5*])
 apply (*auto intro*: *Mem-SUCC-I2 Mem-cong* [*OF Refl*, *THEN Iff-MP2-same*])
 apply (*rule ContraProve* [*THEN rotate5*])
 by (*metis Assume Iff-MP-left′ Var-Eq-subst-Iff thin1*)
qed

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*
 by (*metis cut3* [*OF OrdInd2-lemma*] *assms*)

lemma *OrdInd2H*:
 assumes *H* ⊢ *A*(*i*::= *Zero*)

and $H \vdash$ *All i* (*OrdP* (*Var i*) *IMP A IMP* (*A*(*i*::= *SUCC* (*Var i*))))
shows *insert* (*OrdP* (*Var i*)) $H \vdash A$
by (*metis assms thin1 Assume OrdInd2*)

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 $r \equiv \forall z \in r.\ \forall z' \in r.\ \exists x\ y\ x'\ y'.\ z = \langle x,y \rangle \wedge z' = \langle x',y' \rangle \wedge (x{=}x' \longrightarrow y{=}y')$

definition *hfun-sigma-ord* **where**
hfun-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* $\Rightarrow$ *fm*
where $[\![$*atom* $z \sharp (r,z',x,y,x',y')$; *atom* $z' \sharp (r,x,y,x',y')$;
atom $x \sharp (r,y,x',y')$; *atom* $y \sharp (r,x',y')$; *atom* $x' \sharp (r,y')$; *atom* $y' \sharp (r)$ $]\!]$
$\Longrightarrow$
HFun-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'*))))))))
by (*auto simp*: *eqvt-def HFun-Sigma-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *HFun-Sigma-fresh-iff* [*simp*]: $a \sharp$ *HFun-Sigma* $r \longleftrightarrow a \sharp r$ (**is** *?thesis1*)
and *eval-fm-HFun-Sigma* [*simp*]:
eval-fm e (*HFun-Sigma r*) $\longleftrightarrow$ *hfun-sigma-ord* $[\![r]\!]e$ (**is** *?thesis2*)
proof –
obtain x::*name* **and** y::*name* **and** z::*name* **and** x'::*name* **and** y'::*name* **and** z'::*name*
where *atom* $z \sharp (r,z',x,y,x',y')$ *atom* $z' \sharp (r,x,y,x',y')$
atom $x \sharp (r,y,x',y')$ *atom* $y \sharp (r,x',y')$
atom $x' \sharp (r,y')$ *atom* $y' \sharp (r)$
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2*
by (*auto simp*: *HBall-def hfun-sigma-ord-def*, *metis+*)
qed

lemma *HFun-Sigma-subst* [*simp*]: (*HFun-Sigma r*)(*i*::=*t*) = *HFun-Sigma* (*subst i t r*)
proof −
  obtain *x*::*name* **and** *y*::*name* **and** *z*::*name* **and** *x'*::*name* **and** *y'*::*name* **and** *z'*::*name*
    where *atom z* ♯ (*r*,*t*,*i*,*z'*,*x*,*y*,*x'*,*y'*) *atom z'* ♯ (*r*,*t*,*i*,*x*,*y*,*x'*,*y'*)
        atom x ♯ (*r*,*t*,*i*,*y*,*x'*,*y'*) *atom y* ♯ (*r*,*t*,*i*,*x'*,*y'*)
        atom x' ♯ (*r*,*t*,*i*,*y'*) *atom y'* ♯ (*r*,*t*,*i*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *HFun-Sigma.simps* [*of z - z' x y x' y'*])
qed

lemma *HFun-Sigma-Zero*: *H* ⊢ *HFun-Sigma Zero*
proof −
  obtain *x*::*name* **and** *y*::*name* **and** *z*::*name* **and** *x'*::*name* **and** *y'*::*name* **and** *z'*::*name* **and** *z''*::*name*
    where *atom z''* ♯ (*z*,*z'*,*x*,*y*,*x'*,*y'*) *atom z* ♯ (*z'*,*x*,*y*,*x'*,*y'*) *atom z'* ♯ (*x*,*y*,*x'*,*y'*)
      atom x ♯ (*y*,*x'*,*y'*) *atom y* ♯ (*x'*,*y'*) *atom x'* ♯ *y'*
    by (*metis obtain-fresh*)
  hence {} ⊢ *HFun-Sigma Zero*
    by (*auto simp*: *HFun-Sigma.simps* [*of z - z' x y x' y'*])
  thus *?thesis*
    by (*metis thin0*)
qed

lemma *Subset-HFun-Sigma*: {*HFun-Sigma s*, *s' SUBS s*} ⊢ *HFun-Sigma s'*
proof −
  obtain *x*::*name* **and** *y*::*name* **and** *z*::*name* **and** *x'*::*name* **and** *y'*::*name* **and** *z'*::*name* **and** *z''*::*name*
    where *atom z''* ♯ (*z*,*z'*,*x*,*y*,*x'*,*y'*,*s*,*s'*)
      atom z ♯ (*z'*,*x*,*y*,*x'*,*y'*,*s*,*s'*) *atom z'* ♯ (*x*,*y*,*x'*,*y'*,*s*,*s'*)
      atom x ♯ (*y*,*x'*,*y'*,*s*,*s'*) *atom y* ♯ (*x'*,*y'*,*s*,*s'*)
      atom x' ♯ (*y'*,*s*,*s'*) *atom y'* ♯ (*s*,*s'*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    apply (*auto simp*: *HFun-Sigma.simps* [*of z - z' x y x' y'*])
    apply (*rule Ex-I* [**where** *x*=*Var z*], *auto*)
    apply (*blast intro*: *Subset-D ContraProve*)
    apply (*rule All-E* [**where** *x*=*Var z'*], *auto intro*: *Subset-D ContraProve*)
    done
qed

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 −
  obtain x'::*name* **and** y'::*name* **and** z::*name* **and** z'::*name*
    where *atoms*: *atom* $z \sharp (z',x',y',x,y,a,r)$ *atom* $z' \sharp (x',y',x,y,a,r)$
      atom $x' \sharp (y',x,y,a,r)$ *atom* $y' \sharp (x,y,a,r)$
    by (*metis obtain-fresh*)
  hence {*HFun-Sigma r*, *a IN r*} ⊢ *?concl* **using** *xy*
    apply (*auto simp*: *HFun-Sigma.simps* [*of z r z' x y x' y'*])
    apply (*rule All-E* [**where** *x=a*], *auto*)
    apply (*rule All-E* [**where** *x=a*], *simp*)
    apply (*rule Imp-E*, *blast*)
    apply (*rule Ex-EH Conj-EH*)+
    apply *simp-all*
    apply (*rule Ex-I* [**where** *x=Var x*], *simp*)
    apply (*rule Ex-I* [**where** *x=Var y*], *auto*)
    done
  thus *?thesis*
    by (*rule cut2*) (*rule assms*)+
qed

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* $x \sharp (r,d,y,z)$; *atom* $y \sharp (r,d,z)$; *atom* $z \sharp (r,d)$⟧ ⟹
    HDomain-Incl r d = *All2 x d* (*Ex y* (*Ex z* (*Var z IN r AND Var z EQ HPair* (*Var x*) (*Var y*))))
 by (*auto simp*: *eqvt-def HDomain-Incl-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma
  shows *HDomain-Incl-fresh-iff* [*simp*]:
    $a \sharp$ *HDomain-Incl r d* ⟷ $a \sharp r \wedge a \sharp d$ (**is** *?thesis1*)
  and *eval-fm-HDomain-Incl* [*simp*]:
    eval-fm e (*HDomain-Incl r d*) ⟷ ⟦*d*⟧*e* ≤ *hdomain* ⟦*r*⟧*e* (**is** *?thesis2*)
proof −
  obtain *x*::*name* **and** *y*::*name* **and** *z*::*name*
    where *atom* $x \sharp (r,d,y,z)$ *atom* $y \sharp (r,d,z)$ *atom* $z \sharp (r,d)$
    by (*metis obtain-fresh*)
  thus *?thesis1 ?thesis2*
    by (*auto simp*: *HDomain-Incl.simps* [*of x - - y z*] *hdomain-def*)
qed

lemma *HDomain-Incl-subst* [*simp*]:
    (*HDomain-Incl r d*)(*i*::=*t*) = *HDomain-Incl* (*subst i t r*) (*subst i t d*)
proof −
  obtain *x*::*name* **and** *y*::*name* **and** *z*::*name*

where *atom x* ♯ (*r*,*d*,*y*,*z*,*t*,*i*) *atom y* ♯ (*r*,*d*,*z*,*t*,*i*) *atom z* ♯ (*r*,*d*,*t*,*i*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *HDomain-Incl.simps* [*of x - - y z*])
qed

lemma *HDomain-Incl-Subset-lemma*: { *HDomain-Incl r k*, *k′ SUBS k* } ⊢ *HDomain-Incl r k′*
proof −
obtain *x*::*name* **and** *y*::*name* **and** *z*::*name*
where *atom x* ♯ (*r*,*k*,*k′*,*y*,*z*) *atom y* ♯ (*r*,*k*,*k′*,*z*) *atom z* ♯ (*r*,*k*,*k′*)
by (*metis obtain-fresh*)
thus *?thesis*
apply (*simp add*: *HDomain-Incl.simps* [*of x - - y z*], *auto*)
apply (*rule Ex-I* [**where** *x* = *Var x*], *auto intro*: *ContraProve Subset-D*)
done
qed

lemma *HDomain-Incl-Subset*: *H* ⊢ *HDomain-Incl r k* ⟹ *H* ⊢ *k′ SUBS k* ⟹ *H* ⊢ *HDomain-Incl r k′*
by (*metis HDomain-Incl-Subset-lemma cut2*)

lemma *HDomain-Incl-Mem-Ord*: *H* ⊢ *HDomain-Incl r k* ⟹ *H* ⊢ *k′ IN k* ⟹ *H* ⊢ *OrdP k* ⟹ *H* ⊢ *HDomain-Incl r k′*
by (*metis HDomain-Incl-Subset OrdP-Mem-imp-Subset*)

lemma *HDomain-Incl-Zero* [*simp*]: *H* ⊢ *HDomain-Incl r Zero*
proof −
obtain *x*::*name* **and** *y*::*name* **and** *z*::*name*
where *atom x* ♯ (*r*,*y*,*z*) *atom y* ♯ (*r*,*z*) *atom z* ♯ *r*
by (*metis obtain-fresh*)
hence {} ⊢ *HDomain-Incl r Zero*
by (*auto simp*: *HDomain-Incl.simps* [*of x - - y z*])
thus *?thesis*
by (*metis thin0*)
qed

lemma *HDomain-Incl-Eats*: { *HDomain-Incl r d* } ⊢ *HDomain-Incl* (*Eats r* (*HPair d d′*)) (*SUCC d*)
proof −
obtain *x*::*name* **and** *y*::*name* **and** *z*::*name*
where *x*: *atom x* ♯ (*r*,*d*,*d′*,*y*,*z*) **and** *y*: *atom y* ♯ (*r*,*d*,*d′*,*z*) **and** *z*: *atom z* ♯ (*r*,*d*,*d′*)
by (*metis obtain-fresh*)
thus *?thesis*
apply (*auto simp*: *HDomain-Incl.simps* [*of x - - y z*] *intro*!: *Mem-SUCC-EH*)
apply (*rule Ex-I* [**where** *x* = *Var x*], *auto*)
apply (*rule Ex-I* [**where** *x* = *Var y*], *auto*)
apply (*rule Ex-I* [**where** *x* = *Var z*], *auto intro*: *Mem-Eats-I1*)

apply (*rule rotate2* [*OF Swap*])
apply (*rule Ex-I* [**where** $x = d'$], *auto*)
apply (*rule Ex-I* [**where** $x = HPair\ d\ d'$], *auto intro*: *Mem-Eats-I2 HPair-cong Sym*)
done
qed

lemma *HDomain-Incl-Eats-I*: $H \vdash$ *HDomain-Incl r d* $\Longrightarrow H \vdash$ *HDomain-Incl* (*Eats r* (*HPair d d'*)) (*SUCC d*)
by (*metis HDomain-Incl-Eats cut1*)

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*)) $H \vdash A$
apply (*rule Equality-E*) **using** *assms*
apply (*auto intro*!: *Zero-SubsetE rotate2* [*of a IN b*])
apply (*rule-tac* [!] *rotate3*)
apply (*auto intro*!: *Zero-SubsetE rotate2* [*of a IN b*])
apply (*metis Sym-L insert-commute thin1*)+
done

lemma *HFST*: {*HPair a b EQ HPair c d*} $\vdash$ *a EQ c*
unfolding *HPair-def* **by** (*metis Assume Doubleton-E thin1*)

lemma *b-EQ-d-1*: {*a EQ c*, *a EQ d*, *b EQ c*} $\vdash$ *b EQ d*
by (*metis Assume thin1 Sym Trans*)

lemma *HSND*: {*HPair a b EQ HPair c d*} $\vdash$ *b EQ d*
unfolding *HPair-def*
by (*metis AssumeH*(*2*) *Doubleton-E b-EQ-d-1 rotate3 thin2*)

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*) $H \vdash A$
by (*metis Conj-E* [*OF assms*] *Conj-I* [*OF HFST HSND*] *rcut1*)

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* ⊢ *HPair a b IN r*
and *b′*: *H* ⊢ *HPair a b′ IN r*
shows *H* ⊢ *b EQ b′*
proof −
obtain *x*::*name* **and** *y*::*name* **and** *z*::*name* **and** *x′*::*name* **and** *y′*::*name* **and** *z′*::*name*
where *atoms*: *atom z* ♯ (*r*,*a*,*b*,*b′*,*z′*,*x*,*y*,*x′*,*y′*) *atom z′* ♯ (*r*,*a*,*b*,*b′*,*x*,*y*,*x′*,*y′*)
atom x ♯ (*r*,*a*,*b*,*b′*,*y*,*x′*,*y′*) *atom y* ♯ (*r*,*a*,*b*,*b′*,*x′*,*y′*)
atom x′ ♯ (*r*,*a*,*b*,*b′*,*y′*) *atom y′* ♯ (*r*,*a*,*b*,*b′*)
by (*metis obtain-fresh*)
hence *d1*: *H* ⊢ *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′*))))))))
using *r HFun-Sigma.simps* [*of z r z′ x y x′ y′*]
by *simp*
have *d2*: *H* ⊢ *All2 z′ r* (*Ex x* (*Ex y* (*Ex x′* (*Ex y′*
(*HPair a b 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′*)))))))
using *All-D* [**where** *x* = *HPair a b*, *OF d1*] *atoms*
by *simp* (*metis MP-same b*)
have *d4*: *H* ⊢ *Ex x* (*Ex y* (*Ex x′* (*Ex y′*
(*HPair a b EQ HPair* (*Var x*) (*Var y*) *AND HPair a b′ 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′*))))))
using *All-D* [**where** *x* = *HPair a b′*, *OF d2*] *atoms*
by *simp* (*metis MP-same b′*)
have *d′*: { *Ex x* (*Ex y* (*Ex x′* (*Ex y′*
(*HPair a b EQ HPair* (*Var x*) (*Var y*) *AND HPair a b′ 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′*)))))) } ⊢ *b EQ b′*
using *atoms*
by (*auto intro*: *ContraProve Trans Sym*)
thus *?thesis*
by (*rule cut-thin* [*OF d4*], *auto*)
qed

3.14 *SUCC* is Provably Injective

lemma *SUCC-SUBS-lemma*: {*SUCC x SUBS SUCC y*} ⊢ *x SUBS y*
apply (*rule obtain-fresh* [**where** *x*=(*x*,*y*)])
apply (*auto simp*: *SUCC-def*)
prefer *2* **apply** (*metis Assume Conj-E1 Extensionality Iff-MP-same*)
apply (*auto intro*!: *Subset-I*)

apply (*blast intro*: *Set-MP cut-same* [*OF Mem-cong* [*OF Refl Assume*, *THEN Iff-MP2-same*]]
Mem-not-sym thin2)
done

lemma *SUCC-SUBS*: *insert* (*SUCC x SUBS SUCC y*) $H \vdash$ *x SUBS y*
by (*metis Assume SUCC-SUBS-lemma cut1*)

lemma *SUCC-inject*: *insert* (*SUCC x EQ SUCC y*) $H \vdash$ *x EQ y*
by (*metis Equality-I EQ-imp-SUBS SUCC-SUBS Sym-L cut1*)

lemma *SUCC-inject-E* [*intro*!]: *insert* (*x EQ y*) $H \vdash A \Longrightarrow$ *insert* (*SUCC x EQ SUCC y*) $H \vdash A$
by (*metis SUCC-inject cut-same insert-commute thin1*)

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*
apply (*rule OrdP-linear* [*of - SUCC x SUCC y*])
apply (*auto intro*!: *Mem-SUCC-EH intro*: *OrdP-SUCC-I Ord-IN-Ord0*)
apply (*metis Hyp Mem-SUCC-I1 Mem-not-sym cut-same insertCI*)
apply (*metis Assume EQ-imp-SUBS Mem-SUCC-I1 Mem-non-refl Subset-D thin1*)

apply (*blast intro*: *cut-same* [*OF Mem-cong* [*THEN Iff-MP2-same*]])
done

lemma *OrdP-IN-SUCC*: $H \vdash$ *OrdP x* $\Longrightarrow H \vdash$ *y IN x* $\Longrightarrow H \vdash$ *SUCC y IN SUCC x*
by (*rule cut2* [*OF OrdP-IN-SUCC-lemma*])

lemma *OrdP-IN-SUCC-D-lemma*: {*OrdP x*, *SUCC y IN SUCC x*} $\vdash$ *y IN x*
apply (*rule OrdP-linear* [*of - x y*], *auto*)
apply (*metis Assume AssumeH*(*2*) *Mem-SUCC-Refl OrdP-SUCC-I Ord-IN-Ord*)
apply (*rule Mem-SUCC-E* [*THEN rotate3*])
apply (*blast intro*: *Mem-SUCC-Refl OrdP-Trans*)
apply (*metis AssumeH*(*2*) *EQ-imp-SUBS Mem-SUCC-I1 Mem-non-refl Subset-D*)
apply (*metis EQ-imp-SUBS Mem-SUCC-I2 Mem-SUCC-EH*(*2*) *Mem-SUCC-I1 Refl SUCC-Subset-Ord-lemma Subset-D thin1*)
done

lemma *OrdP-IN-SUCC-D*: $H \vdash$ *OrdP x* $\Longrightarrow H \vdash$ *SUCC y IN SUCC x* $\Longrightarrow H \vdash$ *y IN x*
by (*rule cut2* [*OF OrdP-IN-SUCC-D-lemma*])

lemma *OrdP-IN-SUCC-Iff*: $H \vdash OrdP\ y \Longrightarrow H \vdash SUCC\ x\ IN\ SUCC\ y\ IFF\ x\ IN\ y$
by (*metis Assume Iff-I OrdP-IN-SUCC OrdP-IN-SUCC-D assms thin1*)

3.15 The predicate *LstSeqP*

lemma *hfun-sigma-ord-iff*: *hfun-sigma-ord s* $\longleftrightarrow$ *OrdDom s* $\wedge$ *hfun-sigma s*
by (*auto simp*: *hfun-sigma-ord-def OrdDom-def hfun-sigma-def HBall-def*, *metis+*)

lemma *hfun-sigma-iff*: *hfun-sigma r* $\longleftrightarrow$ *hfunction r* $\wedge$ *hrelation r*
by (*auto simp add*: *HBall-def hfun-sigma-def hfunction-def hrelation-def is-hpair-def*, *metis+*)

lemma *Seq-iff*: *Seq r d* $\longleftrightarrow$ *d* $\leq$ *hdomain r* $\wedge$ *hfun-sigma r*
by (*auto simp*: *Seq-def hfun-sigma-iff*)

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*
by (*auto simp*: *OrdDom-def LstSeq-def Seq-iff hfun-sigma-ord-iff*)

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*
by (*auto simp*: *eqvt-def LstSeqP-graph-aux-def*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *LstSeqP-fresh-iff* [*simp*]:
$a \sharp$ *LstSeqP s k y* $\longleftrightarrow a \sharp s \wedge a \sharp k \wedge a \sharp y$ (**is** *?thesis1*)
and *eval-fm-LstSeqP* [*simp*]:
eval-fm e (*LstSeqP s k y*) $\longleftrightarrow$ *LstSeq* $[\![s]\!]e$ $[\![k]\!]e$ $[\![y]\!]e$ (**is** *?thesis2*)
proof −
show *?thesis1 ?thesis2*
by (*auto simp*: *LstSeq-iff OrdDom-def hfun-sigma-ord-iff*)
qed

lemma *LstSeqP-subst* [*simp*]:
(*LstSeqP s k y*)(*i*::=*t*) = *LstSeqP* (*subst i t s*) (*subst i t k*) (*subst i t y*)
by (*auto simp*: *fresh-Pair fresh-at-base*)

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*

using *assms* **by** (*auto simp*: *insert-commute*)

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 $H \vdash LstSeqP\ s\ k\ y\ IFF\ LstSeqP\ s'\ k'\ y'$
by (*rule P3-cong* [*OF - assms*], *auto*)

lemma *LstSeqP-OrdP*: $H \vdash LstSeqP\ r\ k\ y \Longrightarrow H \vdash OrdP\ k$
by (*metis Conj-E1 LstSeqP.simps*)

lemma *LstSeqP-Mem-lemma*: $\{\ LstSeqP\ r\ k\ y,\ HPair\ k'\ z\ IN\ r,\ k'\ IN\ k\ \} \vdash LstSeqP\ r\ k'\ z$
by (*auto simp*: *LstSeqP.simps intro*: *Ord-IN-Ord OrdP-SUCC-I OrdP-IN-SUCC HDomain-Incl-Mem-Ord*)

lemma *LstSeqP-Mem*: $H \vdash LstSeqP\ r\ k\ y \Longrightarrow H \vdash HPair\ k'\ z\ IN\ r \Longrightarrow H \vdash k'\ IN\ k \Longrightarrow H \vdash LstSeqP\ r\ k'\ z$
by (*rule cut3* [*OF LstSeqP-Mem-lemma*])

lemma *LstSeqP-imp-Mem*: $H \vdash LstSeqP\ s\ k\ y \Longrightarrow H \vdash HPair\ k\ y\ IN\ s$
by (*auto simp*: *LstSeqP.simps*) (*metis Conj-E2*)

lemma *LstSeqP-SUCC*: $H \vdash LstSeqP\ r\ (SUCC\ d)\ y \Longrightarrow H \vdash HPair\ d\ z\ IN\ r \Longrightarrow H \vdash LstSeqP\ r\ d\ z$
by (*metis LstSeqP-Mem Mem-SUCC-I2 Refl*)

lemma *LstSeqP-EQ*: $[\![H \vdash LstSeqP\ s\ k\ y;\ H \vdash HPair\ k\ y'\ IN\ s]\!] \Longrightarrow H \vdash y\ EQ\ y'$
by (*metis AssumeH*(*2*) *HFun-Sigma-E LstSeqP-E cut1 insert-commute*)

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*)
unfolding *ground-aux-def*
by (*simp-all add*: *supp-at-base*)

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*}))
by (*auto simp*: *ground-fm-aux-def ground-aux-def supp-conv-fresh*)

lemma *ground-fresh*[*simp*]:
ground t ⟹ *atom i* ♯ *t*
ground-fm A ⟹ *atom i* ♯ *A*
unfolding *ground-aux-def ground-fm-aux-def fresh-def*
by *simp-all*

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*
by (*simp-all add*: *fresh-star-def*)

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*
by (*metis Sigma-fm-def Iff-trans order-trans*)

lemma *ss-fm-imp-Sigma-fm* [*intro*]: *ss-fm A* ⟹ *Sigma-fm A*
by (*metis Iff-refl Sigma-fm-def order-refl*)

lemma *Sigma-fm-Fls* [*iff*]: *Sigma-fm Fls*
by (*rule Sigma-fm-Iff* [*of - Ex i* (*Var i IN Var i*)]) *auto*

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 −
  obtain *SA SB* **where** *ss-fm SA* $\{\} \vdash$ *A IFF SA supp SA* $\subseteq$ *supp A*
                and *ss-fm SB* $\{\} \vdash$ *B IFF SB supp SB* $\subseteq$ *supp B*
    using *assms* **by** (*auto simp add*: *Sigma-fm-def*)
  then show *Sigma-fm* (*A AND B*) *Sigma-fm* (*A OR B*) *Sigma-fm* (*Ex i A*)
    apply (*auto simp*: *Sigma-fm-def*)
    apply (*metis ss-fm.ConjI Conj-cong Un-mono supp-Conj*)
    apply (*metis ss-fm.DisjI Disj-cong Un-mono fm.supp*(*3*))
    apply (*rule exI* [**where** *x* = *Ex i SA*])
    apply (*auto intro*!: *Ex-cong*)
    done
qed

lemma *Sigma-fm-All2-Var*:
  assumes *H0*: *Sigma-fm A* **and** *ij*: *atom j* $\sharp$ (*i*,*A*)
  shows *Sigma-fm* (*All2 i* (*Var j*) *A*)
proof −
  obtain *SA* **where** *SA*: *ss-fm SA* $\{\} \vdash$ *A IFF SA supp SA* $\subseteq$ *supp A*
    using *H0* **by** (*auto simp add*: *Sigma-fm-def*)
  show *Sigma-fm* (*All2 i* (*Var j*) *A*)
    apply (*rule Sigma-fm-Iff* [*of - All2 i* (*Var j*) *SA*])
    apply (*metis All2-cong Refl SA*(*2*) *emptyE*)
    using *SA ij*
    apply (*auto simp*: *supp-conv-fresh subset-iff*)
    apply (*metis ss-fm.All2I fresh-Pair ss-fm-imp-Sigma-fm*)
    done
qed

4.3 Lemma 2.2: Atomic formulas are Sigma-formulas

lemma *Eq-Eats-Iff*:
   assumes [*unfolded fresh-Pair*, *simp*]: *atom i* $\sharp$ (*z*,*x*,*y*)
   shows $\{\} \vdash$ *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 (*rule Iff-I*, *auto*)
  have {*Var i IN z*, *z EQ Eats x y*} $\vdash$ *Var i IN Eats x y*
    by (*metis Assume Iff-MP-left Iff-sym Mem-cong Refl*)
  then show {*Var i IN z*, *z EQ Eats x y*} $\vdash$ *Var i IN x OR Var i EQ y*
    by (*metis Iff-MP-same Mem-Eats-Iff*)
next
  show {*z EQ Eats x y*} $\vdash$ *x SUBS z*
    by (*metis Iff-MP2-same Subset-cong* [*OF Refl Assume*] *Subset-Eats-I*)

```
next
  show {z EQ Eats x y} ⊢ y IN z
    by (metis Iff-MP2-same Mem-cong Assume Refl Mem-Eats-I2)
next
  show {x SUBS z, y IN z, All2 i z (Var i IN x OR Var i EQ y)} ⊢ z EQ Eats x y
       (is {-, -, ?allHyp} ⊢ -)
    apply (rule Eq-Eats-iff [OF assms, THEN Iff-MP2-same], auto)
    apply (rule Ex-I [where x=Var i])
   apply (auto intro: Subset-D Mem-cong [OF Assume Refl, THEN Iff-MP2-same])
    done
qed

lemma Subset-Zero-sf: Sigma-fm (Var i SUBS Zero)
proof −
  obtain j::name where j: atom j ♯ i
    by (rule obtain-fresh)
  hence Subset-Zero-Iff: {} ⊢ Var i SUBS Zero IFF (All2 j (Var i) Fls)
    by (auto intro!: Subset-I [of j] intro: Eq-Zero-D Subset-Zero-D All2-E [THEN
rotate2])
  thus ?thesis using j
    by (auto simp: supp-conv-fresh
            intro!: Sigma-fm-Iff [OF Subset-Zero-Iff] Sigma-fm-All2-Var)
qed

lemma Eq-Zero-sf: Sigma-fm (Var i EQ Zero)
proof −
  obtain j::name where atom j ♯ i
    by (rule obtain-fresh)
  thus ?thesis
    by (auto simp add: supp-conv-fresh
          intro!: Sigma-fm-Iff [OF - - Subset-Zero-sf] Subset-Zero-D EQ-imp-SUBS)
qed

lemma theorem-sf: assumes {} ⊢ A shows Sigma-fm A
proof −
  obtain i::name and j::name
    where ij: atom i ♯ (j,A) atom j ♯ A
    by (metis obtain-fresh)
  show ?thesis
    apply (rule Sigma-fm-Iff [where A = Ex i (Ex j (Var i IN Var j))])
    using ij
    apply (auto simp: )
    apply (rule Ex-I [where x=Zero], simp)
    apply (rule Ex-I [where x=Eats Zero Zero])
    apply (auto intro: Mem-Eats-I2 assms thin0)
    done
qed
```

The subset relation

lemma *Var-Subset-sf*: *Sigma-fm* (*Var i SUBS Var j*)
proof –
  obtain *k*::*name* **where** *k*: *atom* (*k*::*name*) ♯ (*i*,*j*)
    by (*metis obtain-fresh*)
  thus *?thesis*
  proof (*cases i*=*j*)
    case *True* **thus** *?thesis* **using** *k*
      by (*auto intro*!: *theorem-sf Subset-I* [**where** *i*=*k*])
  next
    case *False* **thus** *?thesis* **using** *k*
      by (*auto simp*: *ss-fm-imp-Sigma-fm Subset.simps* [*of k*] *ss-fm.intros*)
  qed
qed

lemma *Zero-Mem-sf*: *Sigma-fm* (*Zero IN Var i*)
proof –
  obtain *j*::*name* **where** *atom j* ♯ *i*
    by (*rule obtain-fresh*)
  hence *Zero-Mem-Iff*: {} ⊢ *Zero IN Var i IFF* (*Ex j* (*Var j EQ Zero AND Var j IN Var i*))
    by (*auto intro*: *Ex-I* [**where** *x* = *Zero*] *Mem-cong* [*OF Assume Refl*, *THEN Iff-MP-same*])
  show *?thesis*
    by (*auto intro*!: *Sigma-fm-Iff* [*OF Zero-Mem-Iff*] *Eq-Zero-sf*)
qed

lemma *ijk*: $i + k < Suc\ (i + j + k)$
  by *arith*

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*)
apply *auto*
apply (*rule Ex-I* [**where** *x*=*Var j*], *auto*)
apply (*rule Ex-I* [**where** *x*=*Var i*], *auto intro*: *ContraProve Mem-cong* [*THEN Iff-MP-same*])
done

lemma *Sigma-fm-All2-fresh*:
  assumes *Sigma-fm A i*≠*j*
    shows *Sigma-fm* (*All2 i* (*Var j*) *A*)
proof –
  obtain *j'*::*name* **where** *j'*: *atom j'* ♯ (*i*,*j*,*A*)
    by (*metis obtain-fresh*)
  show *Sigma-fm* (*All2 i* (*Var j*) *A*)
    apply (*rule Sigma-fm-Iff* [*OF All2-term-Iff-fresh* [*OF - j'*]])
    using *assms j'*
    apply (*auto simp*: *supp-conv-fresh Var-Subset-sf*
            intro!: *Sigma-fm-All2-Var Sigma-fm-Iff* [*OF Extensionality - -*])
    done

qed

lemma *Subset-Eats-sf*:
  assumes $\bigwedge j$::*name*. *Sigma-fm* (*Var j IN t*)
      and $\bigwedge k$::*name*. *Sigma-fm* (*Var k EQ u*)
  shows *Sigma-fm* (*Var i SUBS Eats t u*)
proof −
  obtain *k*::*name* **where** *k*: *atom k* ♯ (*t*,*u*,*Var i*)
    by (*metis obtain-fresh*)
  hence {} ⊢ *Var i SUBS Eats t u IFF All2 k* (*Var i*) (*Var k IN t OR Var k EQ u*)
    apply (*auto simp*: *fresh-Pair intro*: *Set-MP Disj-I1 Disj-I2*)
    apply (*force intro*!: *Subset-I* [**where** *i*=*k*] *intro*: *All2-E*′ [*OF Hyp*] *Mem-Eats-I1 Mem-Eats-I2*)
    done
  thus *?thesis*
    apply (*rule Sigma-fm-Iff*)
    using *k*
    apply (*auto intro*!: *Sigma-fm-All2-fresh simp add*: *assms fresh-Pair supp-conv-fresh fresh-at-base*)
    done
qed

lemma *Eq-Eats-sf*:
  assumes $\bigwedge j$::*name*. *Sigma-fm* (*Var j EQ t*)
      and $\bigwedge k$::*name*. *Sigma-fm* (*Var k EQ u*)
  shows *Sigma-fm* (*Var i EQ Eats t u*)
proof −
  obtain *j*::*name* **and** *k*::*name* **and** *l*::*name*
    where *atoms*: *atom j* ♯ (*t*,*u*,*i*) *atom k* ♯ (*t*,*u*,*i*,*j*) *atom l* ♯ (*t*,*u*,*i*,*j*,*k*)
    by (*metis obtain-fresh*)
  hence {} ⊢ *Var i EQ Eats t u IFF*
          Ex j (*Ex k* (*Var i EQ Eats* (*Var j*) (*Var k*) *AND Var j EQ t AND Var k EQ u*))
    apply *auto*
    apply (*rule Ex-I* [**where** *x*=*t*], *simp*)
    apply (*rule Ex-I* [**where** *x*=*u*], *auto intro*: *Trans Eats-cong*)
    done
  thus *?thesis*
    apply (*rule Sigma-fm-Iff*)
    apply (*auto simp*: *assms supp-at-base*)
    apply (*rule Sigma-fm-Iff* [*OF Eq-Eats-Iff* [*of l*]])
    using *atoms*
    apply (*auto simp*: *supp-conv-fresh fresh-at-base Var-Subset-sf*
          intro!: *Sigma-fm-All2-Var Sigma-fm-Iff* [*OF Extensionality - -*])
    done
qed

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 −
  obtain j::name where j: atom j ♯ (t,u,Var i)
    by (metis obtain-fresh)
  hence {} ⊢ Eats t u IN Var i IFF
            Ex j (Var j IN Var i AND Var j EQ Eats t u)
    apply (auto simp: fresh-Pair intro: Ex-I [where x=Eats t u])
    apply (metis Assume Mem-cong [OF - Refl, THEN Iff-MP-same] rotate2)
    done
  thus ?thesis
    by (rule Sigma-fm-Iff) (auto simp: assms supp-conv-fresh Eq-Eats-sf)
qed

lemma Subset-Mem-sf-lemma:
  size t + size u < n ⟹ Sigma-fm (t SUBS u) ∧ Sigma-fm (t IN u)
proof (induction n arbitrary: t u rule: less-induct)
  case (less n t u)
  show ?case
  proof
    show Sigma-fm (t SUBS u)
      proof (cases t rule: tm.exhaust)
        case Zero thus ?thesis
          by (auto intro: theorem-sf)
      next
        case (Var i) thus ?thesis using less.prems
          apply (cases u rule: tm.exhaust)
          apply (auto simp: Subset-Zero-sf Var-Subset-sf)
          apply (force simp: supp-conv-fresh less.IH
                       intro: Subset-Eats-sf Sigma-fm-Iff [OF Extensionality])
          done
      next
        case (Eats t1 t2) thus ?thesis using less.IH [OF - ijk] less.prems
         by (auto intro!: Sigma-fm-Iff [OF Eats-Subset-Iff] simp: supp-conv-fresh)
            (metis add.commute)
      qed
  next
    show Sigma-fm (t IN u)
      proof (cases u rule: tm.exhaust)
        case Zero show ?thesis
          by (rule Sigma-fm-Iff [where A=Fls]) (auto simp: supp-conv-fresh Zero)
      next
        case (Var i) show ?thesis
        proof (cases t rule: tm.exhaust)
          case Zero thus ?thesis using ‹u = Var i›
            by (auto intro: Zero-Mem-sf)
        next
          case (Var j)
```

```
        thus ?thesis using ‹u = Var i›
          by auto
      next
        case (Eats t1 t2) thus ?thesis using ‹u = Var i› less.prems
          by (force intro: Eats-Mem-sf Sigma-fm-Iff [OF Extensionality - -]
                   simp: supp-conv-fresh less.IH [THEN conjunct1])
      qed
    next
      case (Eats t1 t2) thus ?thesis using less.prems
       by (force intro: Sigma-fm-Iff [OF Mem-Eats-Iff] Sigma-fm-Iff [OF Exten-
sionality - -]
                 simp: supp-conv-fresh less.IH)
    qed
  qed
qed

lemma Subset-sf [iff]: Sigma-fm (t SUBS u)
  by (metis Subset-Mem-sf-lemma [OF lessI])

lemma Mem-sf [iff]: Sigma-fm (t IN u)
  by (metis Subset-Mem-sf-lemma [OF lessI])
```

The equality relation is a Sigma-Formula

```
lemma Equality-sf [iff]: Sigma-fm (t EQ u)
  by (auto intro: Sigma-fm-Iff [OF Extensionality] simp: supp-conv-fresh)
```

4.4 Universal Quantification Bounded by an Arbitrary Term

```
lemma All2-term-Iff: atom i ♯ t ⟹ atom j ♯ (i,t,A) ⟹
                {} ⊢ (All2 i t A) IFF Ex j (Var j EQ t AND All2 i (Var j) A)
apply auto
apply (rule Ex-I [where x=t], auto)
apply (rule Ex-I [where x=Var i])
apply (auto intro: ContraProve Mem-cong [THEN Iff-MP2-same])
done

lemma Sigma-fm-All2 [intro!]:
  assumes Sigma-fm A atom i ♯ t
    shows Sigma-fm (All2 i t A)
proof –
  obtain j::name where j: atom j ♯ (i,t,A)
    by (metis obtain-fresh)
  show Sigma-fm (All2 i t A)
    apply (rule Sigma-fm-Iff [OF All2-term-Iff [of i t j]])
    using assms j
    apply (auto simp: supp-conv-fresh Sigma-fm-All2-Var)
    done
```

qed

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

lemma *OrdP-sf* [*iff*]: *Sigma-fm* (*OrdP t*)
proof −
  obtain *z*::*name* **and** *y*::*name* **where** *atom z* ♯ *t atom y* ♯ (*t*, *z*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *OrdP.simps*)
qed

lemma *OrdNotEqP-sf* [*iff*]: *Sigma-fm* (*OrdNotEqP t u*)
  by (*auto simp*: *OrdNotEqP.simps*)

lemma *HDomain-Incl-sf* [*iff*]: *Sigma-fm* (*HDomain-Incl t u*)
proof −
  obtain *x*::*name* **and** *y*::*name* **and** *z*::*name*
    where *atom x* ♯ (*t*,*u*,*y*,*z*) *atom y* ♯ (*t*,*u*,*z*) *atom z* ♯ (*t*,*u*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by *auto*
qed

lemma *HFun-Sigma-Iff*:
  assumes *atom z* ♯ (*r*,*z′*,*x*,*y*,*x′*,*y′*) *atom z′* ♯ (*r*,*x*,*y*,*x′*,*y′*)
      atom x ♯ (*r*,*y*,*x′*,*y′*) *atom y* ♯ (*r*,*x′*,*y′*)
      atom x′ ♯ (*r*,*y′*) *atom y′* ♯ (*r*)
  shows
  {} ⊢*HFun-Sigma r IFF*
      All2 z r (*All2 z′ r* (*Ex x* (*Ex y* (*Ex x′* (*Ex y′*
        (*Var z EQ HPair* (*Var x*) (*Var y*) *AND Var z′ EQ HPair* (*Var x′*) (*Var*
y′)
          AND OrdP (*Var x*) *AND OrdP* (*Var x′*) *AND*
          ((*Var x NEQ Var x′*) *OR* (*Var y EQ Var y′*))))))))
  apply (*simp add*: *HFun-Sigma.simps* [*OF assms*])
  apply (*rule Iff-refl All-cong Imp-cong Ex-cong*)+
  apply (*rule Conj-cong* [*OF Iff-refl*])
  apply (*rule Conj-cong* [*OF Iff-refl*], *auto*)
  apply (*blast intro*: *Disj-I1 Neg-D OrdNotEqP-I*)
  apply (*blast intro*: *Disj-I2*)
  apply (*blast intro*: *OrdNotEqP-E rotate2*)
  done

lemma *HFun-Sigma-sf* [*iff*]: *Sigma-fm* (*HFun-Sigma t*)
proof −
  obtain *x*::*name* **and** *y*::*name* **and** *z*::*name* **and** *x′*::*name* **and** *y′*::*name* **and**

z'::*name*
  where *atoms*: *atom* $z \sharp (t,z',x,y,x',y')$ *atom* $z' \sharp (t,x,y,x',y')$
    atom $x \sharp (t,y,x',y')$ *atom* $y \sharp (t,x',y')$
    atom $x' \sharp (t,y')$ *atom* $y' \sharp (t)$
  by (*metis obtain-fresh*)
 show *?thesis*
  by (*auto intro*!: *Sigma-fm-Iff* [*OF HFun-Sigma-Iff* [*OF atoms*]] *simp*: *supp-conv-fresh atoms*)
qed

lemma *LstSeqP-sf* [*iff*]: *Sigma-fm* (*LstSeqP t u v*)
 by (*auto simp*: *LstSeqP.simps*)

4.6 A Key Result: Theorem 2.5

4.6.1 Sigma-Eats Formulas

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

equivariance *se-fm*

nominal-inductive *se-fm*
 avoids *ExI*: *i* | *All2I*: *i*
by (*simp-all add*: *fresh-star-def*)

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

lemma *subst-fm-in-se-fm*: *se-fm A* $\Longrightarrow$ *se-fm* (*A*(*k*::=*x*))
by (*nominal-induct avoiding*: *k x rule*: *se-fm.strong-induct*) (*auto*)

4.6.2 Preparation

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

lemma *obtain-const-tm*: **obtains** *t* **where** $[\![t]\!]e = x$ *ground t*
proof (*induct x rule*: *hf-induct*)
 case *0* **thus** *?case*
  by (*metis ground-aux-simps*(*1*) *eval-tm.simps*(*1*))
next
 case (*hinsert y x*) **thus** *?case*
  by (*metis ground-aux-simps*(*3*) *eval-tm.simps*(*3*))
qed

lemma *ex-eval-fm-iff-exists-tm*:
 eval-fm e (*Ex k A*) $\longleftrightarrow$ ($\exists$ *t*. *eval-fm e* (*A*(*k*::=*t*)) $\wedge$ *ground t*)

by (*auto simp*: *eval-subst-fm*) (*metis obtain-const-tm*)

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

lemma *ex-eval-fm-iff-exists-tm'*:
eval-fm e (*Ex k A*) $\longleftrightarrow$ ($\exists$ *t*. *eval-fm e* (*A*(*k*::=*t*)))
by (*auto simp*: *eval-subst-fm*) (*metis obtain-const-tm*)

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

nominal-function *elts* :: *tm* $\Rightarrow$ *tm set* **where**
elts Zero = {}
| *elts* (*Var k*) = {}
| *elts* (*Eats t u*) = *insert u* (*elts t*)
by (*auto simp*: *eqvt-def elts-graph-aux-def*) (*metis tm.exhaust*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma *eval-fm-All2-Eats*:
atom i $\sharp$ (*t*,*u*) $\Longrightarrow$
eval-fm e (*All2 i* (*Eats t u*) *A*) $\longleftrightarrow$ *eval-fm e* (*A*(*i*::=*u*)) $\wedge$ *eval-fm e* (*All2 i t A*)
by (*simp only*: *ex-eval-fm-iff-exists-tm' eval-fm.simps*) (*auto simp*: *eval-subst-fm*)

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

lemma *eval-fm-All2-Iff-elts*:
ground t $\Longrightarrow$ *eval-fm e* (*All2 i t A*) $\longleftrightarrow$ ($\forall$ *u* $\in$ *elts t*. *eval-fm e* (*A*(*i*::=*u*)))
apply (*induct t rule*: *tm.induct*)
apply *auto* [*2*]
apply (*simp add*: *eval-fm-All2-Eats del*: *eval-fm.simps*)
done

lemma *prove-elts-imp-prove-All2*:
ground t $\Longrightarrow$ ($\bigwedge$*u*. *u* $\in$ *elts t* $\Longrightarrow$ {} $\vdash$ *A*(*i*::=*u*)) $\Longrightarrow$ {} $\vdash$ *All2 i t A*
proof (*induct t rule*: *tm.induct*)
case *Zero* **thus** *?case*
by *auto*
next
case (*Var i*) **thus** *?case* — again: vacuously!
by *simp*
next
case (*Eats t u*)
hence *pt*: {} $\vdash$ *All2 i t A* **and** *pu*: {} $\vdash$ *A*(*i*::=*u*)
by *auto*
have {} $\vdash$ ((*Var i IN t*) *IMP A*)(*i* ::= *Var i*)
by (*rule All-D* [*OF pt*])
hence {} $\vdash$ ((*Var i IN t*) *IMP A*)
by *simp*
thus *?case* **using** *pu*

by (*auto intro*: *anti-deduction*) (*metis Iff-MP-same Var-Eq-subst-Iff thin1*)
qed

4.6.3 The base cases: ground atomic formulas

lemma *ground-prove*:
$[\![size\ t + size\ u < n;\ ground\ t;\ ground\ u]\!]$
$\Longrightarrow ([\![t]\!]e \le [\![u]\!]e \longrightarrow \{\} \vdash t\ SUBS\ u) \wedge ([\![t]\!]e \in [\![u]\!]e \longrightarrow \{\} \vdash t\ IN\ u)$
proof (*induction n arbitrary*: *t u rule*: *less-induct*)
case (*less n t u*)
show *?case*
proof
show $[\![t]\!]e \le [\![u]\!]e \longrightarrow \{\} \vdash t\ SUBS\ u$ **using** *less*
by (*cases t rule*: *tm.exhaust*) *auto*
next
{ fix *y t u*
have $[\![y < n;\ size\ t + size\ u < y;\ ground\ t;\ ground\ u;\ [\![t]\!]e = [\![u]\!]e]\!]$
$\Longrightarrow \{\} \vdash t\ EQ\ u$
by (*metis Equality-I less.IH add.commute order-refl*)
}
thus $[\![t]\!]e \in [\![u]\!]e \longrightarrow \{\} \vdash t\ IN\ u$ **using** *less.prems*
by (*cases u rule*: *tm.exhaust*) (*auto simp*: *Mem-Eats-I1 Mem-Eats-I2 less.IH*)
qed
qed

lemma
assumes *ground t ground u*
shows *ground-prove-SUBS*: $[\![t]\!]e \le [\![u]\!]e \Longrightarrow \{\} \vdash t\ SUBS\ u$
and *ground-prove-IN*: $[\![t]\!]e \in [\![u]\!]e \Longrightarrow \{\} \vdash t\ IN\ u$
and *ground-prove-EQ*: $[\![t]\!]e = [\![u]\!]e \Longrightarrow \{\} \vdash t\ EQ\ u$
by (*metis Equality-I assms ground-prove* [*OF lessI*] *order-refl*)+

lemma *ground-subst*:
ground-aux tm (*insert* (*atom i*) *S*) $\Longrightarrow$ *ground t* $\Longrightarrow$ *ground-aux* (*subst i t tm*) *S*
by (*induct tm rule*: *tm.induct*) (*auto simp*: *ground-aux-def*)

lemma *ground-subst-fm*:
ground-fm-aux A (*insert* (*atom i*) *S*) $\Longrightarrow$ *ground t* $\Longrightarrow$ *ground-fm-aux* ($A(i{::=}t)$) *S*
apply (*nominal-induct A avoiding*: *i arbitrary*: *S rule*: *fm.strong-induct*)
apply (*auto simp*: *ground-subst Set.insert-commute*)
done

lemma *elts-imp-ground*: $u \in elts\ t \Longrightarrow ground\text{-}aux\ t\ S \Longrightarrow ground\text{-}aux\ u\ S$
by (*induct t rule*: *tm.induct*) *auto*

lemma *ground-se-fm-induction*:
$ground\text{-}fm\ \alpha \Longrightarrow size\ \alpha < n \Longrightarrow se\text{-}fm\ \alpha \Longrightarrow eval\text{-}fm\ e\ \alpha \Longrightarrow \{\} \vdash \alpha$
proof (*induction n arbitrary*: α *rule*: *less-induct*)

```
  case (less n α)
  show ?case using ‹se-fm α›
  proof (cases rule: se-fm.cases)
    case (MemI t u) thus {} ⊢ α using less
      by (auto intro: ground-prove-IN)
  next
    case (DisjI A B) thus {} ⊢ α using less
      by (auto intro: Disj-I1 Disj-I2)
  next
    case (ConjI A B) thus {} ⊢ α using less
      by auto
  next
    case (ExI A i)
    thus {} ⊢ α using less.prems
      apply (auto simp: ex-eval-fm-iff-exists-tm simp del: better-ex-eval-fm)
      apply (auto intro!: Ex-I less.IH subst-fm-in-se-fm ground-subst-fm)
      done
  next
    case (All2I A i t)
    hence t: ground t using less.prems
      by (auto simp: ground-aux-def fresh-def)
    hence (∀ u∈elts t. eval-fm e (A(i::=u)))
      by (metis All2I(1) t eval-fm-All2-Iff-elts less(5))
    thus {} ⊢ α using less.prems All2I t
      apply (auto del: Neg-I intro!: prove-elts-imp-prove-All2 less.IH)
      apply (auto intro: subst-fm-in-se-fm ground-subst-fm elts-imp-ground)
      done
  qed
qed

lemma ss-imp-se-fm: ss-fm A ⟹ se-fm A
  by (erule ss-fm.induct) auto

lemma se-fm-imp-thm: ⟦se-fm A; ground-fm A; eval-fm e A⟧ ⟹ {} ⊢ A
  by (metis ground-se-fm-induction lessI)
```

Theorem 2.5

```
theorem Sigma-fm-imp-thm: ⟦Sigma-fm A; ground-fm A; eval-fm e0 A⟧ ⟹ {} ⊢
A
 by (metis Iff-MP2-same ss-imp-se-fm empty-iff Sigma-fm-def eval-fm-Iff ground-fm-aux-def

          hfthm-sound se-fm-imp-thm subset-empty)

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

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

lemma *VarP-eqvt* [*eqvt*]: (*p* · *VarP x*) = *VarP* (*p* · *x*)
 by (*simp add*: *VarP-def*)

lemma *VarP-fresh-iff* [*simp*]: *a* ♯ *VarP x* ⟷ *a* ♯ *x*
 by (*simp add*: *VarP-def*)

lemma *eval-fm-VarP* [*simp*]: *eval-fm e* (*VarP x*) ⟷ *is-Var* ⟦*x*⟧*e*
 by (*simp add*: *VarP-def is-Var-def*)

lemma *VarP-sf* [*iff*]: *Sigma-fm* (*VarP x*)
 by (*auto simp*: *VarP-def*)

lemma *VarP-subst* [*simp*]: (*VarP x*)(*i*::=*t*) = *VarP* (*subst i t x*)
 by (*simp add*: *VarP-def*)

lemma *VarP-cong*: $H \vdash x\ EQ\ x' \Longrightarrow H \vdash VarP\ x\ IFF\ VarP\ x'$
  by (*rule P1-cong*) *auto*

lemma *VarP-HPairE* [*intro*!]: *insert* (*VarP* (*HPair x y*)) $H \vdash A$
  by (*auto simp*: *VarP-def*)

lemma *is-Var-succ-iff* [*simp*]: *is-Var* (*succ x*) = *Ord x*
  by (*metis Ord-succ-iff is-Var-def succ-iff zero-in-Ord*)

lemma *is-Var-q-Var* [*iff*]: *is-Var* (*q-Var i*)
  by (*simp add*: *q-Var-def*)

definition *decode-Var* :: *hf* ⇒ *name*
  where *decode-Var x* ≡ *name-of-nat* (*nat-of-ord* (*pred x*))

lemma *decode-Var-q-Var* [*simp*]: *decode-Var* (*q-Var i*) = *i*
  by (*simp add*: *decode-Var-def q-Var-def*)

lemma *is-Var-imp-decode-Var*: *is-Var x* ⟹ *x* = ⟦⌈*Var* (*decode-Var x*)⌉⟧ *e*
  by (*simp add*: *is-Var-def quot-Var decode-Var-def*) (*metis hempty-iff succ-pred*)

lemma *is-Var-iff*: *is-Var v* ⟷ *v* = *succ* (*ord-of* (*nat-of-name* (*decode-Var v*)))
 by (*metis eval-tm-ORD-OF eval-tm-SUCC is-Var-imp-decode-Var quot-Var is-Var-succ-iff Ord-ord-of*)

lemma *decode-Var-inject* [*simp*]: *is-Var v* ⟹ *is-Var v'* ⟹ *decode-Var v* = *decode-Var v'* ⟷ *v*=*v'*
  by (*metis is-Var-iff*)

5.1.2 De Bruijn Indexes

definition *is-Ind* :: *hf* ⇒ *bool*
  where *is-Ind x* ≡ (∃ *m*. *Ord m* ∧ *x* = ⟨*htuple 6*, *m*⟩)

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*))
  by (*auto simp*: *eqvt-def IndP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma
  shows *IndP-fresh-iff* [*simp*]: *a* ♯ *IndP x* ⟷ *a* ♯ *x*   (**is** *?thesis1*)
    and *eval-fm-IndP* [*simp*]: *eval-fm e* (*IndP x*) ⟷ *is-Ind* ⟦*x*⟧*e* (**is** *?thesis2*)
    and *IndP-sf* [*iff*]: *Sigma-fm* (*IndP x*)   (**is** *?thsf*)

and *OrdP-IndP-Q-Ind*: $\{OrdP\ x\} \vdash IndP\ (Q\text{-}Ind\ x)$ (**is** *?thqind*)
proof −
obtain *m::name* **where** *atom m* ♯ *x*
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thsf ?thqind*
by (*auto simp*: *is-Ind-def intro*: *Ex-I* [**where** *x=x*])
qed

lemma *IndP-Q-Ind*: $H \vdash OrdP\ x \Longrightarrow H \vdash IndP\ (Q\text{-}Ind\ x)$
by (*rule cut1* [*OF OrdP-IndP-Q-Ind*])

lemma *subst-fm-IndP* [*simp*]: (*IndP t*)(*i::=x*) = *IndP* (*subst i x t*)
proof −
obtain *m::name* **where** *atom m* ♯ (*i*,*t*,*x*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *IndP.simps* [*of m*])
qed

lemma *IndP-cong*: $H \vdash x\ EQ\ x' \Longrightarrow H \vdash IndP\ x\ IFF\ IndP\ x'$
by (*rule P1-cong*) *auto*

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

lemma *is-Ind-pair-iff* [*simp*]: $is\text{-}Ind\ \langle x,\ y\rangle \longleftrightarrow x = htuple\ 6 \wedge Ord\ y$
by (*auto simp*: *is-Ind-def*)

5.1.3 Various syntactic lemmas

lemma *eval-Var-q*: $[\![\lceil Var\ i\rceil]\!]\ e = q\text{-}Var\ i$
by (*simp add*: *quot-tm-def q-Var-def*)

lemma *is-Var-eval-Var* [*simp*]: $is\text{-}Var\ [\![\lceil Var\ i\rceil]\!]e$
by (*metis decode-Var-q-Var is-Var-imp-decode-Var is-Var-q-Var*)

5.2 The predicate *SeqCTermP*, for Terms and Constants

definition *SeqCTerm* :: *bool* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SeqCTerm vf s k t* ≡ *BuildSeq* ($\lambda u.\ u=0 \vee vf \wedge is\text{-}Var\ u$) ($\lambda u\ v\ w.\ u = q\text{-}Eats\ v\ w$) *s k t*

nominal-function *SeqCTermP* :: *bool* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom l* ♯ (*s*,*k*,*sl*,*m*,*n*,*sm*,*sn*); *atom sl* ♯ (*s*,*m*,*n*,*sm*,*sn*);
atom m ♯ (*s*,*n*,*sm*,*sn*); *atom n* ♯ (*s*,*sm*,*sn*);
atom sm ♯ (*s*,*sn*); *atom sn* ♯ (*s*)⟧ ⟹
SeqCTermP vf s k t =

LstSeqP s k t AND
All2 l (SUCC k) (Ex sl (HPair (Var l) (Var sl) IN s AND
(Var sl EQ Zero OR (if vf then VarP (Var sl) else Fls) OR
Ex m (Ex n (Ex sm (Ex sn (Var m IN Var l AND Var n IN Var l AND
HPair (Var m) (Var sm) IN s AND HPair (Var n) (Var sn) IN s AND
Var sl EQ Q-Eats (Var sm) (Var sn))))))))
by (*auto simp*: *eqvt-def SeqCTermP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *SeqCTermP-fresh-iff* [*simp*]:
$a \sharp$ *SeqCTermP vf s k t* $\longleftrightarrow a \sharp s \wedge a \sharp k \wedge a \sharp t$ (**is** *?thesis1*)
and *eval-fm-SeqCTermP* [*simp*]:
eval-fm e (*SeqCTermP vf s k t*) $\longleftrightarrow$ *SeqCTerm vf* $[\![s]\!]e$ $[\![k]\!]e$ $[\![t]\!]e$ (**is** *?thesis2*)
and *SeqCTermP-sf* [*iff*]:
Sigma-fm (*SeqCTermP vf s k t*) (**is** *?thsf*)
and *SeqCTermP-imp-LstSeqP*:
{ *SeqCTermP vf s k t* } $\vdash$ *LstSeqP s k t* (**is** *?thlstseq*)
and *SeqCTermP-imp-OrdP* [*simp*]:
{ *SeqCTermP vf s k t* } $\vdash$ *OrdP k* (**is** *?thord*)
proof −
obtain *l*::*name* **and** *sl*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name*
where *atoms*: *atom l* $\sharp$ (*s*,*k*,*sl*,*m*,*n*,*sm*,*sn*) *atom sl* $\sharp$ (*s*,*m*,*n*,*sm*,*sn*)
atom m $\sharp$ (*s*,*n*,*sm*,*sn*) *atom n* $\sharp$ (*s*,*sm*,*sn*)
atom sm $\sharp$ (*s*,*sn*) *atom sn* $\sharp$ (*s*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thsf ?thlstseq ?thord*
by (*auto simp*: *LstSeqP.simps*)
show *?thesis2* **using** *atoms*
by (*simp cong*: *conj-cong add*: *LstSeq-imp-Ord SeqCTerm-def BuildSeq-def Builds-def*
HBall-def HBex-def q-Eats-def Fls-def
Seq-iff-app [*of* $[\![s]\!]e$, *OF LstSeq-imp-Seq-succ*]
Ord-trans [*of - - succ* $[\![k]\!]e$])
qed

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 −
obtain *l*::*name* **and** *sl*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name*
where *atom l* $\sharp$ (*j*,*w*,*s*,*k*,*sl*,*m*,*n*,*sm*,*sn*) *atom sl* $\sharp$ (*j*,*w*,*s*,*m*,*n*,*sm*,*sn*)

atom m ♯ (*j*,*w*,*s*,*n*,*sm*,*sn*) *atom n* ♯ (*j*,*w*,*s*,*sm*,*sn*)
atom sm ♯ (*j*,*w*,*s*,*sn*) *atom sn* ♯ (*j*,*w*,*s*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*force simp add*: *SeqCTermP*.*simps* [*of l - - sl m n sm sn*])
qed

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

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

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

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

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

lemma *SeqConst-imp-SeqTerm*: *SeqConst s k x* ⟹ *SeqTerm s k x*
by (*auto simp*: *SeqCTerm-def intro*: *BuildSeq-mono*)

lemma *SeqConstP-imp-SeqTermP*: {*SeqConstP s k t*} ⊢ *SeqTermP s k t*
proof −
obtain *l*::*name* **and** *sl*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name*
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*)
by (*metis obtain-fresh*)
thus *?thesis*
apply (*auto simp*: *SeqCTermP*.*simps* [*of l s k sl m n sm sn*])
apply (*rule Ex-I* [**where** *x*=*Var l*], *auto*)
apply (*rule Ex-I* [**where** *x* = *Var sl*], *force intro*: *Disj-I1*)
apply (*rule Ex-I* [**where** *x* = *Var sl*], *simp*)
apply (*rule Conj-I*, *blast*)
apply (*rule Disj-I2*)+
apply (*rule Ex-I* [**where** *x* = *Var m*], *simp*)
apply (*rule Ex-I* [**where** *x* = *Var n*], *simp*)
apply (*rule Ex-I* [**where** *x* = *Var sm*], *simp*)
apply (*rule Ex-I* [**where** *x* = *Var sn*], *auto*)
done
qed

5.3 The predicates *TermP* and *ConstP*

5.3.1 Definition

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

nominal-function *CTermP* :: *bool* ⇒ *tm* ⇒ *fm*
  where ⟦*atom k* ♯ (*s*,*t*); *atom s* ♯ *t*⟧ ⟹
    CTermP vf t = *Ex s* (*Ex k* (*SeqCTermP vf* (*Var s*) (*Var k*) *t*))
 by (*auto simp*: *eqvt-def CTermP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma
  shows *CTermP-fresh-iff* [*simp*]: *a* ♯ *CTermP vf t* ⟷ *a* ♯ *t*   (**is** *?thesis1*)
    and *eval-fm-CTermP* [*simp*] :*eval-fm e* (*CTermP vf t*) ⟷ *CTerm vf* ⟦*t*⟧*e* (**is** *?thesis2*)
    and *CTermP-sf* [*iff*]: *Sigma-fm* (*CTermP vf t*)   (**is** *?thsf*)
proof −
  obtain *k*::*name* **and** *s*::*name* **where** *atom k* ♯ (*s*,*t*) *atom s* ♯ *t*
    by (*metis obtain-fresh*)
  thus *?thesis1 ?thesis2 ?thsf*
    by (*auto simp*: *CTerm-def*)
qed

lemma *CTermP-subst* [*simp*]: (*CTermP vf i*)(*j*::=*w*) = *CTermP vf* (*subst j w i*)
proof −
  obtain *k*::*name* **and** *s*::*name* **where** *atom k* ♯ (*s*,*i*,*j*,*w*) *atom s* ♯ (*i*,*j*,*w*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*simp add*: *CTermP.simps* [*of k s*])
qed

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

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

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

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

5.3.2 Correctness: It Corresponds to Quotations of Real Terms

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

by (*induct rule*: *wf-dbtm.induct*)
(*auto simp*: *CTerm-def SeqCTerm-def q-Eats-def intro*: *BuildSeq-combine BuildSeq-exI*)

corollary *Term-quot-tm* [*iff*]: **fixes** t :: *tm* **shows** *Term* $[\![\lceil t \rceil]\!]e$
by (*metis quot-tm-def wf-Term-quot-dbtm wf-dbtm-trans-tm*)

lemma *SeqCTerm-imp-wf-dbtm*:
assumes *SeqCTerm vf s k x*
shows $\exists$ *t*::*dbtm*. *wf-dbtm t* $\wedge$ $x = [\![$*quot-dbtm t*$]\!]e$
using *assms* [*unfolded SeqCTerm-def*]
proof (*induct x rule*: *BuildSeq-induct*)
case (*B x*) **thus** *?case*
by *auto* (*metis ORD-OF.simps*(*2*) *Var quot-dbtm.simps*(*2*) *is-Var-imp-decode-Var quot-Var*)
next
case (*C x y z*)
then obtain *tm1*::*dbtm* **and** *tm2*::*dbtm*
where *wf-dbtm tm1* $y = [\![$*quot-dbtm tm1*$]\!]e$
wf-dbtm tm2 $z = [\![$*quot-dbtm tm2*$]\!]e$
by *blast*
thus *?case*
by (*auto simp*: *wf-dbtm.intros C q-Eats-def intro*!: *exI* [*of - DBEats tm1 tm2*])
qed

corollary *Term-imp-wf-dbtm*:
assumes *Term x* **obtains** *t* **where** *wf-dbtm t* $x = [\![$*quot-dbtm t*$]\!]e$
by (*metis assms SeqCTerm-imp-wf-dbtm CTerm-def*)

corollary *Term-imp-is-tm*: **assumes** *Term x* **obtains** *t*::*tm* **where** $x = [\![\lceil t \rceil]\!]\ e$
by (*metis assms Term-imp-wf-dbtm quot-tm-def wf-dbtm-imp-is-tm*)

lemma *Term-Var*: *Term* (*q-Var i*)
using *wf-Term-quot-dbtm* [*of DBVar i*]
by (*metis Term-quot-tm is-Var-imp-decode-Var is-Var-q-Var*)

lemma *Term-Eats*: **assumes** *x*: *Term x* **and** *y*: *Term y* **shows** *Term* (*q-Eats x y*)
proof −
obtain *t u* **where** $x = [\![$*quot-dbtm t*$]\!]e$ $y = [\![$*quot-dbtm u*$]\!]e$
by (*metis Term-imp-wf-dbtm x y*)
thus *?thesis* **using** *wf-Term-quot-dbtm* [*of DBEats t u*] *x y*
by (*auto simp*: *q-defs*) (*metis Eats Term-imp-wf-dbtm quot-dbtm-inject-lemma*)
qed

5.3.3 Correctness properties for constants

lemma *Const-imp-Term*: *Const x* $\Longrightarrow$ *Term x*
by (*metis SeqConst-imp-SeqTerm CTerm-def*)

lemma *Const-0*: *Const 0*
by (*force simp add*: *CTerm-def SeqCTerm-def intro*: *BuildSeq-exI*)

lemma *ConstP-imp-TermP*: $\{ConstP\ t\} \vdash TermP\ t$
proof −
obtain *k*::*name* **and** *s*::*name* **where** *atom k* ♯ (*s*,*t*) *atom s* ♯ *t*
by (*metis obtain-fresh*)
thus *?thesis*
apply *auto*
apply (*rule Ex-I* [**where** *x* = *Var s*], *simp*)
apply (*rule Ex-I* [**where** *x* = *Var k*], *auto intro*: *SeqConstP-imp-SeqTermP* [*THEN cut1*])
done
qed

5.4 Abstraction over terms

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

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

5.4.1 Defining the syntax: quantified body

nominal-function *SeqStTermP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom l* ♯ (*s*,*k*,*v*,*i*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*);
atom sl ♯ (*s*,*v*,*i*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*); *atom sl′* ♯ (*s*,*v*,*i*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*);
atom m ♯ (*s*,*n*,*sm*,*sm′*,*sn*,*sn′*); *atom n* ♯ (*s*,*sm*,*sm′*,*sn*,*sn′*);
atom sm ♯ (*s*,*sm′*,*sn*,*sn′*); *atom sm′* ♯ (*s*,*sn*,*sn′*);
atom sn ♯ (*s*,*sn′*); *atom sn′* ♯ *s*⟧ ⟹
SeqStTermP v i t u s k =
VarP v AND LstSeqP s k (*HPair t u*) *AND*
All2 l (*SUCC k*) (*Ex sl* (*Ex sl′* (*HPair* (*Var l*) (*HPair* (*Var sl*) (*Var sl′*)) *IN s AND*
(((*Var sl EQ v AND Var sl′ EQ i*) *OR*
((*IndP* (*Var sl*) *OR Var sl NEQ v*) *AND Var sl′ EQ Var sl*)) *OR*
Ex m (*Ex n* (*Ex sm* (*Ex sm′* (*Ex sn* (*Ex sn′* (*Var m IN Var l AND Var n IN Var l AND*
HPair (*Var m*) (*HPair* (*Var sm*) (*Var sm′*)) *IN s AND*
HPair (*Var n*) (*HPair* (*Var sn*) (*Var sn′*)) *IN s AND*
Var sl EQ Q-Eats (*Var sm*) (*Var sn*) *AND*
Var sl′ EQ Q-Eats (*Var sm′*) (*Var sn′*))))))))))))
by (*auto simp*: *eqvt-def SeqStTermP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)

by *lexicographic-order*

lemma
shows *SeqStTermP-fresh-iff* [*simp*]:
$a \sharp$ *SeqStTermP v i t u s k* $\longleftrightarrow a \sharp v \wedge a \sharp i \wedge a \sharp t \wedge a \sharp u \wedge a \sharp s \wedge a \sharp k$ (**is** *?thesis1*)
and *eval-fm-SeqStTermP* [*simp*]:
eval-fm e (*SeqStTermP v i t u s k*) $\longleftrightarrow$ *SeqStTerm* $[\![v]\!]e$ $[\![i]\!]e$ $[\![t]\!]e$ $[\![u]\!]e$ $[\![s]\!]e$ $[\![k]\!]e$ (**is** *?thesis2*)
and *SeqStTermP-sf* [*iff*]:
Sigma-fm (*SeqStTermP v i t u s k*) (**is** *?thsf*)
and *SeqStTermP-imp-OrdP*:
{ *SeqStTermP v i t u s k* } $\vdash$ *OrdP k* (**is** *?thord*)
and *SeqStTermP-imp-VarP*:
{ *SeqStTermP v i t u s k* } $\vdash$ *VarP v* (**is** *?thvar*)
and *SeqStTermP-imp-LstSeqP*:
{ *SeqStTermP v i t u s k* } $\vdash$ *LstSeqP s k* (*HPair t u*) (**is** *?thlstseq*)
proof −
obtain *l*::*name* **and** *sl*::*name* **and** *sl'*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sm'*::*name* **and** *sn*::*name* **and** *sn'*::*name*
where *atoms*:
atom l $\sharp$ (*s*,*k*,*v*,*i*,*sl*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl $\sharp$ (*s*,*v*,*i*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*) *atom sl'* $\sharp$ (*s*,*v*,*i*,*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*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thsf ?thord ?thvar ?thlstseq*
by (*auto intro*: *LstSeqP-OrdP*)
show *?thesis2* **using** *atoms*
apply (*simp add*: *LstSeq-imp-Ord SeqStTerm-def ex-disj-distrib*
BuildSeq2-def BuildSeq-def Builds-def
HBall-def q-Eats-def q-Ind-def is-Var-def
Seq-iff-app [*of* $[\![s]\!]e$, *OF LstSeq-imp-Seq-succ*]
Ord-trans [*of* - - *succ* $[\![k]\!]e$]
cong: *conj-cong*)
apply (*rule conj-cong refl all-cong*)+
apply *auto*
apply (*metis Not-Ord-hpair is-Ind-def*)
done
qed

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 −
obtain *l*::*name* **and** *sl*::*name* **and** *sl'*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sm'*::*name* **and** *sn*::*name* **and** *sn'*::*name*

where *atom l* ♯ (*s*,*k*,*v*,*i*,*w*,*j*,*sl*,*sl*′,*m*,*n*,*sm*,*sm*′,*sn*,*sn*′)
atom sl ♯ (*s*,*v*,*i*,*w*,*j*,*sl*′,*m*,*n*,*sm*,*sm*′,*sn*,*sn*′)
atom sl′ ♯ (*s*,*v*,*i*,*w*,*j*,*m*,*n*,*sm*,*sm*′,*sn*,*sn*′)
atom m ♯ (*s*,*w*,*j*,*n*,*sm*,*sm*′,*sn*,*sn*′) *atom n* ♯ (*s*,*w*,*j*,*sm*,*sm*′,*sn*,*sn*′)
atom sm ♯ (*s*,*w*,*j*,*sm*′,*sn*,*sn*′) *atom sm*′ ♯ (*s*,*w*,*j*,*sn*,*sn*′)
atom sn ♯ (*s*,*w*,*j*,*sn*′) *atom sn*′ ♯ (*s*,*w*,*j*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*force simp add*: *SeqStTermP.simps* [*of l - - - - sl sl*′ *m n sm sm*′ *sn sn*′])
qed

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*′
by (*rule P4-cong* [**where** *tms*=[*v*,*i*]]) (*auto simp*: *fresh-Cons*)

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

5.4.2 Defining the syntax: main predicate

nominal-function *AbstTermP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom s* ♯ (*v*,*i*,*t*,*u*,*k*); *atom k* ♯ (*v*,*i*,*t*,*u*)⟧ ⟹
AbstTermP v i t u =
OrdP i AND Ex s (*Ex k* (*SeqStTermP v* (*Q-Ind i*) *t u* (*Var s*) (*Var k*)))
by (*auto simp*: *eqvt-def AbstTermP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *AbstTermP-fresh-iff* [*simp*]:
a ♯ *AbstTermP v i t u* ⟷ *a* ♯ *v* ∧ *a* ♯ *i* ∧ *a* ♯ *t* ∧ *a* ♯ *u* (**is** *?thesis1*)
and *eval-fm-AbstTermP* [*simp*]:
eval-fm e (*AbstTermP v i t u*) ⟷ *AbstTerm* ⟦*v*⟧*e* ⟦*i*⟧*e* ⟦*t*⟧*e* ⟦*u*⟧*e* (**is** *?thesis2*)
and *AbstTermP-sf* [*iff*]:
Sigma-fm (*AbstTermP v i t u*) (**is** *?thsf*)
proof −
obtain *s*::*name* **and** *k*::*name* **where** *atom s* ♯ (*v*,*i*,*t*,*u*,*k*) *atom k* ♯ (*v*,*i*,*t*,*u*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thsf*
by (*auto simp*: *AbstTerm-def q-defs*)
qed

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 −
obtain *s*::*name* **and** *k*::*name* **where** *atom s* ♯ (*v*,*i*,*t*,*u*,*w*,*j*,*k*) *atom k* ♯ (*v*,*i*,*t*,*u*,*w*,*j*)
by (*metis obtain-fresh*)

```
  thus ?thesis
    by (simp add: AbstTermP.simps [of s - - - - k])
qed

declare AbstTermP.simps [simp del]
```

5.4.3 Correctness: It Coincides with Abstraction over real terms

```
lemma not-is-Var-is-Ind: is-Var v ⟹ ¬ is-Ind v
  by (auto simp: is-Var-def is-Ind-def)

lemma AbstTerm-imp-abst-dbtm:
  assumes AbstTerm v i x x'
  shows ∃t. x = ⟦quot-dbtm t⟧e ∧
            x' = ⟦quot-dbtm (abst-dbtm (decode-Var v) (nat-of-ord i) t)⟧e
proof -
  obtain s k where v: is-Var v and i: Ord i and sk: SeqStTerm v (q-Ind i) x x'
s k
    using assms
    by (auto simp: AbstTerm-def SeqStTerm-def)
  from sk [unfolded SeqStTerm-def, THEN conjunct2]
  show ?thesis
  proof (induct x x' rule: BuildSeq2-induct)
    case (B x x') thus ?case using v i
      apply (auto simp: not-is-Var-is-Ind)
      apply (rule-tac [1] x=DBInd (nat-of-ord (hsnd x)) in exI)
      apply (rule-tac [2] x=DBVar (decode-Var v) in exI)
      apply (case-tac [3] is-Var x)
      apply (rule-tac [3] x=DBVar (decode-Var x) in exI)
      apply (rule-tac [4] x=DBZero in exI)
      apply (auto simp: is-Ind-def q-Ind-def is-Var-iff [symmetric])
      apply (metis hmem-0-Ord is-Var-def)
      done
  next
    case (C x x' y y' z z')
    then obtain tm1 and tm2
      where y = ⟦quot-dbtm tm1⟧e
            y' = ⟦quot-dbtm (abst-dbtm (decode-Var v) (nat-of-ord i) tm1)⟧e
            z = ⟦quot-dbtm tm2⟧e
            z' = ⟦quot-dbtm (abst-dbtm (decode-Var v) (nat-of-ord i) tm2)⟧e
    by blast
  thus ?case
    by (auto simp: wf-dbtm.intros C q-Eats-def intro!: exI [where x=DBEats tm1
tm2])
  qed
qed

lemma AbstTerm-abst-dbtm:
```

AbstTerm (*q-Var i*) (*ord-of n*) $[\![$*quot-dbtm t*$]\!]$*e*
$[\![$*quot-dbtm* (*abst-dbtm i n t*)$]\!]$*e*
by (*induct t rule*: *dbtm.induct*)
(*auto simp*: *AbstTerm-def SeqStTerm-def q-defs intro*: *BuildSeq2-exI BuildSeq2-combine*)

5.5 Substitution over terms

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

5.5.1 Defining the syntax

nominal-function *SubstTermP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where $[\![$*atom s* ♯ (*v*,*i*,*t*,*u*,*k*); *atom k* ♯ (*v*,*i*,*t*,*u*)$]\!]$ ⟹
SubstTermP v i t u = *TermP i AND Ex s* (*Ex k* (*SeqStTermP v i t u* (*Var s*) (*Var k*)))
by (*auto simp*: *eqvt-def SubstTermP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *SubstTermP-fresh-iff* [*simp*]:
a ♯ *SubstTermP v i t u* ⟷ *a* ♯ *v* ∧ *a* ♯ *i* ∧ *a* ♯ *t* ∧ *a* ♯ *u* (**is** *?thesis1*)
and *eval-fm-SubstTermP* [*simp*]:
eval-fm e (*SubstTermP v i t u*) ⟷ *SubstTerm* $[\![$*v*$]\!]$*e* $[\![$*i*$]\!]$*e* $[\![$*t*$]\!]$*e* $[\![$*u*$]\!]$*e* (**is** *?thesis2*)
and *SubstTermP-sf* [*iff*]:
Sigma-fm (*SubstTermP v i t u*) (**is** *?thsf*)
and *SubstTermP-imp-TermP*:
{ *SubstTermP v i t u* } ⊢ *TermP i* (**is** *?thterm*)
and *SubstTermP-imp-VarP*:
{ *SubstTermP v i t u* } ⊢ *VarP v* (**is** *?thvar*)
proof −
obtain *s*::*name* **and** *k*::*name* **where** *atom s* ♯ (*v*,*i*,*t*,*u*,*k*) *atom k* ♯ (*v*,*i*,*t*,*u*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thsf ?thterm ?thvar*
by (*auto simp*: *SubstTerm-def intro*: *SeqStTermP-imp-VarP thin2*)
qed

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 −
obtain *s*::*name* **and** *k*::*name*
where *atom s* ♯ (*v*,*i*,*t*,*u*,*w*,*j*,*k*) *atom k* ♯ (*v*,*i*,*t*,*u*,*w*,*j*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*simp add*: *SubstTermP.simps* [*of s - - - - k*])

qed

lemma *SubstTermP-cong*:
  $[\![H \vdash v\ EQ\ v'; H \vdash i\ EQ\ i'; H \vdash t\ EQ\ t'; H \vdash u\ EQ\ u']\!]$
  $\Longrightarrow H \vdash SubstTermP\ v\ i\ t\ u\ IFF\ SubstTermP\ v'\ i'\ t'\ u'$
  by (*rule P4-cong*) *auto*

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

lemma *SubstTerm-imp-subst-dbtm*:
  assumes *SubstTerm v* $[\![$*quot-dbtm u*$]\!]$*e x x′*
  shows $\exists\, t.\ x = [\![$*quot-dbtm t*$]\!]$*e* $\wedge$
    $x' = [\![$*quot-dbtm* (*subst-dbtm u* (*decode-Var v*) *t*)$]\!]$*e*
proof −
  obtain *s k* **where** *v*: *is-Var v* **and** *u*: *Term* $[\![$*quot-dbtm u*$]\!]$*e*
    and *sk*: *SeqStTerm v* $[\![$*quot-dbtm u*$]\!]$*e x x′ s k*
   using *assms* [*unfolded SubstTerm-def*]
   by (*auto simp*: *SeqStTerm-def*)
  from *sk* [*unfolded SeqStTerm-def*, *THEN conjunct2*]
  show *?thesis*
  proof (*induct x x′ rule*: *BuildSeq2-induct*)
   case (*B x x′*) **thus** *?case* **using** *v*
    apply (*auto simp*: *not-is-Var-is-Ind*)
    apply (*rule-tac* [*1*] *x=DBInd* (*nat-of-ord* (*hsnd x*)) **in** *exI*)
    apply (*rule-tac* [*2*] *x=DBVar* (*decode-Var v*) **in** *exI*)
    apply (*case-tac* [*3*] *is-Var x*)
    apply (*rule-tac* [*3*] *x=DBVar* (*decode-Var x*) **in** *exI*)
    apply (*rule-tac* [*4*] *x=DBZero* **in** *exI*)
    apply (*auto simp*: *is-Ind-def q-Ind-def is-Var-iff* [*symmetric*])
    apply (*metis hmem-0-Ord is-Var-def*)
    done
  next
   case (*C x x′ y y′ z z′*)
   then obtain *tm1* **and** *tm2*
    where $y = [\![$*quot-dbtm tm1*$]\!]$*e*
     $y' = [\![$*quot-dbtm* (*subst-dbtm u* (*decode-Var v*) *tm1*)$]\!]$*e*
     $z = [\![$*quot-dbtm tm2*$]\!]$*e*
     $z' = [\![$*quot-dbtm* (*subst-dbtm u* (*decode-Var v*) *tm2*)$]\!]$*e*
   by *blast*
  thus *?case*
   by (*auto simp*: *wf-dbtm.intros C q-Eats-def intro*!: *exI* [**where** *x=DBEats tm1 tm2*])
  qed
qed

corollary *SubstTerm-imp-subst-dbtm′*:
  assumes *SubstTerm v y x x′*
  obtains *t*::*dbtm* **and** *u*::*dbtm*
  where $y = [\![$*quot-dbtm u*$]\!]$*e*

$x = [\![quot\text{-}dbtm\ t]\!]e$
$x' = [\![quot\text{-}dbtm\ (subst\text{-}dbtm\ u\ (decode\text{-}Var\ v)\ t)]\!]e$
by (*metis SubstTerm-def SubstTerm-imp-subst-dbtm Term-imp-is-tm assms quot-tm-def*)

lemma *SubstTerm-subst-dbtm*:
assumes *Term* $[\![quot\text{-}dbtm\ u]\!]e$
shows *SubstTerm* (*q-Var v*) $[\![quot\text{-}dbtm\ u]\!]e$ $[\![quot\text{-}dbtm\ t]\!]e$ $[\![quot\text{-}dbtm\ (subst\text{-}dbtm\ u\ v\ t)]\!]e$
by (*induct t rule*: *dbtm.induct*)
(*auto simp*: *assms SubstTerm-def SeqStTerm-def q-defs intro*: *BuildSeq2-exI BuildSeq2-combine*)

5.6 Abstraction over formulas

5.6.1 The predicate *AbstAtomicP*

definition *AbstAtomic* :: $hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$
where *AbstAtomic v i y y'* $\equiv$
$(\exists\, t\ u\ t'\ u'.\ AbstTerm\ v\ i\ t\ t' \wedge AbstTerm\ v\ i\ u\ u' \wedge$
$((y = q\text{-}Eq\ t\ u \wedge y' = q\text{-}Eq\ t'\ u') \vee (y = q\text{-}Mem\ t\ u \wedge y' = q\text{-}Mem\ t'\ u')))$

nominal-function *AbstAtomicP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $[\![atom\ t \sharp (v,i,y,y',t',u,u');\ atom\ t' \sharp (v,i,y,y',u,u');$
$atom\ u \sharp (v,i,y,y',u');\ atom\ u' \sharp (v,i,y,y')]\!] \Longrightarrow$
AbstAtomicP v i y y' =
Ex t (*Ex u* (*Ex t'* (*Ex u'*
(*AbstTermP v i* (*Var t*) (*Var t'*) *AND AbstTermP v i* (*Var u*) (*Var u'*) *AND*
((*y EQ Q-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'*)))))))
by (*auto simp*: *eqvt-def AbstAtomicP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *AbstAtomicP-fresh-iff* [*simp*]:
$a \sharp AbstAtomicP\ v\ i\ y\ y' \longleftrightarrow a \sharp v \wedge a \sharp i \wedge a \sharp y \wedge a \sharp y'$ (**is** *?thesis1*)
and *eval-fm-AbstAtomicP* [*simp*]:
eval-fm e (*AbstAtomicP v i y y'*) $\longleftrightarrow$ *AbstAtomic* $[\![v]\!]e$ $[\![i]\!]e$ $[\![y]\!]e$ $[\![y']\!]e$ (**is** *?thesis2*)
and *AbstAtomicP-sf* [*iff*]: *Sigma-fm* (*AbstAtomicP v i y y'*) (**is** *?thsf*)
proof –
obtain *t*::*name* **and** *u*::*name* **and** *t'*::*name* **and** *u'*::*name*
where $atom\ t \sharp (v,i,y,y',t',u,u')$ $atom\ t' \sharp (v,i,y,y',u,u')$

atom u $\sharp$ (*v*,*i*,*y*,*y*′,*u*′) *atom u*′ $\sharp$ (*v*,*i*,*y*,*y*′)
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thsf*
by (*auto simp*: *AbstAtomic-def q-defs*)
qed

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 −
obtain *t*::*name* **and** *u*::*name* **and** *t*′::*name* **and** *u*′::*name*
where *atom t* $\sharp$ (*v*,*tm*,*y*,*y*′,*w*,*i*,*t*′,*u*,*u*′) *atom t*′ $\sharp$ (*v*,*tm*,*y*,*y*′,*w*,*i*,*u*,*u*′)
atom u $\sharp$ (*v*,*tm*,*y*,*y*′,*w*,*i*,*u*′) *atom u*′ $\sharp$ (*v*,*tm*,*y*,*y*′,*w*,*i*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*simp add*: *AbstAtomicP.simps* [*of t - - - - t*′ *u u*′])
qed

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

5.6.2 The predicate *AbsMakeForm*

definition *AbstMakeForm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *AbstMakeForm k y y*′ *i u u*′ *j w w*′ ≡
Ord k ∧
((*k* = *i* ∧ *k* = *j* ∧ *y* = *q-Disj u w* ∧ *y*′ = *q-Disj u*′ *w*′) ∨
(*k* = *i* ∧ *y* = *q-Neg u* ∧ *y*′ = *q-Neg u*′) ∨
(*succ k* = *i* ∧ *y* = *q-Ex u* ∧ *y*′ = *q-Ex u*′))

definition *SeqAbstForm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SeqAbstForm v i x x*′ *s k* ≡
BuildSeq3 (*AbstAtomic v*) *AbstMakeForm s k i x x*′

nominal-function *SeqAbstFormP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *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*)⟧ ⟹
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′))))))))))))))))
by (*auto simp*: *eqvt-def SeqAbstFormP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

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 *eval-fm-SeqAbstFormP* [*simp*]:
eval-fm e (*SeqAbstFormP v i x x′ s k*) $\longleftrightarrow$ *SeqAbstForm* $[\![v]\!]e$ $[\![i]\!]e$ $[\![x]\!]e$ $[\![x']\!]e$ $[\![s]\!]e$ $[\![k]\!]e$ (**is** *?thesis2*)
and *SeqAbstFormP-sf* [*iff*]:
Sigma-fm (*SeqAbstFormP v i x x′ s k*) (**is** *?thsf*)
proof −
obtain *l*::*name* **and** *sli*::*name* **and** *sl*::*name* **and** *sl′*::*name* **and** *m*::*name* **and** *n*::*name* **and**
smi::*name* **and** *sm*::*name* **and** *sm′*::*name* **and** *sni*::*name* **and** *sn*::*name* **and** *sn′*::*name*
where *atoms*:
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*
by (*metis obtain-fresh*)
thus *?thesis1 ?thsf*
by (*auto intro*: *LstSeqP-OrdP*)
show *?thesis2* **using** *atoms*
by (*force simp add*: *LstSeq-imp-Ord SeqAbstForm-def*

BuildSeq3-def BuildSeq-def Builds-def
HBall-def HBex-def q-defs AbstMakeForm-def
Seq-iff-app [*of* ⟦*s*⟧*e*, *OF LstSeq-imp-Seq-succ*]
Ord-trans [*of* - - *succ* ⟦*k*⟧*e*]
cong: *conj-cong* *intro*!: *conj-cong* [*OF refl*] *all-cong*)
qed

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 −
obtain *l*::*name* **and** *sli*::*name* **and** *sl*::*name* **and** *sl′*::*name* **and** *m*::*name* **and** *n*::*name* **and**
smi::*name* **and** *sm*::*name* **and** *sm′*::*name* **and** *sni*::*name* **and** *sn*::*name* **and** *sn′*::*name*
where *atom l* ♯ (*i*,*t*,*s*,*k*,*v*,*sli*,*sl*,*sl′*,*m*,*n*,*smi*,*sm*,*sm′*,*sni*,*sn*,*sn′*)
atom sli ♯ (*i*,*t*,*s*,*v*,*sl*,*sl′*,*m*,*n*,*smi*,*sm*,*sm′*,*sni*,*sn*,*sn′*)
atom sl ♯ (*i*,*t*,*s*,*v*,*sl′*,*m*,*n*,*smi*,*sm*,*sm′*,*sni*,*sn*,*sn′*)
atom sl′ ♯ (*i*,*t*,*s*,*v*,*m*,*n*,*smi*,*sm*,*sm′*,*sni*,*sn*,*sn′*)
atom m ♯ (*i*,*t*,*s*,*n*,*smi*,*sm*,*sm′*,*sni*,*sn*,*sn′*)
atom n ♯ (*i*,*t*,*s*,*smi*,*sm*,*sm′*,*sni*,*sn*,*sn′*)
atom smi ♯ (*i*,*t*,*s*,*sm*,*sm′*,*sni*,*sn*,*sn′*)
atom sm ♯ (*i*,*t*,*s*,*sm′*,*sni*,*sn*,*sn′*) *atom sm′* ♯ (*i*,*t*,*s*,*sni*,*sn*,*sn′*)
atom sni ♯ (*i*,*t*,*s*,*sn*,*sn′*) *atom sn* ♯ (*i*,*t*,*s*,*sn′*) *atom sn′* ♯ (*i*,*t*,*s*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*force simp add*: *SeqAbstFormP.simps* [*of l* - - - *sli sl sl′ m n smi sm sm′ sni sn sn′*])
qed

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

5.6.3 Defining the syntax: the main AbstForm predicate

definition *AbstForm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *AbstForm v i x x′* ≡ *is-Var v* ∧ *Ord i* ∧ (∃ *s k*. *SeqAbstForm v i x x′ s k*)

nominal-function *AbstFormP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom s* ♯ (*v*,*i*,*x*,*x′*,*k*);
atom k ♯ (*v*,*i*,*x*,*x′*)⟧ ⟹
AbstFormP v i x x′ = *VarP v AND OrdP i AND Ex s* (*Ex k* (*SeqAbstFormP v i x x′* (*Var s*) (*Var k*)))
by (*auto simp*: *eqvt-def AbstFormP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma

shows *AbstFormP-fresh-iff* [*simp*]:
$a \sharp$ *AbstFormP* $v\ i\ x\ x' \longleftrightarrow a \sharp v \wedge a \sharp i \wedge a \sharp x \wedge a \sharp x'$ (**is** *?thesis1*)
and *eval-fm-AbstFormP* [*simp*]:
eval-fm e (*AbstFormP* $v\ i\ x\ x'$) $\longleftrightarrow$ *AbstForm* $[\![v]\!]e\ [\![i]\!]e\ [\![x]\!]e\ [\![x']\!]e$ (**is** *?thesis2*)
and *AbstFormP-sf* [*iff*]:
Sigma-fm (*AbstFormP* $v\ i\ x\ x'$) (**is** *?thsf*)
proof −
obtain s::*name* **and** k::*name* **where** *atom* $s \sharp (v,i,x,x',k)$ *atom* $k \sharp (v,i,x,x')$
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thsf*
by (*auto simp*: *AbstForm-def*)
qed

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 −
obtain s::*name* **and** k::*name* **where** *atom* $s \sharp (v,i,x,x',t,j,k)$ *atom* $k \sharp (v,i,x,x',t,j)$
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *AbstFormP.simps* [*of* s - - - - k])
qed

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

5.6.4 Correctness: It Coincides with Abstraction over real Formulas

lemma *AbstForm-imp-Ord*: *AbstForm* $v\ u\ x\ x' \Longrightarrow$ *Ord* v
by (*metis AbstForm-def is-Var-def*)

lemma *AbstForm-imp-abst-dbfm*:
assumes *AbstForm* $v\ i\ x\ x'$
shows $\exists A.\ x = [\![$*quot-dbfm* $A]\!]e \wedge$
$x' = [\![$*quot-dbfm* (*abst-dbfm* (*decode-Var* v) (*nat-of-ord* i) A)$]\!]e$
proof −
obtain $s\ k$ **where** v: *is-Var* v **and** i: *Ord* i **and** *sk*: *SeqAbstForm* $v\ i\ x\ x'\ s\ k$
using *assms* [*unfolded AbstForm-def*]
by *auto*
from *sk* [*unfolded SeqAbstForm-def*]
show *?thesis*
proof (*induction* $i\ x\ x'$ *rule*: *BuildSeq3-induct*)
case ($B\ i\ x\ x'$) **thus** *?case*
apply (*auto simp*: *AbstAtomic-def dest*!: *AbstTerm-imp-abst-dbtm* [**where** e=e])
apply (*rule-tac* [*1*] x=*DBEq ta tb* **in** *exI*)
apply (*rule-tac* [*2*] x=*DBMem ta tb* **in** *exI*)
apply (*auto simp*: *q-defs*)

```
    done
  next
    case (C i x x' j y y' k z z')
    then obtain A1 and A2
      where y = ⟦quot-dbfm A1⟧e
            y' = ⟦quot-dbfm (abst-dbfm (decode-Var v) (nat-of-ord j) A1)⟧e
            z = ⟦quot-dbfm A2⟧e
            z' = ⟦quot-dbfm (abst-dbfm (decode-Var v) (nat-of-ord k) A2)⟧e
    by blast
  with C.hyps show ?case
    apply (auto simp: AbstMakeForm-def)
    apply (rule-tac [1] x=DBDisj A1 A2 in exI)
    apply (rule-tac [2] x=DBNeg A1 in exI)
    apply (rule-tac [3] x=DBEx A1 in exI)
    apply (auto simp: C q-defs)
    done
  qed
qed

lemma AbstForm-abst-dbfm:
  AbstForm (q-Var i) (ord-of n) ⟦quot-dbfm fm⟧e ⟦quot-dbfm (abst-dbfm i n fm)⟧e
apply (induction fm arbitrary: n rule: dbfm.induct)
apply (force simp add: AbstForm-def SeqAbstForm-def AbstMakeForm-def AbstAtomic-def
                       AbstTerm-abst-dbtm htuple-minus-1 q-defs simp del: q-Var-def
             intro: BuildSeq3-exI BuildSeq3-combine)+
done
```

5.7 Substitution over formulas

5.7.1 The predicate *SubstAtomicP*

```
definition SubstAtomic :: hf ⇒ hf ⇒ hf ⇒ hf ⇒ bool
  where SubstAtomic v tm y y' ≡
          (∃ t u t' u'. SubstTerm v tm t t' ∧ SubstTerm v tm u u' ∧
           ((y = q-Eq t u ∧ y' = q-Eq t' u') ∨ (y = q-Mem t u ∧ y' = q-Mem t'
u')))

nominal-function SubstAtomicP :: tm ⇒ tm ⇒ tm ⇒ tm ⇒ fm
  where ⟦atom t ♯ (v,tm,y,y',t',u,u');
         atom t' ♯ (v,tm,y,y',u,u');
         atom u ♯ (v,tm,y,y',u');
         atom u' ♯ (v,tm,y,y')⟧ ⟹
    SubstAtomicP v tm y y' =
         Ex t (Ex u (Ex t' (Ex u'
          (SubstTermP v tm (Var t) (Var t') AND SubstTermP v tm (Var u) (Var
u') AND
                       ((y EQ Q-Eq (Var t) (Var u) AND y' EQ Q-Eq (Var t') (Var
u')) OR
                        (y EQ Q-Mem (Var t) (Var u) AND y' EQ Q-Mem (Var t')
```

$(Var\ u')))))))$
by (*auto simp*: *eqvt-def SubstAtomicP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma
  shows *SubstAtomicP-fresh-iff* [*simp*]:
    $a \sharp SubstAtomicP\ v\ tm\ y\ y' \longleftrightarrow a \sharp v \wedge a \sharp tm \wedge a \sharp y \wedge a \sharp y'$ (**is** *?thesis1*)
  and *eval-fm-SubstAtomicP* [*simp*]:
    $eval\text{-}fm\ e\ (SubstAtomicP\ v\ tm\ y\ y') \longleftrightarrow SubstAtomic\ [\![v]\!]e\ [\![tm]\!]e\ [\![y]\!]e\ [\![y']\!]e$ (**is** *?thesis2*)
  and *SubstAtomicP-sf* [*iff*]: $Sigma\text{-}fm\ (SubstAtomicP\ v\ tm\ y\ y')$ (**is** *?thsf*)
proof –
  obtain t::*name* **and** u::*name* **and** t'::*name* **and** u'::*name*
    where $atom\ t \sharp (v,tm,y,y',t',u,u')$ $atom\ t' \sharp (v,tm,y,y',u,u')$
      $atom\ u \sharp (v,tm,y,y',u')$ $atom\ u' \sharp (v,tm,y,y')$
    by (*metis obtain-fresh*)
  thus *?thesis1 ?thesis2 ?thsf*
    by (*auto simp*: *SubstAtomic-def q-defs*)
qed

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 –
  obtain t::*name* **and** u::*name* **and** t'::*name* **and** u'::*name*
    where $atom\ t \sharp (v,tm,y,y',w,i,t',u,u')$ $atom\ t' \sharp (v,tm,y,y',w,i,u,u')$
      $atom\ u \sharp (v,tm,y,y',w,i,u')$ $atom\ u' \sharp (v,tm,y,y',w,i)$
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*simp add*: *SubstAtomicP.simps* [*of t - - - - t' u u'*])
qed

lemma *SubstAtomicP-cong*:
  $[\![H \vdash v\ EQ\ v';\ H \vdash tm\ EQ\ tm';\ H \vdash x\ EQ\ x';\ H \vdash y\ EQ\ y']\!]$
  $\Longrightarrow H \vdash SubstAtomicP\ v\ tm\ x\ y\ IFF\ SubstAtomicP\ v'\ tm'\ x'\ y'$
  by (*rule P4-cong*) *auto*

5.7.2 The predicate *SubstMakeForm*

definition $SubstMakeForm :: hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$
  where $SubstMakeForm\ y\ y'\ u\ u'\ w\ w' \equiv$
    $((y = q\text{-}Disj\ u\ w \wedge y' = q\text{-}Disj\ u'\ w') \vee$
    $(y = q\text{-}Neg\ u \wedge y' = q\text{-}Neg\ u') \vee$
    $(y = q\text{-}Ex\ u \wedge y' = q\text{-}Ex\ u'))$

definition *SeqSubstForm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
 where *SeqSubstForm v u x x′ s k* ≡ *BuildSeq2* (*SubstAtomic v u*) *SubstMakeForm s k x x′*

nominal-function *SeqSubstFormP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
 where ⟦*atom l* ♯ (*s*,*k*,*v*,*u*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*);
 atom sl ♯ (*s*,*v*,*u*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*);
 atom sl′ ♯ (*s*,*v*,*u*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*);
 atom m ♯ (*s*,*n*,*sm*,*sm′*,*sn*,*sn′*); *atom n* ♯ (*s*,*sm*,*sm′*,*sn*,*sn′*);
 atom sm ♯ (*s*,*sm′*,*sn*,*sn′*); *atom sm′* ♯ (*s*,*sn*,*sn′*);
 atom sn ♯ (*s*,*sn′*); *atom sn′* ♯ *s*⟧ ⟹
 SeqSubstFormP v u x x′ s k =
 LstSeqP s k (*HPair x x′*) *AND*
 All2 l (*SUCC k*) (*Ex sl* (*Ex sl′* (*HPair* (*Var l*) (*HPair* (*Var sl*) (*Var sl′*)) *IN s AND*
 (*SubstAtomicP v u* (*Var sl*) (*Var sl′*) *OR*
 Ex m (*Ex n* (*Ex sm* (*Ex sm′* (*Ex sn* (*Ex sn′* (*Var m IN Var l AND Var n IN Var l AND*
 HPair (*Var m*) (*HPair* (*Var sm*) (*Var sm′*)) *IN s AND*
 HPair (*Var n*) (*HPair* (*Var sn*) (*Var sn′*)) *IN s AND*
 ((*Var sl EQ Q-Disj* (*Var sm*) (*Var sn*) *AND*
 Var sl′ EQ Q-Disj (*Var sm′*) (*Var sn′*)) *OR*
 (*Var sl EQ Q-Neg* (*Var sm*) *AND Var sl′ EQ Q-Neg* (*Var sm′*)) *OR*
 (*Var sl EQ Q-Ex* (*Var sm*) *AND Var sl′ EQ Q-Ex* (*Var sm′*)))))))))))))
by (*auto simp*: *eqvt-def SeqSubstFormP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
 by *lexicographic-order*

lemma
 shows *SeqSubstFormP-fresh-iff* [*simp*]:
 a ♯ *SeqSubstFormP v u x x′ s k* ⟷ *a* ♯ *v* ∧ *a* ♯ *u* ∧ *a* ♯ *x* ∧ *a* ♯ *x′* ∧ *a* ♯ *s* ∧ *a* ♯ *k* (**is** *?thesis1*)
 and *eval-fm-SeqSubstFormP* [*simp*]:
 eval-fm e (*SeqSubstFormP v u x x′ s k*) ⟷
 SeqSubstForm ⟦*v*⟧*e* ⟦*u*⟧*e* ⟦*x*⟧*e* ⟦*x′*⟧*e* ⟦*s*⟧*e* ⟦*k*⟧*e* (**is** *?thesis2*)
 and *SeqSubstFormP-sf* [*iff*]:
 Sigma-fm (*SeqSubstFormP v u x x′ s k*) (**is** *?thsf*)
 and *SeqSubstFormP-imp-OrdP*:
 { *SeqSubstFormP v u x x′ s k* } ⊢ *OrdP k* (**is** *?thOrd*)
 and *SeqSubstFormP-imp-LstSeqP*:
 { *SeqSubstFormP v u x x′ s k* } ⊢ *LstSeqP s k* (*HPair x x′*) (**is** *?thLstSeq*)
proof −
 obtain *l*::*name* **and** *sl*::*name* **and** *sl′*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sm′*::*name* **and** *sn*::*name* **and** *sn′*::*name*
 where *atoms*:
 atom l ♯ (*s*,*k*,*v*,*u*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)

atom sl ♯ (*s*,*v*,*u*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl' ♯ (*s*,*v*,*u*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom m ♯ (*s*,*n*,*sm*,*sm'*,*sn*,*sn'*) *atom n* ♯ (*s*,*sm*,*sm'*,*sn*,*sn'*)
atom sm ♯ (*s*,*sm'*,*sn*,*sn'*) *atom sm'* ♯ (*s*,*sn*,*sn'*)
atom sn ♯ (*s*,*sn'*) *atom sn'* ♯ (*s*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thsf ?thOrd ?thLstSeq*
by (*auto intro*: *LstSeqP-OrdP*)
show *?thesis2* **using** *atoms*
by (*force simp add*: *LstSeq-imp-Ord SeqSubstForm-def*
BuildSeq2-def BuildSeq-def Builds-def
HBall-def HBex-def q-defs SubstMakeForm-def
Seq-iff-app [*of* ⟦*s*⟧*e*, *OF LstSeq-imp-Seq-succ*]
Ord-trans [*of - - succ* ⟦*k*⟧*e*]
cong: *conj-cong* *intro*!: *conj-cong* [*OF refl*] *all-cong*)
qed

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 −
obtain *l*::*name* **and** *sl*::*name* **and** *sl'*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sm'*::*name* **and** *sn*::*name* **and** *sn'*::*name*
where *atom l* ♯ (*s*,*k*,*v*,*u*,*t*,*i*,*sl*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl ♯ (*s*,*v*,*u*,*t*,*i*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl' ♯ (*s*,*v*,*u*,*t*,*i*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom m ♯ (*s*,*t*,*i*,*n*,*sm*,*sm'*,*sn*,*sn'*) *atom n* ♯ (*s*,*t*,*i*,*sm*,*sm'*,*sn*,*sn'*)
atom sm ♯ (*s*,*t*,*i*,*sm'*,*sn*,*sn'*) *atom sm'* ♯ (*s*,*t*,*i*,*sn*,*sn'*)
atom sn ♯ (*s*,*t*,*i*,*sn'*) *atom sn'* ♯ (*s*,*t*,*i*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*force simp add*: *SeqSubstFormP.simps* [*of l - - - - sl sl' m n sm sm' sn sn'*])
qed

lemma *SeqSubstFormP-cong*:
⟦*H* ⊢ *t EQ t'*; *H* ⊢ *u EQ u'*; *H* ⊢ *s EQ s'*; *H* ⊢ *k EQ k'*⟧
⟹ *H* ⊢ *SeqSubstFormP v i t u s k IFF SeqSubstFormP v i t' u' s' k'*
by (*rule P4-cong* [**where** *tms*=[*v*,*i*]]) (*auto simp*: *fresh-Cons*)

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

5.7.3 Defining the syntax: the main SubstForm predicate

definition *SubstForm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SubstForm v u x x'* ≡ *is-Var v* ∧ *Term u* ∧ (∃ *s k*. *SeqSubstForm v u x x' s k*)

nominal-function *SubstFormP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*

where $[\![atom\ s\ \sharp\ (v,i,x,x',k);\ atom\ k\ \sharp\ (v,i,x,x')]\!] \Longrightarrow$
$SubstFormP\ v\ i\ x\ x' =$
$VarP\ v\ AND\ TermP\ i\ AND\ Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ x\ x'\ (Var\ s)\ (Var\ k)))$
by (*auto simp*: *eqvt-def SubstFormP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *SubstFormP-fresh-iff* [*simp*]:
$a\ \sharp\ SubstFormP\ v\ i\ x\ x' \longleftrightarrow a\ \sharp\ v \wedge a\ \sharp\ i \wedge a\ \sharp\ x \wedge a\ \sharp\ x'$ (**is** *?thesis1*)
and *eval-fm-SubstFormP* [*simp*]:
$eval\text{-}fm\ e\ (SubstFormP\ v\ i\ x\ x') \longleftrightarrow SubstForm\ [\![v]\!]e\ [\![i]\!]e\ [\![x]\!]e\ [\![x']\!]e$ (**is** *?thesis2*)
and *SubstFormP-sf* [*iff*]:
$Sigma\text{-}fm\ (SubstFormP\ v\ i\ x\ x')$ (**is** *?thsf*)
proof −
obtain *s*::*name* **and** *k*::*name*
where $atom\ s\ \sharp\ (v,i,x,x',k)$ $atom\ k\ \sharp\ (v,i,x,x')$
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thsf*
by (*auto simp*: *SubstForm-def*)
qed

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 −
obtain *s*::*name* **and** *k*::*name* **where** $atom\ s\ \sharp\ (v,i,x,x',t,j,k)$ $atom\ k\ \sharp\ (v,i,x,x',t,j)$
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SubstFormP.simps* [*of s - - - - k*])
qed

lemma *SubstFormP-cong*:
$[\![H \vdash v\ EQ\ v';\ H \vdash i\ EQ\ i';\ H \vdash t\ EQ\ t';\ H \vdash u\ EQ\ u']\!]$
$\Longrightarrow H \vdash SubstFormP\ v\ i\ t\ u\ IFF\ SubstFormP\ v'\ i'\ t'\ u'$
by (*rule P4-cong*) *auto*

lemma *ground-SubstFormP* [*simp*]: $ground\text{-}fm\ (SubstFormP\ v\ y\ x\ x') \longleftrightarrow ground\ v \wedge ground\ y \wedge ground\ x \wedge ground\ x'$
by (*auto simp*: *ground-aux-def ground-fm-aux-def supp-conv-fresh*)

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

5.7.4 Correctness of substitution over formulas

lemma *SubstForm-imp-subst-dbfm-lemma*:

assumes *SubstForm v* $[\![$*quot-dbtm u*$]\!]$*e x x'*
shows $\exists$ *A. x* = $[\![$*quot-dbfm A*$]\!]$*e* $\wedge$
x' = $[\![$*quot-dbfm* (*subst-dbfm u* (*decode-Var v*) *A*)$]\!]$*e*
proof –
obtain *s k* **where** *v*: *is-Var v* **and** *u*: *Term* $[\![$*quot-dbtm u*$]\!]$*e*
and *sk*: *SeqSubstForm v* $[\![$*quot-dbtm u*$]\!]$*e x x' s k*
using *assms* [*unfolded SubstForm-def*]
by *blast*
from *sk* [*unfolded SeqSubstForm-def*]
show *?thesis*
proof (*induct x x' rule*: *BuildSeq2-induct*)
case (*B x x'*) **thus** *?case*
apply (*auto simp*: *SubstAtomic-def elim*!: *SubstTerm-imp-subst-dbtm'* [**where** *e*=*e*])
apply (*rule-tac* [*1*] *x*=*DBEq ta tb* **in** *exI*)
apply (*rule-tac* [*2*] *x*=*DBMem ta tb* **in** *exI*)
apply (*auto simp*: *q-defs*)
done
next
case (*C x x' y y' z z'*)
then obtain *A* **and** *B*
where *y* = $[\![$*quot-dbfm A*$]\!]$*e y'* = $[\![$*quot-dbfm* (*subst-dbfm u* (*decode-Var v*) *A*)$]\!]$*e*
z = $[\![$*quot-dbfm B*$]\!]$*e z'* = $[\![$*quot-dbfm* (*subst-dbfm u* (*decode-Var v*) *B*)$]\!]$*e*
by *blast*
with *C.hyps* **show** *?case*
apply (*auto simp*: *SubstMakeForm-def*)
apply (*rule-tac* [*1*] *x*=*DBDisj A B* **in** *exI*)
apply (*rule-tac* [*2*] *x*=*DBNeg A* **in** *exI*)
apply (*rule-tac* [*3*] *x*=*DBEx A* **in** *exI*)
apply (*auto simp*: *C q-defs*)
done
qed
qed

lemma *SubstForm-imp-subst-dbfm*:
assumes *SubstForm v u x x'*
obtains *t A* **where** *u* = $[\![$*quot-dbtm t*$]\!]$*e*
x = $[\![$*quot-dbfm A*$]\!]$*e*
x' = $[\![$*quot-dbfm* (*subst-dbfm t* (*decode-Var v*) *A*)$]\!]$*e*
proof –
obtain *t* **where** *u* = $[\![$*quot-dbtm t*$]\!]$*e*
using *assms* [*unfolded SubstForm-def*]
by (*metis Term-imp-wf-dbtm*)
thus *?thesis*
by (*metis SubstForm-imp-subst-dbfm-lemma assms that*)
qed

lemma *SubstForm-subst-dbfm*:

assumes *u*: *wf-dbtm u*
shows *SubstForm* (*q-Var i*) ⟦*quot-dbtm u*⟧*e* ⟦*quot-dbfm A*⟧*e*
⟦*quot-dbfm* (*subst-dbfm u i A*)⟧*e*
apply (*induction A rule*: *dbfm.induct*)
apply (*force simp*: *u SubstForm-def SeqSubstForm-def SubstAtomic-def SubstMakeForm-def*

SubstTerm-subst-dbtm q-defs simp del: *q-Var-def*
intro: *BuildSeq2-exI BuildSeq2-combine*)+
done

corollary *SubstForm-subst-dbfm-eq*:
⟦$v = $ *q-Var i*; *Term ux*; $ux = $ ⟦*quot-dbtm u*⟧*e*; $A' = $ *subst-dbfm u i A*⟧
⟹ *SubstForm v ux* ⟦*quot-dbfm A*⟧*e* ⟦*quot-dbfm A'*⟧*e*
by (*metis SubstForm-subst-dbfm Term-imp-is-tm quot-dbtm-inject-lemma quot-tm-def wf-dbtm-iff-is-tm*)

5.8 The predicate *AtomicP*

definition *Atomic* :: *hf* ⇒ *bool*
where *Atomic y* ≡∃ *t u*. *Term t* ∧ *Term u* ∧ (*y* = *q-Eq t u* ∨ *y* = *q-Mem t u*)

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*))))
by (*auto simp*: *eqvt-def AtomicP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *AtomicP-fresh-iff* [*simp*]: *a* ♯ *AtomicP y* ⟷ *a* ♯ *y* (**is** *?thesis1*)
and *eval-fm-AtomicP* [*simp*]: *eval-fm e* (*AtomicP y*) ⟷ *Atomic*⟦*y*⟧*e* (**is** *?thesis2*)
and *AtomicP-sf* [*iff*]: *Sigma-fm* (*AtomicP y*) (**is** *?thsf*)
proof −
obtain *t*::*name* **and** *u*::*name* **where** *atom t* ♯ (*u*,*y*) *atom u* ♯ *y*
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thsf*
by (*auto simp*: *Atomic-def q-defs*)
qed

5.9 The predicate *MakeForm*

definition *MakeForm* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *MakeForm y u w* ≡
y = *q-Disj u w* ∨ *y* = *q-Neg u* ∨

($\exists$ *v u'. AbstForm v 0 u u'* $\wedge$ *y* = *q-Ex u'*)

nominal-function *MakeFormP* :: *tm* $\Rightarrow$ *tm* $\Rightarrow$ *tm* $\Rightarrow$ *fm*
where $[\![$*atom v* $\sharp$ (*y*,*u*,*w*,*au*); *atom au* $\sharp$ (*y*,*u*,*w*)$]\!]$ $\Longrightarrow$
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*)))
by (*auto simp*: *eqvt-def MakeFormP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *MakeFormP-fresh-iff* [*simp*]:
a $\sharp$ *MakeFormP y u w* $\longleftrightarrow$ *a* $\sharp$ *y* $\wedge$ *a* $\sharp$ *u* $\wedge$ *a* $\sharp$ *w* (**is** *?thesis1*)
and *eval-fm-MakeFormP* [*simp*]:
eval-fm e (*MakeFormP y u w*) $\longleftrightarrow$ *MakeForm* $[\![y]\!]e$ $[\![u]\!]e$ $[\![w]\!]e$ (**is** *?thesis2*)
and *MakeFormP-sf* [*iff*]:
Sigma-fm (*MakeFormP y u w*) (**is** *?thsf*)
proof −
obtain *v*::*name* **and** *au*::*name* **where** *atom v* $\sharp$ (*y*,*u*,*w*,*au*) *atom au* $\sharp$ (*y*,*u*,*w*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thsf*
by (*auto simp*: *MakeForm-def q-defs*)
qed

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

5.10 The predicate *SeqFormP*

definition *SeqForm* :: *hf* $\Rightarrow$ *hf* $\Rightarrow$ *hf* $\Rightarrow$ *bool*
where *SeqForm s k y* $\equiv$ *BuildSeq Atomic MakeForm s k y*

nominal-function *SeqFormP* :: *tm* $\Rightarrow$ *tm* $\Rightarrow$ *tm* $\Rightarrow$ *fm*
where $[\![$*atom l* $\sharp$ (*s*,*k*,*t*,*sl*,*m*,*n*,*sm*,*sn*); *atom sl* $\sharp$ (*s*,*k*,*t*,*m*,*n*,*sm*,*sn*);
atom m $\sharp$ (*s*,*k*,*t*,*n*,*sm*,*sn*); *atom n* $\sharp$ (*s*,*k*,*t*,*sm*,*sn*);
atom sm $\sharp$ (*s*,*k*,*t*,*sn*); *atom sn* $\sharp$ (*s*,*k*,*t*)$]\!]$ $\Longrightarrow$
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*))))))))
by (*auto simp*: *eqvt-def SeqFormP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *SeqFormP-fresh-iff* [*simp*]:
$a \sharp$ *SeqFormP s k t* $\longleftrightarrow a \sharp s \wedge a \sharp k \wedge a \sharp t$ (**is** *?thesis1*)
and *eval-fm-SeqFormP* [*simp*]:
eval-fm e (*SeqFormP s k t*) $\longleftrightarrow$ *SeqForm* $[\![s]\!]e$ $[\![k]\!]e$ $[\![t]\!]e$ (**is** *?thesis2*)
and *SeqFormP-sf* [*iff*]: *Sigma-fm* (*SeqFormP s k t*) (**is** *?thsf*)
proof −
obtain *l*::*name* **and** *sl*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name*
where *atoms*: *atom l* $\sharp$ (*s*,*k*,*t*,*sl*,*m*,*n*,*sm*,*sn*) *atom sl* $\sharp$ (*s*,*k*,*t*,*m*,*n*,*sm*,*sn*)
atom m $\sharp$ (*s*,*k*,*t*,*n*,*sm*,*sn*) *atom n* $\sharp$ (*s*,*k*,*t*,*sm*,*sn*)
atom sm $\sharp$ (*s*,*k*,*t*,*sn*) *atom sn* $\sharp$ (*s*,*k*,*t*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thsf*
by *auto*
show *?thesis2* **using** *atoms*
by (*simp cong*: *conj-cong add*: *LstSeq-imp-Ord SeqForm-def BuildSeq-def Builds-def*
HBall-def HBex-def q-defs
Seq-iff-app [*of* $[\![s]\!]e$, *OF LstSeq-imp-Seq-succ*]
Ord-trans [*of* - - *succ* $[\![k]\!]e$])
qed

lemma *SeqFormP-subst* [*simp*]:
(*SeqFormP s k t*)(*j*::=*w*) = *SeqFormP* (*subst j w s*) (*subst j w k*) (*subst j w t*)
proof −
obtain *l*::*name* **and** *sl*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name*
where *atom l* $\sharp$ (*j*,*w*,*s*,*t*,*k*,*sl*,*m*,*n*,*sm*,*sn*) *atom sl* $\sharp$ (*j*,*w*,*s*,*k*,*t*,*m*,*n*,*sm*,*sn*)
atom m $\sharp$ (*j*,*w*,*s*,*k*,*t*,*n*,*sm*,*sn*) *atom n* $\sharp$ (*j*,*w*,*s*,*k*,*t*,*sm*,*sn*)
atom sm $\sharp$ (*j*,*w*,*s*,*k*,*t*,*sn*) *atom sn* $\sharp$ (*j*,*w*,*s*,*k*,*t*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SeqFormP.simps* [*of l* - - - *sl m n sm sn*])
qed

5.11 The predicate *FormP*

5.11.1 Definition

definition *Form* :: *hf* ⇒ *bool*
where *Form y* ≡ (∃ *s k*. *SeqForm s k y*)

nominal-function *FormP* :: *tm* ⇒ *fm*
where $[\![$*atom k* $\sharp$ (*s*,*y*); *atom s* $\sharp$ *y*$]\!]$ ⟹
FormP y = *Ex k* (*Ex s* (*SeqFormP* (*Var s*) (*Var k*) *y*))
by (*auto simp*: *eqvt-def FormP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma
  shows *FormP-fresh-iff* [*simp*]: $a \sharp$ *FormP* $y \longleftrightarrow a \sharp y$ (**is** *?thesis1*)
  and *eval-fm-FormP* [*simp*]: *eval-fm e* (*FormP y*) $\longleftrightarrow$ *Form* $[\![y]\!]e$ (**is** *?thesis2*)
  and *FormP-sf* [*iff*]: *Sigma-fm* (*FormP y*) (**is** *?thsf*)
proof −
  obtain *k*::*name* **and** *s*::*name* **where** *k*: *atom k* $\sharp$ (*s*,*y*) *atom s* $\sharp$ *y*
    by (*metis obtain-fresh*)
  thus *?thesis1 ?thesis2 ?thsf*
    by (*auto simp*: *Form-def*)
qed

lemma *FormP-subst* [*simp*]: (*FormP y*)(*j*::=*w*) = *FormP* (*subst j w y*)
proof −
  obtain *k*::*name* **and** *s*::*name* **where** *atom k* $\sharp$ (*s*,*j*,*w*,*y*) *atom s* $\sharp$ (*j*,*w*,*y*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *FormP.simps* [*of k s*])
qed

5.11.2 Correctness: It Corresponds to Quotations of Real Formulas

lemma *AbstForm-trans-fm*:
  AbstForm (*q-Var i*) *0* $[\![\lceil A\rceil]\!]e$ $[\![$*quot-dbfm* (*trans-fm* [*i*] *A*)$]\!]e$
  by (*metis abst-trans-fm ord-of.simps*(*1*) *quot-fm-def AbstForm-abst-dbfm*)

corollary *AbstForm-trans-fm-eq*:
  $[\![x = [\![\lceil A\rceil]\!]\ e;\ \ x' = [\![$*quot-dbfm* (*trans-fm* [*i*] *A*)$]\!]e]\!] \Longrightarrow$ *AbstForm* (*q-Var i*) *0 x*
x'
  by (*metis AbstForm-trans-fm*)

lemma *wf-Form-quot-dbfm* [*simp*]:
  assumes *wf-dbfm A* **shows** *Form* $[\![$*quot-dbfm A*$]\!]e$
using *assms*
proof (*induct rule*: *wf-dbfm.induct*)
  case (*Mem tm1 tm2*)
  hence *Atomic* $[\![$*quot-dbfm* (*DBMem tm1 tm2*)$]\!]e$
    by (*auto simp*: *Atomic-def quot-Mem q-Mem-def dest*: *wf-Term-quot-dbtm*)
  thus *?case*
    by (*auto simp*: *Form-def SeqForm-def BuildSeq-exI*)
next
  case (*Eq tm1 tm2*)
  hence *Atomic* $[\![$*quot-dbfm* (*DBEq tm1 tm2*)$]\!]e$
    by (*auto simp*: *Atomic-def quot-Eq q-Eq-def dest*: *wf-Term-quot-dbtm*)
  thus *?case*

by (*auto simp*: *Form-def SeqForm-def BuildSeq-exI*)
next
  case (*Disj A1 A2*)
   have *MakeForm* ⟦*quot-dbfm* (*DBDisj A1 A2*)⟧*e* ⟦*quot-dbfm A1*⟧*e* ⟦*quot-dbfm A2*⟧*e*
      by (*simp add*: *quot-Disj q-Disj-def MakeForm-def*)
  thus *?case* **using** *Disj*
    by (*force simp add*: *Form-def SeqForm-def intro*: *BuildSeq-combine*)
next
  case (*Neg A*)
    have $\bigwedge y$. *MakeForm* ⟦*quot-dbfm* (*DBNeg A*)⟧*e* ⟦*quot-dbfm A*⟧*e y*
      by (*simp add*: *quot-Neg q-Neg-def MakeForm-def*)
  thus *?case* **using** *Neg*
    by (*force simp add*: *Form-def SeqForm-def intro*: *BuildSeq-combine*)
next
  case (*Ex A i*)
  have $\bigwedge A$ *y*. *MakeForm* ⟦*quot-dbfm* (*DBEx* (*abst-dbfm i 0 A*))⟧*e* ⟦*quot-dbfm A*⟧*e y*
   by (*simp add*: *quot-Ex q-defs MakeForm-def*) (*metis AbstForm-abst-dbfm ord-of.simps*(*1*))
  thus *?case* **using** *Ex*
    by (*force simp add*: *Form-def SeqForm-def intro*: *BuildSeq-combine*)
qed

lemma *Form-quot-fm* [*iff*]: **fixes** *A* :: *fm* **shows** *Form* ⟦⌈*A*⌉⟧*e*
  by (*metis quot-fm-def wf-Form-quot-dbfm wf-dbfm-trans-fm*)

lemma *Atomic-Form-is-wf-dbfm*: *Atomic x* ⟹ ∃ *A*. *wf-dbfm A* ∧ *x* = ⟦*quot-dbfm A*⟧*e*
proof (*auto simp*: *Atomic-def*)
  fix *t u*
  assume *t*: *Term t* **and** *u*: *Term u*
  then obtain *tm1* **and** *tm2*
    where *tm1*: *wf-dbtm tm1 t* = ⟦*quot-dbtm tm1*⟧*e*
      and *tm2*: *wf-dbtm tm2 u* = ⟦*quot-dbtm tm2*⟧*e*
      by (*metis Term-imp-is-tm quot-tm-def wf-dbtm-trans-tm*)+
  thus ∃ *A*. *wf-dbfm A* ∧ *q-Eq t u* = ⟦*quot-dbfm A*⟧*e*
    by (*auto simp*: *quot-Eq q-Eq-def*)
next
  fix *t u*
  assume *t*: *Term t* **and** *u*: *Term u*
  then obtain *tm1* **and** *tm2*
    where *tm1*: *wf-dbtm tm1 t* = ⟦*quot-dbtm tm1*⟧*e*
      and *tm2*: *wf-dbtm tm2 u* = ⟦*quot-dbtm tm2*⟧*e*
      by (*metis Term-imp-is-tm quot-tm-def wf-dbtm-trans-tm*)+
  thus ∃ *A*. *wf-dbfm A* ∧ *q-Mem t u* = ⟦*quot-dbfm A*⟧*e*
    by (*auto simp*: *quot-Mem q-Mem-def*)
qed

lemma *SeqForm-imp-wf-dbfm*:

assumes *SeqForm s k x*
shows $\exists$ *A. wf-dbfm A* $\wedge$ *x* = $[\![$*quot-dbfm A*$]\!]$*e*
using *assms* [*unfolded SeqForm-def*]
proof (*induct x rule*: *BuildSeq-induct*)
case (*B x*) **thus** *?case*
by (*rule Atomic-Form-is-wf-dbfm*)
next
case (*C x y z*)
then obtain *A B* **where** *wf-dbfm A y* = $[\![$*quot-dbfm A*$]\!]$*e*
wf-dbfm B z = $[\![$*quot-dbfm B*$]\!]$*e*
by *blast*
thus *?case* **using** *C*
apply (*auto simp*: *MakeForm-def dest*!: *AbstForm-imp-abst-dbfm* [**where** *e*=*e*])
apply (*rule exI* [**where** *x*=*DBDisj A B*])
apply (*rule-tac* [*2*] *x*=*DBNeg A* **in** *exI*)
apply (*rule-tac* [*3*] *x*=*DBEx* (*abst-dbfm* (*decode-Var v*) *0 A*) **in** *exI*)
apply (*auto simp*: *q-defs*)
done
qed

lemma *Form-imp-wf-dbfm*:
assumes *Form x* **obtains** *A* **where** *wf-dbfm A x* = $[\![$*quot-dbfm A*$]\!]$*e*
by (*metis assms SeqForm-imp-wf-dbfm Form-def*)

lemma *Form-imp-is-fm*: **assumes** *Form x* **obtains** *A*::*fm* **where** *x* = $[\![\lceil A\rceil]\!]$ *e*
by (*metis assms Form-imp-wf-dbfm quot-fm-def wf-dbfm-imp-is-fm*)

lemma *SubstForm-imp-subst-fm*:
assumes *SubstForm v* $[\![\lceil u\rceil]\!]$*e x x′ Form x*
obtains *A*::*fm* **where** *x* = $[\![\lceil A\rceil]\!]$ *e x′* = $[\![\lceil A($*decode-Var v*::=*u*$)\rceil]\!]$ *e*
using *assms* [*unfolded quot-tm-def*]
by (*auto simp*: *quot-fm-def dest*!: *SubstForm-imp-subst-dbfm-lemma*)
(*metis Form-imp-is-fm eval-quot-dbfm-ignore quot-dbfm-inject-lemma quot-fm-def*)

lemma *SubstForm-unique*:
assumes *is-Var v* **and** *Term y* **and** *Form x*
shows *SubstForm v y x x′* $\longleftrightarrow$
($\exists$ *t*::*tm*. *y* = $[\![\lceil t\rceil]\!]$*e* $\wedge$ ($\exists$ *A*::*fm*. *x* = $[\![\lceil A\rceil]\!]$*e* $\wedge$ *x′* = $[\![\lceil A($*decode-Var v*::=*t*$)\rceil]\!]$*e*))
using *assms*
apply (*auto elim*!: *Term-imp-wf-dbtm* [**where** *e*=*e*] *Form-imp-is-fm* [**where** *e*=*e*]
SubstForm-imp-subst-dbfm [**where** *e*=*e*])
apply (*auto simp*: *quot-tm-def quot-fm-def is-Var-iff q-Var-def intro*: *SubstForm-subst-dbfm-eq*)
apply (*metis subst-fm-trans-commute wf-dbtm-imp-is-tm*)
done

lemma *SubstForm-quot-unique*: *SubstForm* (*q-Var i*) $[\![\lceil t\rceil]\!]$*e* $[\![\lceil A\rceil]\!]$*e x′* $\longleftrightarrow$ *x′* = $[\![\lceil A(i::=t)\rceil]\!]$ *e*

by (*subst SubstForm-unique* [**where** *e*=*e*]) *auto*

lemma *SubstForm-quot*: *SubstForm* ⟦⌈*Var i*⌉⟧*e* ⟦⌈*t*⌉⟧*e* ⟦⌈*A*⌉⟧*e* ⟦⌈*A*(*i*::=*t*)⌉⟧*e*
by (*metis SubstForm-quot-unique eval-Var-q*)

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

definition *VarNonOccForm* :: *hf* ⇒ *hf* ⇒ *bool*
where *VarNonOccForm v x* ≡ *Form x* ∧ *SubstForm v 0 x x*

nominal-function *VarNonOccFormP* :: *tm* ⇒ *tm* ⇒ *fm*
where *VarNonOccFormP v x* = *FormP x AND SubstFormP v Zero x x*
by (*auto simp*: *eqvt-def VarNonOccFormP-graph-aux-def*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *VarNonOccFormP-fresh-iff* [*simp*]: *a* ♯ *VarNonOccFormP v y* ⟷ *a* ♯ *v* ∧ *a* ♯ *y* (**is** *?thesis1*)
and *eval-fm-VarNonOccFormP* [*simp*]:
eval-fm e (*VarNonOccFormP v y*) ⟷ *VarNonOccForm* ⟦*v*⟧*e* ⟦*y*⟧*e* (**is** *?thesis2*)
and *VarNonOccFormP-sf* [*iff*]: *Sigma-fm* (*VarNonOccFormP v y*) (**is** *?thsf*)
proof −
show *?thesis1 ?thsf ?thesis2*
by (*auto simp add*: *VarNonOccForm-def*)
qed

5.11.4 Correctness for Real Terms and Formulas

lemma *VarNonOccForm-imp-dbfm-fresh*:
assumes *VarNonOccForm v x*
shows ∃ *A*. *wf-dbfm A* ∧ *x* = ⟦*quot-dbfm A*⟧*e* ∧ *atom* (*decode-Var v*) ♯ *A*
proof −
obtain *A′* **where** *A′*: *wf-dbfm A′ x* = ⟦*quot-dbfm A′*⟧*e SubstForm v* ⟦*quot-dbtm DBZero*⟧*e x x*
using *assms* [*unfolded VarNonOccForm-def*]
by *auto* (*metis Form-imp-wf-dbfm*)
then obtain *A* **where** *x* = ⟦*quot-dbfm A*⟧*e*
x = ⟦*quot-dbfm* (*subst-dbfm DBZero* (*decode-Var v*) *A*)⟧*e*
by (*metis SubstForm-imp-subst-dbfm-lemma*)
thus *?thesis* **using** *A′*
by *auto* (*metis fresh-iff-non-subst-dbfm*)
qed

corollary *VarNonOccForm-imp-fresh*:
assumes *VarNonOccForm v x* **obtains** *A*::*fm* **where** *x* = ⟦⌈*A*⌉⟧*e atom* (*decode-Var*

v) ♯ *A*
using *VarNonOccForm-imp-dbfm-fresh* [*OF assms*, **where** *e*=*e*]
by (*auto simp*: *quot-fm-def wf-dbfm-iff-is-fm*)

lemma *VarNonOccForm-dbfm*:
wf-dbfm A ⟹ *atom i* ♯ *A* ⟹ *VarNonOccForm* (*q-Var i*) ⟦*quot-dbfm A*⟧*e*
by (*auto intro*: *SubstForm-subst-dbfm-eq* [**where** *u*=*DBZero*]
simp add: *VarNonOccForm-def Const-0 Const-imp-Term fresh-iff-non-subst-dbfm* [*symmetric*])

corollary *fresh-imp-VarNonOccForm*:
fixes *A*::*fm* **shows** *atom i* ♯ *A* ⟹ *VarNonOccForm* (*q-Var i*) ⟦⌈*A*⌉⟧*e*
by (*simp add*: *quot-fm-def wf-dbfm-trans-fm VarNonOccForm-dbfm*)

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

definition *Sent-axioms* :: $hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$ **where**
$Sent\text{-}axioms\ x\ y\ z\ w \equiv$
$x = q\text{-}Imp\ y\ y\ \vee$
$x = q\text{-}Imp\ y\ (q\text{-}Disj\ y\ z)\ \vee$
$x = q\text{-}Imp\ (q\text{-}Disj\ y\ y)\ y\ \vee$
$x = q\text{-}Imp\ (q\text{-}Disj\ y\ (q\text{-}Disj\ z\ w))\ (q\text{-}Disj\ (q\text{-}Disj\ y\ z)\ w)\ \vee$
$x = q\text{-}Imp\ (q\text{-}Disj\ y\ z)\ (q\text{-}Imp\ (q\text{-}Disj\ (q\text{-}Neg\ y)\ w)\ (q\text{-}Disj\ z\ w))$

definition *Sent* :: *hf set* **where**
$Sent \equiv \{x.\ \exists\, y\ z\ w.\ Form\ y \wedge Form\ z \wedge Form\ w \wedge Sent\text{-}axioms\ x\ y\ z\ w\}$

nominal-function *SentP* :: $tm \Rightarrow fm$
where $[\![atom\ y \sharp (z,w,x);\ atom\ z \sharp (w,x);\ atom\ w \sharp x]\!] \Longrightarrow$
$SentP\ x = Ex\ y\ (Ex\ z\ (Ex\ w\ (FormP\ (Var\ y)\ AND\ FormP\ (Var\ z)\ AND$
$FormP\ (Var\ w)\ AND$
$(\ (x\ EQ\ Q\text{-}Imp\ (Var\ y)\ (Var\ y))\ OR$
$(x\ EQ\ Q\text{-}Imp\ (Var\ y)\ (Q\text{-}Disj\ (Var\ y)\ (Var\ z))\ OR$
$(x\ EQ\ Q\text{-}Imp\ (Q\text{-}Disj\ (Var\ y)\ (Var\ y))\ (Var\ y))\ OR$
$(x\ EQ\ Q\text{-}Imp\ (Q\text{-}Disj\ (Var\ y)\ (Q\text{-}Disj\ (Var\ z)\ (Var\ w)))$
$(Q\text{-}Disj\ (Q\text{-}Disj\ (Var\ y)\ (Var\ z))\ (Var\ w)))\ OR$
$(x\ EQ\ Q\text{-}Imp\ (Q\text{-}Disj\ (Var\ y)\ (Var\ z))$
$(Q\text{-}Imp\ (Q\text{-}Disj\ (Q\text{-}Neg\ (Var\ y))\ (Var\ w))\ (Q\text{-}Disj\ (Var$
$z)\ (Var\ w)))))))))$
by (*auto simp*: *eqvt-def SentP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma
 shows *SentP-fresh-iff* [*simp*]: $a \sharp$ *SentP x* $\longleftrightarrow a \sharp x$ (**is** *?thesis1*)
  and *eval-fm-SentP* [*simp*]: *eval-fm e* (*SentP x*) $\longleftrightarrow$ $[\![x]\!]e \in$ *Sent* (**is** *?thesis2*)
  and *SentP-sf* [*iff*]: *Sigma-fm* (*SentP x*) (**is** *?thsf*)
proof −
  obtain *y*::*name* **and** *z*::*name* **and** *w*::*name* **where** *atom y* $\sharp$ (*z*,*w*,*x*) *atom z* $\sharp$ (*w*,*x*) *atom w* $\sharp$ *x*
    by (*metis obtain-fresh*)
  thus *?thesis1 ?thesis2 ?thsf*
    by (*auto simp*: *Sent-def Sent-axioms-def q-defs*)
qed

6.1.2 The predicate *Equality-axP*, for the Equality Axioms

definition *Equality-ax* :: *hf set* **where**
Equality-ax ≡ { $[\![\lceil$*refl-ax*$\rceil]\!]$*e0*, $[\![\lceil$*eq-cong-ax*$\rceil]\!]$*e0*, $[\![\lceil$*mem-cong-ax*$\rceil]\!]$*e0*, $[\![\lceil$*eats-cong-ax*$\rceil]\!]$*e0* }

function *Equality-axP* :: *tm* ⇒ *fm*
  where *Equality-axP x* =
    x EQ ⌈*refl-ax*⌉ *OR x EQ* ⌈*eq-cong-ax*⌉ *OR x EQ* ⌈*mem-cong-ax*⌉ *OR x EQ* ⌈*eats-cong-ax*⌉
by *auto*

termination
  by *lexicographic-order*

lemma *eval-fm-Equality-axP* [*simp*]: *eval-fm e* (*Equality-axP x*) $\longleftrightarrow$ $[\![x]\!]e \in$ *Equality-ax*
  by (*auto simp*: *Equality-ax-def intro*: *eval-quot-fm-ignore*)

6.1.3 The predicate *HF-axP*, for the HF Axioms

definition *HF-ax* :: *hf set* **where**
  HF-ax ≡ {$[\![\lceil$*HF1*$\rceil]\!]$*e0*, $[\![\lceil$*HF2*$\rceil]\!]$*e0*}

function *HF-axP* :: *tm* ⇒ *fm*
  where *HF-axP x* = *x EQ* ⌈*HF1*⌉ *OR x EQ* ⌈*HF2*⌉
by *auto*

termination
  by *lexicographic-order*

lemma *eval-fm-HF-axP* [*simp*]: *eval-fm e* (*HF-axP x*) $\longleftrightarrow$ $[\![x]\!]e \in$ *HF-ax*
  by (*auto simp*: *HF-ax-def intro*: *eval-quot-fm-ignore*)

lemma *HF-axP-sf* [*iff*]: *Sigma-fm* (*HF-axP t*)
  by *auto*

6.1.4 The specialisation axioms

inductive-set *Special-ax* :: *hf set* **where**
  I: $[\![$*AbstForm v 0 x ax*; *SubstForm v y x sx*; *Form x*; *is-Var v*; *Term y*$]\!]$
    $\Longrightarrow$ *q-Imp sx* (*q-Ex ax*) $\in$ *Special-ax*

Defining the syntax

nominal-function *Special-axP* :: *tm* $\Rightarrow$ *fm* **where**
  $[\![$*atom v* $\sharp$ (*p*,*sx*,*y*,*ax*,*x*); *atom x* $\sharp$ (*p*,*sx*,*y*,*ax*);
    atom ax $\sharp$ (*p*,*sx*,*y*); *atom y* $\sharp$ (*p*,*sx*); *atom sx* $\sharp$ *p*$]\!]$ $\Longrightarrow$
  Special-axP p = *Ex v* (*Ex x* (*Ex ax* (*Ex y* (*Ex sx*
        (*FormP* (*Var x*) *AND VarP* (*Var v*) *AND TermP* (*Var y*) *AND*
        AbstFormP (*Var v*) *Zero* (*Var x*) (*Var ax*) *AND*
        SubstFormP (*Var v*) (*Var y*) (*Var x*) (*Var sx*) *AND*
        p EQ Q-Imp (*Var sx*) (*Q-Ex* (*Var ax*)))))))
 by (*auto simp*: *eqvt-def Special-axP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma
 shows *Special-axP-fresh-iff* [*simp*]: *a* $\sharp$ *Special-axP p* $\longleftrightarrow$ *a* $\sharp$ *p* (**is** *?thesis1*)
  and *eval-fm-Special-axP* [*simp*]: *eval-fm e* (*Special-axP p*) $\longleftrightarrow$ $[\![p]\!]e \in$ *Special-ax*
(**is** *?thesis2*)
  and *Special-axP-sf* [*iff*]:   *Sigma-fm* (*Special-axP p*) (**is** *?thesis3*)
proof −
  obtain *v*::*name* **and** *x*::*name* **and** *ax*::*name* **and** *y*::*name* **and** *sx*::*name*
    where *atom v* $\sharp$ (*p*,*sx*,*y*,*ax*,*x*) *atom x* $\sharp$ (*p*,*sx*,*y*,*ax*)
      atom ax $\sharp$ (*p*,*sx*,*y*) *atom y* $\sharp$ (*p*,*sx*) *atom sx* $\sharp$ *p*
    by (*metis obtain-fresh*)
  thus *?thesis1 ?thesis2 ?thesis3*
    apply *auto*
    apply (*metis q-Disj-def q-Ex-def q-Imp-def q-Neg-def Special-ax.intros*)
    apply (*metis q-Disj-def q-Ex-def q-Imp-def q-Neg-def Special-ax.cases*)
    done
qed

Correctness (or, correspondence)

lemma *Special-ax-imp-special-axioms*:
  assumes *x* $\in$ *Special-ax* **shows** $\exists$ *A*. *x* = $[\![\lceil A\rceil]\!]e$ $\wedge$ *A* $\in$ *special-axioms*
using *assms*
proof (*induction rule*: *Special-ax.induct*)
  case (*I v x ax y sx*)
  obtain *fm*::*fm* **and** *u*::*tm* **where** *fm*: *x* = $[\![\lceil fm\rceil]\!]e$ **and** *u*: *y* = $[\![\lceil u\rceil]\!]e$
    using *I* **by** (*auto elim*!: *Form-imp-is-fm Term-imp-is-tm*)
  obtain *B* **where** *x*: *x* = $[\![$*quot-dbfm B*$]\!]e$
      and *ax*: *ax* = $[\![$*quot-dbfm* (*abst-dbfm* (*decode-Var v*) *0 B*)$]\!]e$
    using *I AbstForm-imp-abst-dbfm* **by** *force*

obtain *B'* **where** *x'*: *x* = ⟦*quot-dbfm B'*⟧*e*
and *sx*: *sx* = ⟦*quot-dbfm* (*subst-dbfm* (*trans-tm* [] *u*) (*decode-Var v*) *B'*)⟧*e*
using *I* **by** (*metis u SubstForm-imp-subst-dbfm-lemma quot-tm-def*)
have *eq*: *B'*=*B*
by (*metis quot-dbfm-inject-lemma x x'*)
have *fm*(*decode-Var v*::=*u*) *IMP SyntaxN.Ex* (*decode-Var v*) *fm* ∈ *special-axioms*
by (*metis special-axioms.intros*)
thus *?case* **using** *eq*
apply (*auto simp*: *quot-simps q-defs*
intro!: *exI* [**where** *x* = *fm*((*decode-Var v*)::=*u*) *IMP* (*Ex* (*decode-Var v*) *fm*)])
apply (*metis fm quot-dbfm-inject-lemma quot-fm-def subst-fm-trans-commute sx x'*)
apply (*metis abst-trans-fm ax fm quot-dbfm-inject-lemma quot-fm-def x*)
done
qed

lemma *special-axioms-into-Special-ax*: *A* ∈ *special-axioms* ⟹ ⟦⌈*A*⌉⟧*e* ∈ *Special-ax*
proof (*induct rule*: *special-axioms.induct*)
case (*I A i t*)
have ⟦⌈*A*(*i*::=*t*) *IMP SyntaxN.Ex i A*⌉⟧*e* =
q-Imp ⟦*quot-dbfm* (*subst-dbfm* (*trans-tm* [] *t*) *i* (*trans-fm* [] *A*))⟧*e*
(*q-Ex* ⟦*quot-dbfm* (*trans-fm* [*i*] *A*)⟧*e*)
by (*simp add*: *quot-fm-def q-defs*)
also have ... ∈ *Special-ax*
apply (*rule Special-ax.intros* [*OF AbstForm-trans-fm*])
apply (*auto simp*: *quot-fm-def* [*symmetric*] *intro*: *SubstForm-quot* [*unfolded eval-Var-q*])
done
finally show *?case* **.**
qed

We have precisely captured the codes of the specialisation axioms.

corollary *Special-ax-eq-special-axioms*: *Special-ax* = (⋃ *A* ∈ *special-axioms*. { ⟦⌈*A*⌉⟧*e* })
by (*force dest*: *special-axioms-into-Special-ax Special-ax-imp-special-axioms*)

6.1.5 The induction axioms

inductive-set *Induction-ax* :: *hf set* **where**
I: ⟦*SubstForm v 0 x x0*;
SubstForm v w x xw;
SubstForm v (*q-Eats v w*) *x xevw*;
AbstForm w 0 (*q-Imp x* (*q-Imp xw xevw*)) *allw*;
AbstForm v 0 (*q-All allw*) *allvw*;
AbstForm v 0 x ax;
v ≠ *w*; *VarNonOccForm w x*⟧
⟹ *q-Imp x0* (*q-Imp* (*q-All allvw*) (*q-All ax*)) ∈ *Induction-ax*

Defining the syntax

nominal-function *Induction-axP* :: *tm* ⇒ *fm* **where**
⟦*atom ax* ♯ (*p*,*v*,*w*,*x*,*x0*,*xw*,*xevw*,*allw*,*allvw*);
atom allvw ♯ (*p*,*v*,*w*,*x*,*x0*,*xw*,*xevw*,*allw*); *atom allw* ♯ (*p*,*v*,*w*,*x*,*x0*,*xw*,*xevw*);
atom xevw ♯ (*p*,*v*,*w*,*x*,*x0*,*xw*); *atom xw* ♯ (*p*,*v*,*w*,*x*,*x0*);
atom x0 ♯ (*p*,*v*,*w*,*x*); *atom x* ♯ (*p*,*v*,*w*);
atom w ♯ (*p*,*v*); *atom v* ♯ *p*⟧ ⟹
Induction-axP p = *Ex v* (*Ex w* (*Ex x* (*Ex x0* (*Ex xw* (*Ex xevw* (*Ex allw* (*Ex allvw* (*Ex ax*
((*Var v NEQ Var w*) *AND VarNonOccFormP* (*Var w*) (*Var x*) *AND*
SubstFormP (*Var v*) *Zero* (*Var x*) (*Var x0*) *AND*
SubstFormP (*Var v*) (*Var w*) (*Var x*) (*Var xw*) *AND*
SubstFormP (*Var v*) (*Q-Eats* (*Var v*) (*Var w*)) (*Var x*) (*Var xevw*) *AND*
AbstFormP (*Var w*) *Zero* (*Q-Imp* (*Var x*) (*Q-Imp* (*Var xw*) (*Var xevw*))) (*Var allw*) *AND*
AbstFormP (*Var v*) *Zero* (*Q-All* (*Var allw*)) (*Var allvw*) *AND*
AbstFormP (*Var v*) *Zero* (*Var x*) (*Var ax*) *AND*
p EQ Q-Imp (*Var x0*) (*Q-Imp* (*Q-All* (*Var allvw*)) (*Q-All* (*Var ax*))))))))))))
by (*auto simp*: *eqvt-def Induction-axP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *Induction-axP-fresh-iff* [*simp*]: *a* ♯ *Induction-axP p* ⟷ *a* ♯ *p* (**is** *?thesis1*)
and *eval-fm-Induction-axP* [*simp*]:
eval-fm e (*Induction-axP p*) ⟷ ⟦*p*⟧*e* ∈ *Induction-ax* (**is** *?thesis2*)
and *Induction-axP-sf* [*iff*]: *Sigma-fm* (*Induction-axP p*) (**is** *?thesis3*)
proof −
obtain *v*::*name* **and** *w*::*name* **and** *x*::*name* **and** *x0*::*name* **and** *xw*::*name* **and** *xevw*::*name*
and *allw*::*name* **and** *allvw*::*name* **and** *ax*::*name*
where *atoms*: *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*
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis3*
by *auto*
show *?thesis2*
proof
assume *eval-fm e* (*Induction-axP p*)
thus ⟦*p*⟧*e* ∈ *Induction-ax* **using** *atoms*
by (*auto intro*!: *Induction-ax.I* [*unfolded q-defs*])
next
assume ⟦*p*⟧*e* ∈ *Induction-ax*

thus *eval-fm e* (*Induction-axP p*)
apply (*rule Induction-ax.cases*) **using** *atoms*
apply (*force simp*: *q-defs htuple-minus-1 intro*!: *AbstForm-imp-Ord*)
done
qed
qed

Correctness (or, correspondence)

lemma *Induction-ax-imp-induction-axioms*:
assumes $x \in$ *Induction-ax* **shows** $\exists A.\ x = [\![\lceil A\rceil]\!]e \wedge A \in$ *induction-axioms*
using *assms*
proof (*induction rule*: *Induction-ax.induct*)
case (*I v x x0 w xw xevw allw allvw ax*)
then have *v*: *is-Var v* **and** *w*: *is-Var w*
and *dvw* [*simp*]: *decode-Var v* $\neq$ *decode-Var w* *atom* (*decode-Var w*) $\sharp$ [*decode-Var v*]
by (*auto simp*: *AbstForm-def fresh-Cons*)
obtain *A*::*fm* **where** *A*: $x = [\![\lceil A\rceil]\!]e$ **and** *wfresh*: *atom* (*decode-Var w*) $\sharp$ *A*
using *I VarNonOccForm-imp-fresh* **by** *blast*
then obtain A' A'' **where** A': *q-Imp* ($[\![\lceil A\rceil]\!]e$) (*q-Imp xw xevw*) = $[\![$*quot-dbfm* $A']\!]e$
and A'': *q-All allw* = $[\![$*quot-dbfm* $A'']\!]e$
using *I VarNonOccForm-imp-fresh* **by** (*auto dest*!: *AbstForm-imp-abst-dbfm*)
def *Aw* $\equiv$ *A*(*decode-Var v*::=*Var* (*decode-Var w*))
def *Ae* $\equiv$ *A*(*decode-Var v*::= *Eats* (*Var* (*decode-Var v*)) (*Var* (*decode-Var w*)))
have *x0*: *x0* = $[\![\lceil$*A*(*decode-Var v*::=*Zero*)$\rceil]\!]e$ **using** *I SubstForm-imp-subst-fm*
by (*metis A Form-quot-fm eval-fm-inject eval-tm.simps*(*1*) *quot-Zero*)
have *xw*: *xw* = $[\![\lceil Aw\rceil]\!]e$ **using** *I SubstForm-imp-subst-fm*
by (*metis A Form-quot-fm eval-fm-inject is-Var-imp-decode-Var w Aw-def*)
have *SubstForm v* ($[\![\lceil$*Eats* (*Var* (*decode-Var v*)) (*Var* (*decode-Var w*))$\rceil]\!]e$) *x xevw*
using *I* **by** (*simp add*: *quot-simps q-defs*) (*metis is-Var-iff v w*)
hence *xevw*: *xevw* = $[\![\lceil Ae\rceil]\!]e$
by (*metis A Ae-def Form-quot-fm SubstForm-imp-subst-fm eval-fm-inject*)
have *ax*: *ax* = $[\![$*quot-dbfm* (*abst-dbfm* (*decode-Var v*) *0* (*trans-fm* [] *A*))$]\!]e$
using *I* **by** (*metis A AbstForm-imp-abst-dbfm nat-of-ord-0 quot-dbfm-inject-lemma quot-fm-def*)
have *evw*: *q-Imp x* (*q-Imp xw xevw*) =
$[\![$*quot-dbfm* (*trans-fm* [] (*A IMP* (*Aw IMP Ae*)))$]\!]e$
using *A xw xevw* **by** (*auto simp*: *quot-simps q-defs quot-fm-def*)
hence *allw*: *allw* = $[\![$*quot-dbfm* (*abst-dbfm* (*decode-Var w*) *0*
(*trans-fm* [] (*A IMP* (*Aw IMP Ae*))))$]\!]e$
using *I* **by** (*metis AbstForm-imp-abst-dbfm nat-of-ord-0 quot-dbfm-inject-lemma*)
then have *evw*: *q-All allw* = $[\![$*quot-dbfm* (*trans-fm* [] (*All* (*decode-Var w*) (*A IMP* (*Aw IMP Ae*))))$]\!]e$
by (*auto simp*: *q-defs abst-trans-fm*)
hence *allvw*: *allvw* = $[\![$*quot-dbfm* (*abst-dbfm* (*decode-Var v*) *0*
(*trans-fm* [] (*All* (*decode-Var w*) (*A IMP* (*Aw IMP Ae*)))))$]\!]e$

using *I* **by** (*metis AbstForm-imp-abst-dbfm nat-of-ord-0 quot-dbfm-inject-lemma*)
def *ind-ax* ≡
A(*decode-Var v*::=*Zero*) *IMP*
((*All* (*decode-Var v*) (*All* (*decode-Var w*) (*A IMP* (*Aw IMP Ae*)))) *IMP*
(*All* (*decode-Var v*) *A*))
have *atom* (*decode-Var w*) ♯ (*decode-Var v*, *A*) **using** *I wfresh v w*
by (*metis atom-eq-iff decode-Var-inject fresh-Pair fresh-ineq-at-base*)
hence *ind-ax* ∈ *induction-axioms*
by (*auto simp*: *ind-ax-def Aw-def Ae-def induction-axioms.intros*)
thus *?case*
by (*force simp*: *quot-simps q-defs ind-ax-def allvw ax x0 abst-trans-fm2 abst-trans-fm*)
qed

lemma *induction-axioms-into-Induction-ax*:
A ∈ *induction-axioms* ⟹ ⟦⌈*A*⌉⟧*e* ∈ *Induction-ax*
proof (*induct rule*: *induction-axioms.induct*)
case (*ind j i A*)
hence *eq*: ⟦⌈*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*⌉⟧*e* =
q-Imp ⟦*quot-dbfm* (*subst-dbfm* (*trans-tm* [] *Zero*) *i* (*trans-fm* [] *A*))⟧*e*
(*q-Imp* (*q-All* (*q-All*
(*q-Imp* ⟦*quot-dbfm* (*trans-fm* [*j*, *i*] *A*)⟧*e*
(*q-Imp*
⟦*quot-dbfm* (*trans-fm* [*j*, *i*] (*A*(*i*::=*Var j*)))⟧*e*
⟦*quot-dbfm* (*trans-fm* [*j*, *i*] (*A*(*i*::=*Eats* (*Var i*) (*Var j*))))⟧*e*))))
(*q-All* ⟦*quot-dbfm* (*trans-fm* [*i*] *A*)⟧*e*))
by (*simp add*: *quot-simps q-defs quot-subst-eq fresh-Cons fresh-Pair*)
have [*simp*]: *atom j* ♯ [*i*] **using** *ind*
by (*metis fresh-Cons fresh-Nil fresh-Pair*)
show *?case*
proof (*simp only*: *eq*, *rule Induction-ax.intros* [**where** *v* = *q-Var i* **and** *w* = *q-Var j*])
show *SubstForm* (*q-Var i*) *0* ⟦⌈*A*⌉⟧*e*
⟦*quot-dbfm* (*subst-dbfm* (*trans-tm* [] *Zero*) *i* (*trans-fm* [] *A*))⟧*e*
by (*metis SubstForm-subst-dbfm-eq Term-quot-tm eval-tm.simps*(*1*) *quot-Zero quot-fm-def quot-tm-def*)
next
show *SubstForm* (*q-Var i*) (*q-Var j*) ⟦⌈*A*⌉⟧*e* ⟦*quot-dbfm* (*subst-dbfm* (*DBVar j*) *i* (*trans-fm* [] *A*))⟧*e*
by (*auto simp*: *quot-fm-def intro*!: *SubstForm-subst-dbfm-eq Term-Var*)
(*metis q-Var-def*)
next
show *SubstForm* (*q-Var i*) (*q-Eats* (*q-Var i*) (*q-Var j*)) ⟦⌈*A*⌉⟧*e*
⟦*quot-dbfm* (*subst-dbfm* (*DBEats* (*DBVar i*) (*DBVar j*)) *i* (*trans-fm* [] *A*))⟧*e*
unfolding *quot-fm-def*
by (*auto intro*!: *SubstForm-subst-dbfm-eq Term-Eats Term-Var*) (*simp add*: *q-defs*)

next
  show *AbstForm* (*q-Var j*) *0*
    (*q-Imp* ⟦⌈*A*⌉⟧*e*
      (*q-Imp* ⟦*quot-dbfm* (*subst-dbfm* (*DBVar j*) *i* (*trans-fm* [] *A*))⟧*e*
        ⟦*quot-dbfm* (*subst-dbfm* (*DBEats* (*DBVar i*) (*DBVar j*)) *i* (*trans-fm* [] *A*))⟧*e*))
    ⟦*quot-dbfm* (*trans-fm* [*j*] (*A IMP* (*A*(*i*::= *Var j*) *IMP A*(*i*::= *Eats*(*Var i*)(*Var j*)))))⟧*e*
    by (*rule AbstForm-trans-fm-eq* [**where** *A* = (*A IMP A*(*i*::= *Var j*) *IMP A*(*i*::= *Eats*(*Var i*)(*Var j*)))])
      (*auto simp*: *quot-simps q-defs quot-fm-def subst-fm-trans-commute-eq*)
next
  show *AbstForm* (*q-Var i*) *0*
   (*q-All* ⟦*quot-dbfm* (*trans-fm* [*j*] (*A IMP A*(*i*::=*Var j*) *IMP A*(*i*::=*Eats* (*Var i*) (*Var j*))))⟧*e*)
   (*q-All*
    (*q-Imp* ⟦*quot-dbfm* (*trans-fm* [*j*, *i*] *A*)⟧*e*
     (*q-Imp* ⟦*quot-dbfm* (*trans-fm* [*j*, *i*] (*A*(*i*::=*Var j*)))⟧*e*
       ⟦*quot-dbfm* (*trans-fm* [*j*, *i*] (*A*(*i*::=*Eats* (*Var i*) (*Var j*))))⟧*e*)))
   apply (*rule AbstForm-trans-fm-eq*
      [**where** *A* = *All j* (*A IMP* (*A*(*i*::= *Var j*) *IMP A*(*i*::= *Eats*(*Var i*)(*Var j*))))])
   apply (*auto simp*: *q-defs quot-fm-def*)
   done
next
  show *AbstForm* (*q-Var i*) *0* (⟦⌈*A*⌉⟧*e*) ⟦*quot-dbfm* (*trans-fm* [*i*] *A*)⟧*e*
   by (*metis AbstForm-trans-fm*)
next
  show *q-Var i* ≠ *q-Var j* **using** *ind*
   by (*simp add*: *q-Var-def*)
next
  show *VarNonOccForm* (*q-Var j*) (⟦⌈*A*⌉⟧*e*)
   by (*metis fresh-Pair fresh-imp-VarNonOccForm ind*)
qed
qed

We have captured the codes of the induction axioms.

corollary *Induction-ax-eq-induction-axioms*:
Induction-ax = (⋃*A* ∈ *induction-axioms*. {⟦⌈*A*⌉⟧*e*})
by (*force dest*: *induction-axioms-into-Induction-ax Induction-ax-imp-induction-axioms*)

6.1.6 The predicate *AxiomP*, for any Axioms

definition *Extra-ax* :: *hf set* **where**
Extra-ax ≡ {⟦⌈*extra-axiom*⌉⟧*e0*}

definition *Axiom* :: *hf set* **where**
Axiom ≡ *Extra-ax* ∪ *Sent* ∪ *Equality-ax* ∪ *HF-ax* ∪ *Special-ax* ∪ *Induction-ax*

definition *AxiomP* :: *tm* ⇒ *fm*

where *AxiomP x* ≡ *x EQ* ⌈*extra-axiom*⌉ *OR SentP x OR Equality-axP x OR HF-axP x OR Special-axP x OR Induction-axP x*

lemma *AxiomP-eqvt* [*eqvt*]: (*p* • *AxiomP x*) = *AxiomP* (*p* • *x*)
by (*simp add*: *AxiomP-def*)

lemma *AxiomP-fresh-iff* [*simp*]: *a* ♯ *AxiomP x* ⟷ *a* ♯ *x*
by (*auto simp*: *AxiomP-def*)

lemma *eval-fm-AxiomP* [*simp*]: *eval-fm e* (*AxiomP x*) ⟷ ⟦*x*⟧*e* ∈ *Axiom*
unfolding *AxiomP-def Axiom-def Extra-ax-def*
by (*auto simp del*: *Equality-axP.simps HF-axP.simps intro*: *eval-quot-fm-ignore*)

lemma *AxiomP-sf* [*iff*]: *Sigma-fm* (*AxiomP t*)
by (*auto simp*: *AxiomP-def*)

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

definition *ModPon* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool* **where**
ModPon x y z ≡ (*y* = *q-Imp x z*)

definition *ModPonP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where *ModPonP x y z* = (*y EQ Q-Imp x z*)

lemma *ModPonP-eqvt* [*eqvt*]: (*p* • *ModPonP x y z*) = *ModPonP* (*p* • *x*) (*p* • *y*) (*p* • *z*)
by (*simp add*: *ModPonP-def*)

lemma *ModPonP-fresh-iff* [*simp*]: *a* ♯ *ModPonP x y z* ⟷ *a* ♯ *x* ∧ *a* ♯ *y* ∧ *a* ♯ *z*
by (*auto simp*: *ModPonP-def*)

lemma *eval-fm-ModPonP* [*simp*]: *eval-fm e* (*ModPonP x y z*) ⟷ *ModPon* ⟦*x*⟧*e* ⟦*y*⟧*e* ⟦*z*⟧*e*
by (*auto simp*: *ModPon-def ModPonP-def q-defs*)

lemma *ModPonP-sf* [*iff*]: *Sigma-fm* (*ModPonP t u v*)
by (*auto simp*: *ModPonP-def*)

lemma *ModPonP-subst* [*simp*]:
(*ModPonP t u v*)(*i*::=*w*) = *ModPonP* (*subst i w t*) (*subst i w u*) (*subst i w v*)
by (*auto simp*: *ModPonP-def*)

6.1.8 The predicate *ExistsP*, for the existential rule

Definition

definition *Exists* :: *hf* ⇒ *hf* ⇒ *bool* **where**
Exists p q ≡ (∃ *x x' y v*. *Form x* ∧ *VarNonOccForm v y* ∧ *AbstForm v 0 x x'* ∧ *p* = *q-Imp x y* ∧ *q* = *q-Imp* (*q-Ex x'*) *y*)

nominal-function *ExistsP* :: *tm* ⇒ *tm* ⇒ *fm* **where**
⟦*atom x* ♯ (*p*,*q*,*v*,*y*,*x'*); *atom x'* ♯ (*p*,*q*,*v*,*y*);
atom y ♯ (*p*,*q*,*v*); *atom v* ♯ (*p*,*q*)⟧ ⟹
ExistsP p q = *Ex x* (*Ex x'* (*Ex y* (*Ex v* (*FormP* (*Var x*) *AND*
VarNonOccFormP (*Var v*) (*Var y*) *AND*
AbstFormP (*Var v*) *Zero* (*Var x*) (*Var x'*) *AND*
p EQ Q-Imp (*Var x*) (*Var y*) *AND*
q EQ Q-Imp (*Q-Ex* (*Var x'*)) (*Var y*)))))
by (*auto simp*: *eqvt-def ExistsP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *ExistsP-fresh-iff* [*simp*]: *a* ♯ *ExistsP p q* ⟷ *a* ♯ *p* ∧ *a* ♯ *q* (**is** *?thesis1*)
and *eval-fm-ExistsP* [*simp*]: *eval-fm e* (*ExistsP p q*) ⟷ *Exists* ⟦*p*⟧*e* ⟦*q*⟧*e* (**is** *?thesis2*)
and *ExistsP-sf* [*iff*]: *Sigma-fm* (*ExistsP p q*) (**is** *?thesis3*)
proof −
obtain *x*::*name* **and** *x'*::*name* **and** *y*::*name* **and** *v*::*name*
where *atom x* ♯ (*p*,*q*,*v*,*y*,*x'*) *atom x'* ♯ (*p*,*q*,*v*,*y*) *atom y* ♯ (*p*,*q*,*v*) *atom v* ♯ (*p*,*q*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thesis3*
by (*auto simp*: *Exists-def q-defs*)
qed

lemma *ExistsP-subst* [*simp*]: (*ExistsP p q*)(*j*::=*w*) = *ExistsP* (*subst j w p*) (*subst j w q*)
proof −
obtain *x*::*name* **and** *x'*::*name* **and** *y*::*name* **and** *v*::*name*
where *atom x* ♯ (*j*,*w*,*p*,*q*,*v*,*y*,*x'*) *atom x'* ♯ (*j*,*w*,*p*,*q*,*v*,*y*)
atom y ♯ (*j*,*w*,*p*,*q*,*v*) *atom v* ♯ (*j*,*w*,*p*,*q*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *ExistsP.simps* [*of x - - x' y v*])
qed

Correctness

lemma *Exists-imp-exists*:
assumes *Exists p q*
shows ∃ *A B i*. *p* = ⟦⌈*A IMP B*⌉⟧*e* ∧ *q* = ⟦⌈(*Ex i A*) *IMP B*⌉⟧*e* ∧ *atom i* ♯ *B*
proof −
obtain *x ax y v*
where *x*: *Form x*
and *noc*: *VarNonOccForm v y*
and *abst*: *AbstForm v 0 x ax*

and *p*: $p = q\text{-}Imp\ x\ y$
and *q*: $q = q\text{-}Imp\ (q\text{-}Ex\ ax)\ y$
using *assms* **by** (*auto simp*: *Exists-def*)
then obtain *B*::*fm* **where** *B*: $y = [\![\lceil B \rceil]\!]e$ **and** *vfresh*: *atom* (*decode-Var v*) $\sharp$ *B*
by (*metis VarNonOccForm-imp-fresh*)
obtain *A*::*fm* **where** *A*: $x = [\![\lceil A \rceil]\!]e$
by (*metis Form-imp-is-fm x*)
with *AbstForm-imp-abst-dbfm* [*OF abst*, *of e*]
have *ax*: $ax = [\![quot\text{-}dbfm\ (abst\text{-}dbfm\ (decode\text{-}Var\ v)\ 0\ (trans\text{-}fm\ []\ A))]\!]e$
$p = [\![\lceil A\ IMP\ B \rceil]\!]e$ **using** *p A B*
by (*auto simp*: *quot-simps quot-fm-def q-defs*)
have $q = [\![\lceil (Ex\ (decode\text{-}Var\ v)\ A)\ IMP\ B \rceil]\!]e$ **using** *q A B ax*
by (*auto simp*: *abst-trans-fm quot-simps q-defs*)
then show *?thesis* **using** *vfresh ax*
by *blast*
qed

lemma *Exists-intro*: *atom i* $\sharp$ *B* $\Longrightarrow$ *Exists* ($[\![\lceil A\ IMP\ B \rceil]\!]e$) $[\![\lceil (Ex\ i\ A)\ IMP\ B \rceil]\!]e$
by (*simp add*: *Exists-def quot-simps q-defs*)
(*metis AbstForm-trans-fm Form-quot-fm fresh-imp-VarNonOccForm*)

Thus, we have precisely captured the codes of the specialisation axioms.

corollary *Exists-iff-exists*:
Exists p q $\longleftrightarrow$ ($\exists$ *A B i*. $p = [\![\lceil A\ IMP\ B \rceil]\!]e \wedge q = [\![\lceil (Ex\ i\ A)\ IMP\ B \rceil]\!]e \wedge$ *atom i* $\sharp$ *B*)
by (*force dest*: *Exists-imp-exists Exists-intro*)

6.1.9 The predicate *SubstP*, for the substitution rule

Although the substitution rule is derivable in the calculus, the derivation is too complicated to reproduce within the proof function. It is much easier to provide it as an immediate inference step, justifying its soundness in terms of other inference rules.

Definition

This is the inference $H \vdash A \Longrightarrow H \vdash A\ (i{::=}x)$

definition *Subst* :: *hf* ⇒ *hf* ⇒ *bool* **where**
Subst p q ≡ ($\exists$ *v u*. *SubstForm v u p q*)

nominal-function *SubstP* :: *tm* ⇒ *tm* ⇒ *fm* **where**
⟦*atom u* $\sharp$ (*p*,*q*,*v*); *atom v* $\sharp$ (*p*,*q*)⟧ $\Longrightarrow$
SubstP p q = *Ex v* (*Ex u* (*SubstFormP* (*Var v*) (*Var u*) *p q*))
by (*auto simp*: *eqvt-def SubstP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *SubstP-fresh-iff* [*simp*]: $a \sharp SubstP\ p\ q \longleftrightarrow a \sharp p \wedge a \sharp q$ (**is** *?thesis1*)
and *eval-fm-SubstP* [*simp*]: *eval-fm e* (*SubstP p q*) $\longleftrightarrow$ *Subst* $[\![p]\!]e$ $[\![q]\!]e$ (**is** *?thesis2*)
and *SubstP-sf* [*iff*]: *Sigma-fm* (*SubstP p q*) (**is** *?thesis3*)
proof −
obtain *u*::*name* **and** *v*::*name* **where** *atom u* $\sharp$ (*p*,*q*,*v*) *atom v* $\sharp$ (*p*,*q*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thesis3*
by (*auto simp*: *Subst-def q-defs*)
qed

lemma *SubstP-subst* [*simp*]: (*SubstP p q*)(*j*::=*w*) = *SubstP* (*subst j w p*) (*subst j w q*)
proof −
obtain *u*::*name* **and** *v*::*name* **where** *atom u* $\sharp$ (*j*,*w*,*p*,*q*,*v*) *atom v* $\sharp$ (*j*,*w*,*p*,*q*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*simp add*: *SubstP.simps* [*of u - - v*])
qed

Correctness

lemma *Subst-imp-subst*:
assumes *Subst p q Form p*
shows $\exists$ *A*::*fm*. $\exists$ *i t*. $p = [\![\lceil A\rceil]\!]e \wedge q = [\![\lceil A(i::=t)\rceil]\!]e$
proof −
obtain *v u* **where** *subst*: *SubstForm v u p q* **using** *assms*
by (*auto simp*: *Subst-def*)
then obtain *t*::*tm* **where** *substt*: *SubstForm v* $[\![\lceil t\rceil]\!]e$ *p q*
by (*metis SubstForm-def Term-imp-is-tm*)
with *SubstForm-imp-subst-fm* [*OF substt*] *assms*
obtain *A* **where** $p = [\![\lceil A\rceil]\!]e$ $q = [\![\lceil A(decode\text{-}Var\ v::=t)\rceil]\!]e$
by *auto*
thus *?thesis*
by *blast*
qed

6.1.10 The predicate *PrfP*

definition *Prf* :: *hf* $\Rightarrow$ *hf* $\Rightarrow$ *hf* $\Rightarrow$ *bool*
where *Prf s k y* $\equiv$ *BuildSeq* ($\lambda x.\ x \in Axiom$) ($\lambda u\ v\ w.$ *ModPon v w u* $\vee$ *Exists v u* $\vee$ *Subst v u*) *s k y*

nominal-function *PrfP* :: *tm* $\Rightarrow$ *tm* $\Rightarrow$ *tm* $\Rightarrow$ *fm*
where $[\![$*atom l* $\sharp$ (*s*,*sl*,*m*,*n*,*sm*,*sn*); *atom sl* $\sharp$ (*s*,*m*,*n*,*sm*,*sn*);
atom m $\sharp$ (*s*,*n*,*sm*,*sn*); *atom n* $\sharp$ (*s*,*k*,*sm*,*sn*);
atom sm $\sharp$ (*s*,*sn*); *atom sn* $\sharp$ (*s*)$]\!] \Longrightarrow$
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)))))))))
by (*auto simp*: *eqvt-def PrfP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *PrfP-fresh-iff* [*simp*]: $a \sharp PrfP\ s\ k\ t \longleftrightarrow a \sharp s \wedge a \sharp k \wedge a \sharp t$ (**is** *?thesis1*)
and *eval-fm-PrfP* [*simp*]: *eval-fm e* (*PrfP s k t*) $\longleftrightarrow$ *Prf* $[\![s]\!]e$ $[\![k]\!]e$ $[\![t]\!]e$ (**is** *?thesis2*)
and *PrfP-imp-OrdP* [*simp*]: {*PrfP s k t*} $\vdash$ *OrdP k* (**is** *?thord*)
and *PrfP-imp-LstSeqP* [*simp*]: {*PrfP s k t*} $\vdash$ *LstSeqP s k t* (**is** *?thlstseq*)
and *PrfP-sf* [*iff*]: *Sigma-fm* (*PrfP s k t*) (**is** *?thsf*)
proof −
obtain *l*::*name* **and** *sl*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name*
where *atoms*: *atom l* $\sharp$ (*s*,*sl*,*m*,*n*,*sm*,*sn*) *atom sl* $\sharp$ (*s*,*m*,*n*,*sm*,*sn*)
atom m $\sharp$ (*s*,*n*,*sm*,*sn*) *atom n* $\sharp$ (*s*,*k*,*sm*,*sn*)
atom sm $\sharp$ (*s*,*sn*) *atom sn* $\sharp$ (*s*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thord ?thlstseq ?thsf*
by (*auto intro*: *LstSeqP-OrdP*)
show *?thesis2* **using** *atoms*
by *simp*
(*simp cong*: *conj-cong add*: *LstSeq-imp-Ord Prf-def BuildSeq-def Builds-def*
ModPon-def Exists-def HBall-def HBex-def
Seq-iff-app [*OF LstSeq-imp-Seq-succ*]
Ord-trans [*of - - succ* $[\![k]\!]e$])
qed

lemma *PrfP-subst* [*simp*]:
(*PrfP t u v*)(*j*::=*w*) = *PrfP* (*subst j w t*) (*subst j w u*) (*subst j w v*)
proof −
obtain *l*::*name* **and** *sl*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name*
where *atom l* $\sharp$ (*t*,*u*,*v*,*j*,*w*,*sl*,*m*,*n*,*sm*,*sn*) *atom sl* $\sharp$ (*t*,*u*,*v*,*j*,*w*,*m*,*n*,*sm*,*sn*)
atom m $\sharp$ (*t*,*u*,*v*,*j*,*w*,*n*,*sm*,*sn*) *atom n* $\sharp$ (*t*,*u*,*v*,*j*,*w*,*sm*,*sn*)
atom sm $\sharp$ (*t*,*u*,*v*,*j*,*w*,*sn*) *atom sn* $\sharp$ (*t*,*u*,*v*,*j*,*w*)
by (*metis obtain-fresh*)
thus *?thesis*

by (*simp add*: *PrfP.simps* [*of l - sl m n sm sn*])
qed

6.1.11 The predicate *PfP*

definition *Pf* :: *hf* ⇒ *bool*
  where *Pf y* ≡ (∃ *s k*. *Prf s k y*)

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*))
  by (*auto simp*: *eqvt-def PfP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma
 shows *PfP-fresh-iff* [*simp*]: *a* ♯ *PfP y* ⟷ *a* ♯ *y*   (**is** *?thesis1*)
  and *eval-fm-PfP* [*simp*]: *eval-fm e* (*PfP y*) ⟷ *Pf* ⟦*y*⟧*e* (**is** *?thesis2*)
  and *PfP-sf* [*iff*]: *Sigma-fm* (*PfP y*)   (**is** *?thsf*)
proof −
  obtain *k*::*name* **and** *s*::*name* **where** *atom k* ♯ (*s*,*y*) *atom s* ♯ *y*
    by (*metis obtain-fresh*)
  thus *?thesis1 ?thesis2 ?thsf*
    by (*auto simp*: *Pf-def*)
qed

lemma *PfP-subst* [*simp*]: (*PfP t*)(*j*::=*w*) = *PfP* (*subst j w t*)
proof −
  obtain *k*::*name* **and** *s*::*name* **where** *atom k* ♯ (*s*,*t*,*j*,*w*) *atom s* ♯ (*t*,*j*,*w*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *PfP.simps* [*of k s*])
qed

lemma *ground-PfP* [*simp*]: *ground-fm* (*PfP y*) = *ground y*
  by (*simp add*: *ground-aux-def ground-fm-aux-def supp-conv-fresh*)

6.2 Proposition 4.4

6.2.1 Left-to-Right Proof

lemma *extra-axiom-imp-Pf*: *Pf* ⟦⌈*extra-axiom*⌉⟧*e*
proof −
  have ⟦⌈*extra-axiom*⌉⟧*e* ∈ *Extra-ax*
    by (*simp add*: *Extra-ax-def*) (*rule eval-quot-fm-ignore*)
  thus *?thesis*
    by (*force simp add*: *Pf-def Prf-def Axiom-def intro*: *BuildSeq-exI*)
qed

lemma *boolean-axioms-imp-Pf*:
  assumes $\alpha \in$ *boolean-axioms* **shows** *Pf* $[\![\lceil\alpha\rceil]\!]e$
proof −
  have $[\![\lceil\alpha\rceil]\!]e \in$ *Sent* **using** *assms*
    by (*rule boolean-axioms.cases*)
      (*auto simp*: *Sent-def Sent-axioms-def quot-Disj quot-Neg q-defs*)
  thus *?thesis*
    by (*force simp add*: *Pf-def Prf-def Axiom-def intro*: *BuildSeq-exI*)
qed

lemma *equality-axioms-imp-Pf*:
  assumes $\alpha \in$ *equality-axioms* **shows** *Pf* $[\![\lceil\alpha\rceil]\!]e$
proof −
  have $[\![\lceil\alpha\rceil]\!]e \in$ *Equality-ax* **using** *assms* [*unfolded equality-axioms-def*]
    by (*auto simp*: *Equality-ax-def eval-quot-fm-ignore*)
  thus *?thesis*
    by (*force simp add*: *Pf-def Prf-def Axiom-def intro*: *BuildSeq-exI*)
qed

lemma *HF-axioms-imp-Pf*:
  assumes $\alpha \in$ *HF-axioms* **shows** *Pf* $[\![\lceil\alpha\rceil]\!]e$
proof −
  have $[\![\lceil\alpha\rceil]\!]e \in$ *HF-ax* **using** *assms* [*unfolded HF-axioms-def*]
    by (*auto simp*: *HF-ax-def eval-quot-fm-ignore*)
  thus *?thesis*
    by (*force simp add*: *Pf-def Prf-def Axiom-def intro*: *BuildSeq-exI*)
qed

lemma *special-axioms-imp-Pf*:
  assumes $\alpha \in$ *special-axioms* **shows** *Pf* $[\![\lceil\alpha\rceil]\!]e$
proof −
  have $[\![\lceil\alpha\rceil]\!]e \in$ *Special-ax*
    by (*metis special-axioms-into-Special-ax assms*)
  thus *?thesis*
    by (*force simp add*: *Pf-def Prf-def Axiom-def intro*: *BuildSeq-exI*)
qed

lemma *induction-axioms-imp-Pf*:
  assumes $\alpha \in$ *induction-axioms* **shows** *Pf* $[\![\lceil\alpha\rceil]\!]e$
proof −
  have $[\![\lceil\alpha\rceil]\!]e \in$ *Induction-ax*
    by (*metis induction-axioms-into-Induction-ax assms*)
  thus *?thesis*
    by (*force simp add*: *Pf-def Prf-def Axiom-def intro*: *BuildSeq-exI*)
qed

lemma *ModPon-imp-Pf*: $[\![$*Pf* $[\![$*Q-Imp x y*$]\!]e$; *Pf* $[\![x]\!]e]\!] \Longrightarrow$ *Pf* $[\![y]\!]e$
  by (*auto simp*: *Pf-def Prf-def ModPon-def q-defs intro*: *BuildSeq-combine*)

lemma *quot-ModPon-imp-Pf*: $[\![Pf\ [\![\lceil \alpha\ IMP\ \beta \rceil]\!]e;\ Pf\ [\![\lceil \alpha \rceil]\!]e]\!] \Longrightarrow Pf\ [\![\lceil \beta \rceil]\!]e$
  by (*simp add*: *ModPon-imp-Pf quot-fm-def quot-simps q-defs*)

lemma *quot-Exists-imp-Pf*: $[\![Pf\ [\![\lceil \alpha\ IMP\ \beta \rceil]\!]e;\ atom\ i\ \sharp\ \beta]\!] \Longrightarrow Pf\ [\![\lceil Ex\ i\ \alpha\ IMP\ \beta \rceil]\!]e$
  by (*force simp*: *Pf-def Prf-def Exists-def quot-simps q-defs*
      intro: *BuildSeq-combine AbstForm-trans-fm-eq fresh-imp-VarNonOccForm*)

lemma *proved-imp-Pf*: **assumes** $H \vdash \alpha$ $H=\{\}$ **shows** $Pf\ [\![\lceil \alpha \rceil]\!]e$
using *assms*
proof (*induct*)
  case (*Hyp A H*) **thus** *?case*
    by *auto*
next
  case (*Extra H*) **thus** *?case*
    by (*metis extra-axiom-imp-Pf*)
next
  case (*Bool A H*) **thus** *?case*
    by (*metis boolean-axioms-imp-Pf*)
next
  case (*Eq A H*) **thus** *?case*
    by (*metis equality-axioms-imp-Pf*)
next
  case (*HF A H*) **thus** *?case*
    by (*metis HF-axioms-imp-Pf*)
next
  case (*Spec A H*) **thus** *?case*
    by (*metis special-axioms-imp-Pf*)
next
  case (*Ind A H*) **thus** *?case*
    by (*metis induction-axioms-imp-Pf*)
next
  case (*MP H A B H'*) **thus** *?case*
    by (*metis quot-ModPon-imp-Pf Un-empty*)
next
  case (*Exists H A B i*) **thus** *?case*
    by (*metis quot-Exists-imp-Pf*)
qed

corollary *proved-imp-proved-PfP*: $\{\} \vdash \alpha \Longrightarrow \{\} \vdash PfP\ \lceil \alpha \rceil$
  by (*rule Sigma-fm-imp-thm* [*OF PfP-sf*])
    (*auto simp*: *ground-aux-def supp-conv-fresh proved-imp-Pf*)

6.2.2 Right-to-Left Proof

lemma *Sent-imp-hfthm*:
  assumes $x \in Sent$ **shows** $\exists A.\ x = [\![\lceil A \rceil]\!]e \wedge \{\} \vdash A$
proof −

obtain $y\ z\ w$ **where** $Form\ y\ Form\ z\ Form\ w$ **and** axs: $Sent\text{-}axioms\ x\ y\ z\ w$
using $assms$ **by** $(auto\ simp: Sent\text{-}def)$
then obtain $A::fm$ **and** $B::fm$ **and** $C::fm$
where A: $y = [\![\lceil A\rceil]\!]e$ **and** B: $z = [\![\lceil B\rceil]\!]e$ **and** C: $w = [\![\lceil C\rceil]\!]e$
by $(metis\ Form\text{-}imp\text{-}is\text{-}fm)$
have $\exists A.\ q\text{-}Imp\ y\ y = [\![\lceil A\rceil]\!]e \wedge \{\} \vdash A$
by $(force\ simp\ add: A\ quot\text{-}Disj\ quot\text{-}Neg\ q\text{-}defs\ hfthm.Bool\ boolean\text{-}axioms.intros)$
moreover have $\exists A.\ q\text{-}Imp\ y\ (q\text{-}Disj\ y\ z) = [\![\lceil A\rceil]\!]e \wedge \{\} \vdash A$
by $(force\ intro!: exI$ $[$**where** $x{=}A\ IMP\ (A\ OR\ B)]$
$simp\ add: A\ B\ quot\text{-}Disj\ quot\text{-}Neg\ q\text{-}defs\ hfthm.Bool\ boolean\text{-}axioms.intros)$
moreover have $\exists A.\ q\text{-}Imp\ (q\text{-}Disj\ y\ y)\ y = [\![\lceil A\rceil]\!]e \wedge \{\} \vdash A$
by $(force\ intro!: exI$ $[$**where** $x{=}(A\ OR\ A)\ IMP\ A]$
$simp\ add: A\ quot\text{-}Disj\ quot\text{-}Neg\ q\text{-}defs\ hfthm.Bool\ boolean\text{-}axioms.intros)$
moreover have $\exists A.\ q\text{-}Imp\ (q\text{-}Disj\ y\ (q\text{-}Disj\ z\ w))\ (q\text{-}Disj\ (q\text{-}Disj\ y\ z)\ w) = [\![\lceil A\rceil]\!]e \wedge \{\} \vdash A$
by $(force\ intro!: exI$ $[$**where** $x{=}(A\ OR\ (B\ OR\ C))\ IMP\ ((A\ OR\ B)\ OR\ C)]$
$simp\ add: A\ B\ C\ quot\text{-}Disj\ quot\text{-}Neg\ q\text{-}defs\ hfthm.Bool\ boolean\text{-}axioms.intros)$
moreover have $\exists A.\ q\text{-}Imp\ (q\text{-}Disj\ y\ z)\ (q\text{-}Imp\ (q\text{-}Disj\ (q\text{-}Neg\ y)\ w)\ (q\text{-}Disj\ z\ w)) = [\![\lceil A\rceil]\!]e \wedge \{\} \vdash A$
by $(force\ intro!: exI$ $[$**where** $x{=}(A\ OR\ B)\ IMP\ ((Neg\ A\ OR\ C)\ IMP\ (B\ OR\ C))]$
$simp\ add: A\ B\ C\ quot\text{-}Disj\ quot\text{-}Neg\ q\text{-}defs\ hfthm.Bool\ boolean\text{-}axioms.intros)$
ultimately show $?thesis$ **using** axs $[unfolded\ Sent\text{-}axioms\text{-}def]$
by $blast$
qed

lemma $Extra\text{-}ax\text{-}imp\text{-}hfthm$:
assumes $x \in Extra\text{-}ax$ **obtains** A **where** $x = [\![\lceil A\rceil]\!]e \wedge \{\} \vdash A$
using $assms$ **unfolding** $Extra\text{-}ax\text{-}def$
by $(auto\ intro: eval\text{-}quot\text{-}fm\text{-}ignore\ hfthm.Extra)$

lemma $Equality\text{-}ax\text{-}imp\text{-}hfthm$:
assumes $x \in Equality\text{-}ax$ **obtains** A **where** $x = [\![\lceil A\rceil]\!]e \wedge \{\} \vdash A$
using $assms$ **unfolding** $Equality\text{-}ax\text{-}def$
by $(auto\ intro: eval\text{-}quot\text{-}fm\text{-}ignore\ hfthm.Eq\ [unfolded\ equality\text{-}axioms\text{-}def])$

lemma $HF\text{-}ax\text{-}imp\text{-}hfthm$:
assumes $x \in HF\text{-}ax$ **obtains** A **where** $x = [\![\lceil A\rceil]\!]e \wedge \{\} \vdash A$
using $assms$ **unfolding** $HF\text{-}ax\text{-}def$
by $(auto\ intro: eval\text{-}quot\text{-}fm\text{-}ignore\ hfthm.HF\ [unfolded\ HF\text{-}axioms\text{-}def])$

lemma $Special\text{-}ax\text{-}imp\text{-}hfthm$:
assumes $x \in Special\text{-}ax$ **obtains** A **where** $x = [\![\lceil A\rceil]\!]e\ \{\} \vdash A$
by $(metis\ Spec\ Special\text{-}ax\text{-}imp\text{-}special\text{-}axioms\ assms)$

lemma $Induction\text{-}ax\text{-}imp\text{-}hfthm$:
assumes $x \in Induction\text{-}ax$ **obtains** A **where** $x = [\![\lceil A\rceil]\!]e\ \{\} \vdash A$
by $(metis\ Induction\text{-}ax\text{-}imp\text{-}induction\text{-}axioms\ assms\ hfthm.Ind)$

lemma *Exists-imp-hfthm*: $[\![Exists\ [\![\lceil A\rceil]\!]e\ y;\ \{\} \vdash A]\!] \Longrightarrow \exists B.\ y = [\![\lceil B\rceil]\!]e \wedge \{\} \vdash B$
by (*drule Exists-imp-exists* [**where** *e*=*e*]) (*auto intro*: *anti-deduction*)

lemma *Subst-imp-hfthm*: $[\![Subst\ [\![\lceil A\rceil]\!]e\ y;\ \{\} \vdash A]\!] \Longrightarrow \exists B.\ y = [\![\lceil B\rceil]\!]e \wedge \{\} \vdash B$
by (*drule Subst-imp-subst* [**where** *e*=*e*], *auto intro*: *Subst*)

lemma *eval-Neg-imp-Neg*: $[\![\lceil \alpha\rceil]\!]e = q\text{-}Neg\ x \Longrightarrow \exists A.\ \alpha = Neg\ A \wedge [\![\lceil A\rceil]\!]e = x$
by (*cases* α *rule*: *fm.exhaust*) (*auto simp*: *quot-simps q-defs htuple-minus-1*)

lemma *eval-Disj-imp-Disj*: $[\![\lceil \alpha\rceil]\!]e = q\text{-}Disj\ x\ y \Longrightarrow \exists A\ B.\ \alpha = A\ OR\ B \wedge [\![\lceil A\rceil]\!]e = x \wedge [\![\lceil B\rceil]\!]e = y$
by (*cases* α *rule*: *fm.exhaust*) (*auto simp*: *quot-simps q-defs htuple-minus-1*)

lemma *Prf-imp-proved*: **assumes** *Prf s k x* **shows** $\exists A.\ x = [\![\lceil A\rceil]\!]e \wedge \{\} \vdash A$
using *assms* [*unfolded Prf-def Axiom-def*]
proof (*induction x rule*: *BuildSeq-induct*)
  case (*B x*) **thus** *?case*
   by (*auto intro*: *Extra-ax-imp-hfthm Sent-imp-hfthm Equality-ax-imp-hfthm HF-ax-imp-hfthm Special-ax-imp-hfthm Induction-ax-imp-hfthm*)
next
  case (*C x y z*)
  then obtain *A*::*fm* **and** *B*::*fm* **where** $y = [\![\lceil A\rceil]\!]e$ $\{\} \vdash A$ $z = [\![\lceil B\rceil]\!]e$ $\{\} \vdash B$
   by *blast*
  thus *?case* **using** *C.hyps ModPon-def q-Imp-def*
   by (*auto dest*!: *MP-same eval-Neg-imp-Neg eval-Disj-imp-Disj Exists-imp-hfthm Subst-imp-hfthm*)
qed

corollary *Pf-quot-imp-is-proved*: $Pf\ [\![\lceil \alpha\rceil]\!]e \Longrightarrow \{\} \vdash \alpha$
by (*metis Pf-def Prf-imp-proved eval-fm-inject*)

Proposition 4.4!

theorem *proved-iff-proved-PfP*: $\{\} \vdash \alpha \longleftrightarrow \{\} \vdash PfP\ \lceil \alpha\rceil$
by (*metis Pf-quot-imp-is-proved emptyE eval-fm-PfP hfthm-sound proved-imp-proved-PfP*)

end

Chapter 7

Uniqueness Results: Syntactic Relations are Functions

theory *Functions*
imports *Coding-Predicates*
begin

7.0.3 SeqStTermP

lemma *not-IndP-VarP*: $\{IndP\ x,\ VarP\ x\} \vdash A$
proof −
  obtain *m*::*name* **where** *atom m* ♯ (*x*,*A*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *fresh-Pair*) (*blast intro*: *ExFalso cut-same* [*OF VarP-cong* [*THEN Iff-MP-same*]])
qed

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 −
  obtain *m*::*name* **where** *atom m* ♯ (*x*,*A*)
    by (*metis obtain-fresh*)
  hence { *IndP* (*HPair* (*HPair Zero* (*HPair Zero Zero*)) *x*) } $\vdash A$
    by (*auto simp*: *IndP.simps* [*of m*] *HTuple-minus-1 intro*: *thin1*)
  thus *?thesis*
    by (*metis Assume cut1*)
qed

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 −
  have { *IndP x OR x NEQ v*, *x EQ HPair* (*HPair Zero* (*HPair Zero Zero*)) *y* }
⊢ *A*
    by (*auto intro*!: *OrdNotEqP-OrdP-E IndP-HPairE*
            intro: *cut-same* [*OF IndP-cong* [*THEN Iff-MP-same*]]
                  cut-same [*OF OrdP-cong* [*THEN Iff-MP-same*]])
  thus *?thesis*
    by (*metis Assume assms rcut2*)
qed

lemma *SeqStTermP-lemma*:
 assumes *atom m* ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*n*,*sm*,*sm′*,*sn*,*sn′*) *atom n* ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*sm*,*sm′*,*sn*,*sn′*)
         atom sm ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*sm′*,*sn*,*sn′*) *atom sm′* ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*sn*,*sn′*)
         atom sn ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*sn′*) *atom sn′* ♯ (*v*,*i*,*t*,*u*,*s*,*k*)
    shows { *SeqStTermP v i t u s k* }
        ⊢ ((*t EQ v AND u EQ i*) *OR*
          ((*IndP t OR t NEQ v*) *AND u EQ t*)) *OR*
          Ex m (*Ex n* (*Ex sm* (*Ex sm′* (*Ex sn* (*Ex sn′* (*Var m IN k AND Var n*
IN k AND
                    SeqStTermP v i (*Var sm*) (*Var sm′*) *s* (*Var m*) *AND*
                    SeqStTermP v i (*Var sn*) (*Var sn′*) *s* (*Var n*) *AND*
                    t EQ Q-Eats (*Var sm*) (*Var sn*) *AND*
                    u EQ Q-Eats (*Var sm′*) (*Var sn′*)))))))
proof −
  obtain *l*::*name* **and** *sl*::*name* **and** *sl′*::*name*
    where *atom l* ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
        atom sl ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
        atom sl′ ♯ (*v*,*i*,*t*,*u*,*s*,*k*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
    by (*metis obtain-fresh*)
  thus *?thesis* **using** *assms*
    apply (*simp add*: *SeqStTermP.simps* [*of l s k v i sl sl′ m n sm sm′ sn sn′*])
    apply (*rule Conj-EH Ex-EH All2-SUCC-E* [*THEN rotate2*] | *simp*)+
    apply (*rule cut-same* [**where** *A* = *HPair t u EQ HPair* (*Var sl*) (*Var sl′*)])
    apply (*metis Assume AssumeH*(*4*) *LstSeqP-EQ*)
    apply *clarify*
    apply (*rule Disj-EH*)
    apply (*rule Disj-I1*)
    apply (*rule anti-deduction*)
    apply (*rule Var-Eq-subst-Iff* [*THEN Sym-L*, *THEN Iff-MP-same*])
    apply (*rule Sym-L* [*THEN rotate2*])
    apply (*rule Var-Eq-subst-Iff* [*THEN Iff-MP-same*], *force*)
    — now the quantified case
    — auto could be used but is VERY SLOW
    apply (*rule Ex-EH Conj-EH*)+
    apply *simp-all*
    apply (*rule Disj-I2*)
    apply (*rule Ex-I* [**where** *x* = *Var m*], *simp*)
    apply (*rule Ex-I* [**where** *x* = *Var n*], *simp*)
    apply (*rule Ex-I* [**where** *x* = *Var sm*], *simp*)

apply (*rule Ex-I* [**where** x = *Var sm'*], *simp*)
apply (*rule Ex-I* [**where** x = *Var sn*], *simp*)
apply (*rule Ex-I* [**where** x = *Var sn'*], *simp*)
apply (*simp-all add*: *SeqStTermP.simps* [*of l s - v i sl sl' m n sm sm' sn sn'*])
apply ((*rule Conj-I*)+, *blast intro*: *LstSeqP-Mem*)+
— first SeqStTermP subgoal
apply (*rule All2-Subset* [*OF Hyp*], *blast*)
apply (*blast intro*!: *SUCC-Subset-Ord LstSeqP-OrdP*, *blast*, *simp*)
— next SeqStTermP subgoal
apply ((*rule Conj-I*)+, *blast intro*: *LstSeqP-Mem*)+
apply (*rule All2-Subset* [*OF Hyp*], *blast*)
apply (*blast intro*!: *SUCC-Subset-Ord LstSeqP-OrdP*, *blast*, *simp*)
— finally, the equality pair
apply (*blast intro*: *Trans*)
done
qed

lemma *SeqStTermP-unique*: $\{$*SeqStTermP v a t u s kk*, *SeqStTermP v a t u' s' kk'*$\}$ $\vdash$ *u' EQ u*
proof −
obtain *i*::*name* **and** *j*::*name* **and** *j'*::*name* **and** *k*::*name* **and** *k'*::*name* **and** *l*::*name*
and *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name* **and** *sm'*::*name* **and** *sn'*::*name*
and *m2*::*name* **and** *n2*::*name* **and** *sm2*::*name* **and** *sn2*::*name* **and** *sm2'*::*name* **and** *sn2'*::*name*
where *atoms*: *atom i* $\sharp$ (s,s',v,a,t,u,u') *atom j* $\sharp$ (s,s',v,a,t,i,t,u,u')
atom j' $\sharp$ (s,s',v,a,t,i,j,t,u,u')
atom k $\sharp$ $(s,s',v,a,t,u,u',kk',i,j,j')$ *atom k'* $\sharp$ $(s,s',v,a,t,u,u',k,i,j,j')$
atom l $\sharp$ (s,s',v,a,t,i,j,j',k,k')
atom m $\sharp$ (s,s',v,a,i,j,j',k,k',l) *atom n* $\sharp$ $(s,s',v,a,i,j,j',k,k',l,m)$
atom sm $\sharp$ $(s,s',v,a,i,j,j',k,k',l,m,n)$ *atom sn* $\sharp$ $(s,s',v,a,i,j,j',k,k',l,m,n,sm)$
atom sm' $\sharp$ $(s,s',v,a,i,j,j',k,k',l,m,n,sm,sn)$ *atom sn'* $\sharp$ $(s,s',v,a,i,j,j',k,k',l,m,n,sm,sn,sm')$
atom m2 $\sharp$ $(s,s',v,a,i,j,j',k,k',l,m,n,sm,sn,sm',sn')$ *atom n2* $\sharp$ $(s,s',v,a,i,j,j',k,k',l,m,n,sm,sn,sm',sn',m2)$
atom sm2 $\sharp$ $(s,s',v,a,i,j,j',k,k',l,m,n,sm,sn,sm',sn',m2,n2)$ *atom sn2* $\sharp$ $(s,s',v,a,i,j,j',k,k',l,m,n,sm,sn,sm',sn',m2,n2,sm2)$
atom sm2' $\sharp$ $(s,s',v,a,i,j,j',k,k',l,m,n,sm,sn,sm',sn',m2,n2,sm2,sn2)$ *atom sn2'* $\sharp$ $(s,s',v,a,i,j,j',k,k',l,m,n,sm,sn,sm',sn',m2,n2,sm2,sn2,sm2')$
by (*metis obtain-fresh*)
have $\{$ *OrdP* (*Var k*), *VarP v* $\}$
$\vdash$ *All i* (*All j* (*All j'* (*All k'* (*SeqStTermP v a* (*Var i*) (*Var j*) *s* (*Var k*)
IMP (*SeqStTermP v a* (*Var i*) (*Var j'*) *s'* (*Var k'*) *IMP* *Var j' EQ Var j*)))))
apply (*rule OrdIndH* [**where** *j*=*l*])
using *atoms* **apply** *auto*
apply (*rule Swap*)

apply (*rule cut-same*)
apply (*rule cut1* [*OF SeqStTermP-lemma* [*of m v a Var i Var j s Var k n sm sm′ sn sn′*]], *simp-all*, *blast*)
apply (*rule cut-same*)
apply (*rule cut1* [*OF SeqStTermP-lemma* [*of m2 v a Var i Var j′ s′ Var k′ n2 sm2 sm2′ sn2 sn2′*]], *simp-all*, *blast*)
apply (*rule Disj-EH Conj-EH*)+
— case 1, both sides equal "v"
apply (*blast intro*: *Trans Sym*)
— case 2, *Var i EQ v* and also *IndP* (*Var i*) *OR Var i NEQ v*
apply (*rule Conj-EH Disj-EH*)+
apply (*blast intro*: *IndP-cong* [*THEN Iff-MP-same*] *not-IndP-VarP* [*THEN cut2*])
apply (*metis Assume OrdNotEqP-E*)
— case 3, both a variable and a pair
apply (*rule Ex-EH Conj-EH*)+
apply *simp-all*
apply (*rule cut-same* [**where** *A* = *VarP* (*Q-Eats* (*Var sm*) (*Var sn*))])
apply (*blast intro*: *Trans Sym VarP-cong* [**where** *x*=*v*, *THEN Iff-MP-same*] *Hyp*, *blast*)
— towards remaining cases
apply (*rule Disj-EH Ex-EH*)+
— case 4, *Var i EQ v* and also *IndP* (*Var i*) *OR Var i NEQ v*
apply (*blast intro*: *IndP-cong* [*THEN Iff-MP-same*] *not-IndP-VarP* [*THEN cut2*] *OrdNotEqP-E*)
— case 5, *Var i EQ v* for both
apply (*blast intro*: *Trans Sym*)
— case 6, both an atom and a pair
apply (*rule Ex-EH Conj-EH*)+
apply *simp-all*
apply (*rule atom-HPairE*)
apply (*simp add*: *HTuple.simps*)
apply (*blast intro*: *Trans*)
— towards remaining cases
apply (*rule Conj-EH Disj-EH Ex-EH*)+
apply *simp-all*
— case 7, both an atom and a pair
apply (*rule cut-same* [**where** *A* = *VarP* (*Q-Eats* (*Var sm2*) (*Var sn2*))])
apply (*blast intro*: *Trans Sym VarP-cong* [**where** *x*=*v*, *THEN Iff-MP-same*] *Hyp*, *blast*)
— case 8, both an atom and a pair
apply (*rule Ex-EH Conj-EH*)+
apply *simp-all*
apply (*rule atom-HPairE*)
apply (*simp add*: *HTuple.simps*)
apply (*blast intro*: *Trans*)
— case 9, two Eats terms
apply (*rule Ex-EH Disj-EH Conj-EH*)+
apply *simp-all*

apply (*rule All-E'* [*OF Hyp*, **where** *x*=*Var m*], *blast*)
apply (*rule All-E'* [*OF Hyp*, **where** *x*=*Var n*], *blast*, *simp*)
apply (*rule Disj-EH*, *blast intro*: *thin1 ContraProve*)+
apply (*rule All-E* [**where** *x*=*Var sm*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm'*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm2'*], *simp*)
apply (*rule All-E* [**where** *x*=*Var m2*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn*, *THEN rotate2*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn'*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn2'*], *simp*)
apply (*rule All-E* [**where** *x*=*Var n2*], *simp*)
apply (*rule cut-same* [**where** *A* = *Q-Eats* (*Var sm*) (*Var sn*) *EQ Q-Eats* (*Var sm2*) (*Var sn2*)])
apply (*blast intro*: *Sym Trans*, *clarify*)
apply (*rule cut-same* [**where** *A* = *SeqStTermP v a* (*Var sn*) (*Var sn2'*) *s'* (*Var n2*)])
apply (*blast intro*: *Hyp SeqStTermP-cong* [*OF Hyp Refl Refl*, *THEN Iff-MP2-same*])
apply (*rule cut-same* [**where** *A* = *SeqStTermP v a* (*Var sm*) (*Var sm2'*) *s'* (*Var m2*)])
apply (*blast intro*: *Hyp SeqStTermP-cong* [*OF Hyp Refl Refl*, *THEN Iff-MP2-same*])
apply (*rule Disj-EH*, *blast intro*: *thin1 ContraProve*)+
apply (*blast intro*: *HPair-cong Trans* [*OF Hyp Sym*])
done
hence *p1*: {*OrdP* (*Var k*), *VarP v*}
⊢ (*All j* (*All j'* (*All k'* (*SeqStTermP v a* (*Var i*) (*Var j*) *s* (*Var k*)
IMP (*SeqStTermP v a* (*Var i*) (*Var j'*) *s'* (*Var k'*) *IMP Var j' EQ Var j*)))))(*i*::=*t*)
by (*metis All-D*)
have *p2*: {*OrdP* (*Var k*), *VarP v*}
⊢ (*All j'* (*All k'* (*SeqStTermP v a t* (*Var j*) *s* (*Var k*)
IMP (*SeqStTermP v a t* (*Var j'*) *s'* (*Var k'*) *IMP Var j' EQ Var j*))))(*j*::=*u*)
apply (*rule All-D*)
using *atoms p1* **by** *simp*
have *p3*: {*OrdP* (*Var k*), *VarP v*}
⊢ (*All k'* (*SeqStTermP v a t u s* (*Var k*) *IMP* (*SeqStTermP v a t* (*Var j'*) *s'* (*Var k'*) *IMP Var j' EQ u*)))(*j'*::=*u'*)
apply (*rule All-D*)
using *atoms p2* **by** *simp*
have *p4*: {*OrdP* (*Var k*), *VarP v*}
⊢ (*SeqStTermP v a t u s* (*Var k*) *IMP* (*SeqStTermP v a t u' s'* (*Var k'*) *IMP u' EQ u*))(*k'*::=*kk'*)
apply (*rule All-D*)
using *atoms p3* **by** *simp*
hence {*SeqStTermP v a t u s* (*Var k*), *VarP v*} ⊢ *SeqStTermP v a t u s* (*Var k*) *IMP* (*SeqStTermP v a t u' s' kk' IMP u' EQ u*)
using *atoms* **apply** *simp*
by (*metis SeqStTermP-imp-OrdP rcut1*)
hence {*VarP v*} ⊢ ((*SeqStTermP v a t u s* (*Var k*) *IMP* (*SeqStTermP v a t u'*

s′ kk′ IMP u′ EQ u)))
  by (*metis Assume MP-same Imp-I*)
 hence {*VarP v*} ⊢ ((*SeqStTermP v a t u s* (*Var k*) *IMP* (*SeqStTermP v a t u′ s′ kk′ IMP u′ EQ u*)))(*k*::=*kk*)
  using *atoms* **by** (*force intro*!: *Subst*)
 hence {*VarP v*} ⊢ *SeqStTermP v a t u s kk IMP* (*SeqStTermP v a t u′ s′ kk′ IMP u′ EQ u*)
  using *atoms* **by** *simp*
 hence {*SeqStTermP v a t u s kk*} ⊢ *SeqStTermP v a t u s kk IMP* (*SeqStTermP v a t u′ s′ kk′ IMP u′ EQ u*)
  by (*metis SeqStTermP-imp-VarP rcut1*)
 thus *?thesis*
  by (*metis Assume AssumeH*(*2*) *MP-same rcut1*)
qed

theorem *SubstTermP-unique*: {*SubstTermP v tm t u*, *SubstTermP v tm t u′*} ⊢ *u′ EQ u*
proof −
 obtain *s*::*name* **and** *s′*::*name* **and** *k*::*name* **and** *k′*::*name*
  where *atom s* ♯ (*v*,*tm*,*t*,*u*,*u′*,*k*,*k′*) *atom s′* ♯ (*v*,*tm*,*t*,*u*,*u′*,*k*,*k′*,*s*)
   atom k ♯ (*v*,*tm*,*t*,*u*,*u′*) *atom k′* ♯ (*v*,*tm*,*t*,*u*,*u′*,*k*)
  by (*metis obtain-fresh*)
 thus *?thesis*
  by (*auto simp*: *SubstTermP.simps* [*of s v tm t u k*] *SubstTermP.simps* [*of s′ v tm t u′ k′*])
   (*metis SeqStTermP-unique rotate3 thin1*)
qed

7.0.4 *SubstAtomicP*

lemma *SubstTermP-eq*:
 ⟦*H* ⊢ *SubstTermP v tm x z*; *insert* (*SubstTermP v tm y z*) *H* ⊢ *A*⟧ ⟹ *insert* (*x EQ y*) *H* ⊢ *A*
 by (*metis Assume rotate2 Iff-E1 cut-same thin1 SubstTermP-cong* [*OF Refl Refl - Refl*])

lemma *SubstAtomicP-unique*: {*SubstAtomicP v tm x y*, *SubstAtomicP v tm x y′*} ⊢ *y′ EQ y*
proof −
 obtain *t*::*name* **and** *ts*::*name* **and** *u*::*name* **and** *us*::*name*
   and *t′*::*name* **and** *ts′*::*name* **and** *u′*::*name* **and** *us′*::*name*
  where *atom t* ♯ (*v*,*tm*,*x*,*y*,*y′*,*ts*,*u*,*us*) *atom ts* ♯ (*v*,*tm*,*x*,*y*,*y′*,*u*,*us*)
   atom u ♯ (*v*,*tm*,*x*,*y*,*y′*,*us*) *atom us* ♯ (*v*,*tm*,*x*,*y*,*y′*)
   atom t′ ♯ (*v*,*tm*,*x*,*y*,*y′*,*t*,*ts*,*u*,*us*,*ts′*,*u′*,*us′*) *atom ts′* ♯ (*v*,*tm*,*x*,*y*,*y′*,*t*,*ts*,*u*,*us*,*u′*,*us′*)
   atom u′ ♯ (*v*,*tm*,*x*,*y*,*y′*,*t*,*ts*,*u*,*us*,*us′*) *atom us′* ♯ (*v*,*tm*,*x*,*y*,*y′*,*t*,*ts*,*u*,*us*)
  by (*metis obtain-fresh*)
 thus *?thesis*
  apply (*simp add*: *SubstAtomicP.simps* [*of t v tm x y ts u us*]

SubstAtomicP.simps [*of t′ v tm x y′ ts′ u′ us′*])
apply (*rule Ex-EH Disj-EH Conj-EH*)+
apply *simp-all*
apply (*rule Eq-Trans-E* [*OF Hyp*], *auto simp*: *HTS*)
apply (*rule SubstTermP-eq* [*THEN thin1*], *blast*)
apply (*rule SubstTermP-eq* [*THEN rotate2*], *blast*)
apply (*rule Trans* [*OF Hyp Sym*], *blast*)
apply (*rule Trans* [*OF Hyp*], *blast*)
apply (*metis Assume AssumeH*(*8*) *HPair-cong Refl cut2* [*OF SubstTermP-unique*] *thin1*)
apply (*rule Eq-Trans-E* [*OF Hyp*], *blast*, *force simp add*: *HTS*)
apply (*rule Eq-Trans-E* [*OF Hyp*], *blast*, *force simp add*: *HTS*)
apply (*rule Eq-Trans-E* [*OF Hyp*], *auto simp*: *HTS*)
apply (*rule SubstTermP-eq* [*THEN thin1*], *blast*)
apply (*rule SubstTermP-eq* [*THEN rotate2*], *blast*)
apply (*rule Trans* [*OF Hyp Sym*], *blast*)
apply (*rule Trans* [*OF Hyp*], *blast*)
apply (*metis Assume AssumeH*(*8*) *HPair-cong Refl cut2* [*OF SubstTermP-unique*] *thin1*)
done
qed

7.0.5 *SeqSubstFormP*

lemma *SeqSubstFormP-lemma*:
assumes *atom m* ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*n*,*sm*,*sm′*,*sn*,*sn′*) *atom n* ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*sm*,*sm′*,*sn*,*sn′*)
atom sm ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*sm′*,*sn*,*sn′*) *atom sm′* ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*sn*,*sn′*)
atom sn ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*sn′*) *atom sn′* ♯ (*v*,*u*,*x*,*y*,*s*,*k*)
shows { *SeqSubstFormP v u x y s k* }
⊢ *SubstAtomicP v u x y OR*
Ex m (*Ex n* (*Ex sm* (*Ex sm′* (*Ex sn* (*Ex sn′* (*Var m IN k AND Var n IN k AND*
SeqSubstFormP v u (*Var sm*) (*Var sm′*) *s* (*Var m*) *AND*
SeqSubstFormP v u (*Var sn*) (*Var sn′*) *s* (*Var n*) *AND*
(((*x EQ Q-Disj* (*Var sm*) (*Var sn*) *AND y EQ Q-Disj* (*Var sm′*) (*Var sn′*)) *OR*
(*x EQ Q-Neg* (*Var sm*) *AND y EQ Q-Neg* (*Var sm′*)) *OR*
(*x EQ Q-Ex* (*Var sm*) *AND y EQ Q-Ex* (*Var sm′*))))))))))
proof −
obtain *l*::*name* **and** *sl*::*name* **and** *sl′*::*name*
where *atom l* ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
atom sl ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
atom sl′ ♯ (*v*,*u*,*x*,*y*,*s*,*k*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
by (*metis obtain-fresh*)
thus *?thesis* **using** *assms*
apply (*simp add*: *SeqSubstFormP.simps* [*of l s k v u sl sl′ m n sm sm′ sn sn′*])
apply (*rule Conj-EH Ex-EH All2-SUCC-E* [*THEN rotate2*] | *simp*)+
apply (*rule cut-same* [**where** *A* = *HPair x y EQ HPair* (*Var sl*) (*Var sl′*)])
apply (*metis Assume AssumeH*(*4*) *LstSeqP-EQ*)

apply *clarify*
apply (*rule Disj-EH*)
apply (*blast intro*: *Disj-I1 SubstAtomicP-cong* [*THEN Iff-MP2-same*])
— now the quantified cases
apply (*rule Ex-EH Conj-EH*)+
apply *simp-all*
apply (*rule Disj-I2*)
apply (*rule Ex-I* [**where** x = *Var m*], *simp*)
apply (*rule Ex-I* [**where** x = *Var n*], *simp*)
apply (*rule Ex-I* [**where** x = *Var sm*], *simp*)
apply (*rule Ex-I* [**where** x = *Var sm'*], *simp*)
apply (*rule Ex-I* [**where** x = *Var sn*], *simp*)
apply (*rule Ex-I* [**where** x = *Var sn'*], *simp*)
apply (*simp-all add*: *SeqSubstFormP.simps* [*of l s - v u sl sl' m n sm sm' sn sn'*])
apply ((*rule Conj-I*)+, *blast intro*: *LstSeqP-Mem*)+
— first SeqSubstFormP subgoal
apply (*rule All2-Subset* [*OF Hyp*], *blast*)
apply (*blast intro*!: *SUCC-Subset-Ord LstSeqP-OrdP*, *blast*, *simp*)
— next SeqSubstFormP subgoal
apply ((*rule Conj-I*)+, *blast intro*: *LstSeqP-Mem*)+
apply (*rule All2-Subset* [*OF Hyp*], *blast*)
apply (*blast intro*!: *SUCC-Subset-Ord LstSeqP-OrdP*, *blast*, *simp*)
— finally, the equality pairs
apply (*rule anti-deduction* [*THEN thin1*])
apply (*rule Sym-L* [*THEN rotate4*])
apply (*rule Var-Eq-subst-Iff* [*THEN Iff-MP-same*])
apply (*rule Sym-L* [*THEN rotate5*])
apply (*rule Var-Eq-subst-Iff* [*THEN Iff-MP-same*], *force*)
done
qed

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 −
obtain t::*name* **and** u::*name* **and** t'::*name* **and** u'::*name*
where *atom* $t \sharp (z,w,v,tm,y,y',t',u,u')$ *atom* $t' \sharp (z,w,v,tm,y,y',u,u')$
atom $u \sharp (z,w,v,tm,y,y',u')$ *atom* $u' \sharp (z,w,v,tm,y,y')$
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thesis3*
by (*auto simp*: *SubstAtomicP.simps* [*of t v tm y y' t' u u'*] *HTS intro*: *Eq-Trans-E* [*OF Hyp*])
qed

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*) $H \vdash A$
apply (*rule cut-same* [*OF SeqSubstFormP-cong* [*OF Assume Refl Refl Refl*, *THEN Iff-MP-same*]])
apply (*auto simp*: *insert-commute intro*: *thin1*)
done

lemma *SeqSubstFormP-unique*: {*SeqSubstFormP v a x y s kk*, *SeqSubstFormP v a x y' s' kk'*} $\vdash$ *y' EQ y*
proof −
obtain *i*::*name* **and** *j*::*name* **and** *j'*::*name* **and** *k*::*name* **and** *k'*::*name* **and** *l*::*name*
and *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name* **and** *sm'*::*name* **and** *sn'*::*name*
and *m2*::*name* **and** *n2*::*name* **and** *sm2*::*name* **and** *sn2*::*name* **and** *sm2'*::*name* **and** *sn2'*::*name*
where *atoms*: *atom i* ♯ (*s*,*s'*,*v*,*a*,*x*,*y*,*y'*) *atom j* ♯ (*s*,*s'*,*v*,*a*,*x*,*i*,*x*,*y*,*y'*)
atom j' ♯ (*s*,*s'*,*v*,*a*,*x*,*i*,*j*,*x*,*y*,*y'*)
atom k ♯ (*s*,*s'*,*v*,*a*,*x*,*y*,*y'*,*kk'*,*i*,*j*,*j'*) *atom k'* ♯ (*s*,*s'*,*v*,*a*,*x*,*y*,*y'*,*k*,*i*,*j*,*j'*)
atom l ♯ (*s*,*s'*,*v*,*a*,*x*,*i*,*j*,*j'*,*k*,*k'*)
atom m ♯ (*s*,*s'*,*v*,*a*,*i*,*j*,*j'*,*k*,*k'*,*l*) *atom n* ♯ (*s*,*s'*,*v*,*a*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*)
atom sm ♯ (*s*,*s'*,*v*,*a*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*) *atom sn* ♯ (*s*,*s'*,*v*,*a*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*)
atom sm' ♯ (*s*,*s'*,*v*,*a*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*) *atom sn'* ♯ (*s*,*s'*,*v*,*a*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*)
atom m2 ♯ (*s*,*s'*,*v*,*a*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*,*sn'*) *atom n2* ♯ (*s*,*s'*,*v*,*a*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*,*sn'*,*m2*)
atom sm2 ♯ (*s*,*s'*,*v*,*a*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*,*sn'*,*m2*,*n2*) *atom sn2* ♯ (*s*,*s'*,*v*,*a*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*,*sn'*,*m2*,*n2*,*sm2*)
atom sm2' ♯ (*s*,*s'*,*v*,*a*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*,*sn'*,*m2*,*n2*,*sm2*,*sn2*)
atom sn2' ♯ (*s*,*s'*,*v*,*a*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*,*sn'*,*m2*,*n2*,*sm2*,*sn2*,*sm2'*)
by (*metis obtain-fresh*)
have { *OrdP* (*Var k*) }
$\vdash$ *All i* (*All j* (*All j'* (*All k'* (*SeqSubstFormP v a* (*Var i*) (*Var j*) *s* (*Var k*)
IMP (*SeqSubstFormP v a* (*Var i*) (*Var j'*) *s'* (*Var k'*)
IMP Var j' EQ Var j)))))
apply (*rule OrdIndH* [**where** *j*=*l*])
using *atoms* **apply** *auto*
apply (*rule Swap*)
apply (*rule cut-same*)
apply (*rule cut1* [*OF SeqSubstFormP-lemma* [*of m v a Var i Var j s Var k n sm sm' sn sn'*]], *simp-all*, *blast*)
apply (*rule cut-same*)
apply (*rule cut1* [*OF SeqSubstFormP-lemma* [*of m2 v a Var i Var j' s' Var k' n2 sm2 sm2' sn2 sn2'*]], *simp-all*, *blast*)
apply (*rule Disj-EH Conj-EH*)+
— case 1, both sides are atomic
apply (*blast intro*: *cut2* [*OF SubstAtomicP-unique*])
— case 2, atomic and also not

apply (*rule Ex-EH Conj-EH Disj-EH*)+
apply *simp-all*
apply (*metis Assume AssumeH*(*7*) *Disj-I1 Neg-I anti-deduction cut2* [*OF Disj-SubstAtomicP-Fls*])
apply (*rule Conj-EH Disj-EH*)+
apply (*metis Assume AssumeH*(*7*) *Disj-I1 Neg-I anti-deduction cut2* [*OF Neg-SubstAtomicP-Fls*])
apply (*rule Conj-EH*)+
apply (*metis Assume AssumeH*(*7*) *Disj-I1 Neg-I anti-deduction cut2* [*OF Ex-SubstAtomicP-Fls*])
— towards remaining cases
apply (*rule Conj-EH Disj-EH Ex-EH*)+
apply *simp-all*
apply (*metis Assume AssumeH*(*7*) *Disj-I1 Neg-I anti-deduction cut2* [*OF Disj-SubstAtomicP-Fls*])
apply (*rule Conj-EH Disj-EH*)+
apply (*metis Assume AssumeH*(*7*) *Disj-I1 Neg-I anti-deduction cut2* [*OF Neg-SubstAtomicP-Fls*])
apply (*rule Conj-EH*)+
apply (*metis Assume AssumeH*(*7*) *Disj-I1 Neg-I anti-deduction cut2* [*OF Ex-SubstAtomicP-Fls*])
— towards remaining cases
apply (*rule Conj-EH Disj-EH Ex-EH*)+
apply *simp-all*
— case two Disj terms
apply (*rule All-E′* [*OF Hyp*, **where** *x*=*Var m*], *blast*)
apply (*rule All-E′* [*OF Hyp*, **where** *x*=*Var n*], *blast*, *simp*)
apply (*rule Disj-EH*, *blast intro*: *thin1 ContraProve*)+
apply (*rule All-E* [**where** *x*=*Var sm*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm2′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var m2*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn*, *THEN rotate2*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn2′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var n2*], *simp*)
apply (*rule rotate3*)
apply (*rule Eq-Trans-E* [*OF Hyp*], *blast*)
apply (*clarsimp simp add*: *HTS*)
apply (*rule thin1*)
apply (*rule Disj-EH* [*OF ContraProve*], *blast intro*: *thin1 SeqSubstFormP-eq*)+
apply (*blast intro*: *HPair-cong Trans* [*OF Hyp Sym*])
— towards remaining cases
apply (*rule Conj-EH Disj-EH*)+
— Negation = Disjunction?
apply (*rule Eq-Trans-E* [*OF Hyp*], *blast*, *force simp add*: *HTS*)
— Existential = Disjunction?
apply (*rule Conj-EH*)
apply (*rule Eq-Trans-E* [*OF Hyp*], *blast*, *force simp add*: *HTS*)

— towards remaining cases
apply (*rule Conj-EH Disj-EH Ex-EH*)+
apply *simp-all*
— Disjunction = Negation?
apply (*rule Eq-Trans-E* [*OF Hyp*], *blast*, *force simp add*: *HTS*)
apply (*rule Conj-EH Disj-EH*)+
— case two Neg terms
apply (*rule Eq-Trans-E* [*OF Hyp*], *blast*, *clarify*)
apply (*rule thin1*)
apply (*rule All-E′* [*OF Hyp*, **where** *x*=*Var m*], *blast*, *simp*)
apply (*rule Disj-EH*, *blast intro*: *thin1 ContraProve*)+
apply (*rule All-E* [**where** *x*=*Var sm*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm2′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var m2*], *simp*)
apply (*rule Disj-EH* [*OF ContraProve*], *blast intro*: *SeqSubstFormP-eq Sym-L*)+
apply (*blast intro*: *HPair-cong Sym Trans* [*OF Hyp*])
— Existential = Negation?
apply (*rule Conj-EH*)+
apply (*rule Eq-Trans-E* [*OF Hyp*], *blast*, *force simp add*: *HTS*)
— towards remaining cases
apply (*rule Conj-EH Disj-EH Ex-EH*)+
apply *simp-all*
— Disjunction = Existential
apply (*rule Eq-Trans-E* [*OF Hyp*], *blast*, *force simp add*: *HTS*)
apply (*rule Conj-EH Disj-EH Ex-EH*)+
— Negation = Existential
apply (*rule Eq-Trans-E* [*OF Hyp*], *blast*, *force simp add*: *HTS*)
— case two Ex terms
apply (*rule Conj-EH*)+
apply (*rule Eq-Trans-E* [*OF Hyp*], *blast*, *clarify*)
apply (*rule thin1*)
apply (*rule All-E′* [*OF Hyp*, **where** *x*=*Var m*], *blast*, *simp*)
apply (*rule Disj-EH*, *blast intro*: *thin1 ContraProve*)+
apply (*rule All-E* [**where** *x*=*Var sm*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm2′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var m2*], *simp*)
apply (*rule Disj-EH* [*OF ContraProve*], *blast intro*: *SeqSubstFormP-eq Sym-L*)+
apply (*blast intro*: *HPair-cong Sym Trans* [*OF Hyp*])
done
hence *p1*: {*OrdP* (*Var k*)}
⊢ (*All j* (*All j′* (*All k′* (*SeqSubstFormP v a* (*Var i*) (*Var j*) *s* (*Var k*)
IMP (*SeqSubstFormP v a* (*Var i*) (*Var j′*) *s′* (*Var k′*) *IMP Var j′*
EQ Var j)))))(*i*::=*x*)
by (*metis All-D*)
have *p2*: {*OrdP* (*Var k*)}
⊢ (*All j′* (*All k′* (*SeqSubstFormP v a x* (*Var j*) *s* (*Var k*)
IMP (*SeqSubstFormP v a x* (*Var j′*) *s′* (*Var k′*) *IMP Var j′ EQ Var*

j))))(*j*::=*y*)
  apply (*rule All-D*)
  using *atoms p1* **by** *simp*
 have *p3*: {*OrdP* (*Var k*)}
   ⊢ (*All k′* (*SeqSubstFormP v a x y s* (*Var k*)
     IMP (*SeqSubstFormP v a x* (*Var j′*) *s′* (*Var k′*) *IMP Var j′ EQ* *y*)))(*j′*::=*y′*)
  apply (*rule All-D*)
  using *atoms p2* **by** *simp*
 have *p4*: {*OrdP* (*Var k*)}
   ⊢ (*SeqSubstFormP v a x y s* (*Var k*) *IMP* (*SeqSubstFormP v a x y′ s′* (*Var* *k′*) *IMP y′ EQ y*))(*k′*::=*kk′*)
  apply (*rule All-D*)
  using *atoms p3* **by** *simp*
 hence {*OrdP* (*Var k*)} ⊢ *SeqSubstFormP v a x y s* (*Var k*) *IMP* (*SeqSubstFormP* *v a x y′ s′ kk′ IMP y′ EQ y*)
  using *atoms* **by** *simp*
 hence {*SeqSubstFormP v a x y s* (*Var k*)}
   ⊢ *SeqSubstFormP v a x y s* (*Var k*) *IMP* (*SeqSubstFormP v a x y′ s′ kk′* *IMP y′ EQ y*)
  by (*metis SeqSubstFormP-imp-OrdP rcut1*)
 hence {} ⊢ *SeqSubstFormP v a x y s* (*Var k*) *IMP* (*SeqSubstFormP v a x y′ s′* *kk′ IMP y′ EQ y*)
  by (*metis Assume Disj-Neg-2 Disj-commute anti-deduction Imp-I*)
 hence {} ⊢ ((*SeqSubstFormP v a x y s* (*Var k*) *IMP* (*SeqSubstFormP v a x y′ s′* *kk′ IMP y′ EQ y*)))(*k*::=*kk*)
  using *atoms* **by** (*force intro*!: *Subst*)
 thus *?thesis*
  using *atoms* **by** *simp* (*metis DisjAssoc2 Disj-commute anti-deduction*)
qed

7.0.6 *SubstFormP*

theorem *SubstFormP-unique*: {*SubstFormP v tm x y*, *SubstFormP v tm x y′*} ⊢ *y′ EQ y*
proof −
 obtain *s*::*name* **and** *s′*::*name* **and** *k*::*name* **and** *k′*::*name*
  where *atom s* ♯ (*v*,*tm*,*x*,*y*,*y′*,*k*,*k′*) *atom s′* ♯ (*v*,*tm*,*x*,*y*,*y′*,*k*,*k′*,*s*)
   atom k ♯ (*v*,*tm*,*x*,*y*,*y′*) *atom k′* ♯ (*v*,*tm*,*x*,*y*,*y′*,*k*)
  by (*metis obtain-fresh*)
 thus *?thesis*
  by (*force simp*: *SubstFormP.simps* [*of s v tm x y k*] *SubstFormP.simps* [*of s′* *v tm x y′ k′*]
     SeqSubstFormP-unique rotate3 thin1)
qed

end

Chapter 8

Section 6 Material and Gdel's First Incompleteness Theorem

theory *Goedel-I*
imports *Pf-Predicates Functions*
begin

8.1 The Function W and Lemma 6.1

8.1.1 Predicate form, defined on sequences

definition *SeqWR* :: $hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$
  where $SeqWR\ s\ k\ y \equiv LstSeq\ s\ k\ y \wedge app\ s\ 0 = 0 \wedge$
  $(\forall l \in k.\ app\ s\ (succ\ l) = q\text{-}Eats\ (app\ s\ l)\ (app\ s\ l))$

nominal-function *SeqWRP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
  where $[\![atom\ l\ \sharp\ (s,k,sl);\ atom\ sl\ \sharp\ (s)]\!] \Longrightarrow$
    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))
  by (*auto simp*: *eqvt-def SeqWRP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma
  shows *SeqWRP-fresh-iff* [*simp*]: $a \sharp SeqWRP\ s\ k\ y \longleftrightarrow a \sharp s \wedge a \sharp k \wedge a \sharp y$ (**is** *?thesis1*)
    and *eval-fm-SeqWRP* [*simp*]: $eval\text{-}fm\ e\ (SeqWRP\ s\ k\ y) \longleftrightarrow SeqWR\ [\![s]\!]e\ [\![k]\!]e\ [\![y]\!]e$ (**is** *?thesis2*)
    and *SeqWRP-sf* [*iff*]: *Sigma-fm* (*SeqWRP s k y*) (**is** *?thsf*)

proof −
obtain *l*::*name* **and** *sl*::*name* **where** *atom l* ♯ (*s*,*k*,*sl*) *atom sl* ♯ (*s*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thsf*
by (*auto simp*: *SeqWR-def q-defs LstSeq-imp-Ord*
Seq-iff-app [*of* ⟦*s*⟧*e*, *OF LstSeq-imp-Seq-succ*]
Ord-trans [*of - - succ* ⟦*k*⟧*e*])
qed

lemma *SeqWRP-subst* [*simp*]:
(*SeqWRP s k y*)(*i*::=*t*) = *SeqWRP* (*subst i t s*) (*subst i t k*) (*subst i t y*)
proof −
obtain *l*::*name* **and** *sl*::*name*
where *atom l* ♯ (*s*,*k*,*sl*,*t*,*i*) *atom sl* ♯ (*s*,*k*,*t*,*i*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SeqWRP.simps* [**where** *l*=*l* **and** *sl*=*sl*])
qed

lemma *SeqWRP-cong*:
assumes $H \vdash s\ EQ\ s'$ **and** $H \vdash k\ EQ\ k'$ **and** $H \vdash y\ EQ\ y'$
shows $H \vdash SeqWRP\ s\ k\ y\ IFF\ SeqWRP\ s'\ k'\ y'$
by (*rule P3-cong* [*OF - assms*], *auto*)

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

8.1.2 Predicate form of W

definition *WR* :: *hf* ⇒ *hf* ⇒ *bool*
where *WR x y* ≡ (∃ *s*. *SeqWR s x y*)

nominal-function *WRP* :: *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom s* ♯ (*x*,*y*)⟧ ⟹
WRP x y = *Ex s* (*SeqWRP* (*Var s*) *x y*)
by (*auto simp*: *eqvt-def WRP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *WRP-fresh-iff* [*simp*]: *a* ♯ *WRP x y* ⟷ *a* ♯ *x* ∧ *a* ♯ *y* (**is** *?thesis1*)
and *eval-fm-WRP* [*simp*]: *eval-fm e* (*WRP x y*) ⟷ *WR* ⟦*x*⟧*e* ⟦*y*⟧*e* (**is** *?thesis2*)
and *sigma-fm-WRP* [*simp*]: *Sigma-fm* (*WRP x y*) (**is** *?thsf*)
proof −
obtain *s*::*name* **where** *atom s* ♯ (*x*,*y*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thsf*
by (*auto simp*: *WR-def*)

qed

lemma *WRP-subst* [*simp*]: $(WRP\ x\ y)(i::=t) = WRP\ (subst\ i\ t\ x)\ (subst\ i\ t\ y)$
proof –
  obtain *s*::*name* **where** $atom\ s\ \sharp\ (x,y,t,i)$
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *WRP.simps* [*of s*])
qed

lemma *WRP-cong*: $H \vdash t\ EQ\ t' \Longrightarrow H \vdash u\ EQ\ u' \Longrightarrow H \vdash WRP\ t\ u\ IFF\ WRP\ t'\ u'$
  by (*rule P2-cong*) *auto*

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

lemma *WR0-iff*: $WR\ 0\ y \longleftrightarrow y=0$
  by (*simp add*: *WR-def SeqWR-def*) (*metis LstSeq-1 LstSeq-app*)

lemma *WR0*: $WR\ 0\ 0$
  by (*simp add*: *WR0-iff*)

lemma *WR-succ-iff*: **assumes** *i*: *Ord i* **shows** $WR\ (succ\ i)\ z = (\exists\, y.\ z = q\text{-}Eats\ y\ y \wedge WR\ i\ y)$
proof
  assume $WR\ (succ\ i)\ z$
  then obtain *s* **where** *s*: $SeqWR\ s\ (succ\ i)\ z$
    by (*auto simp*: *WR-def i*)
  moreover then have $app\ s\ (succ\ i) = z$
    by (*auto simp*: *SeqWR-def*)
  ultimately show $\exists\, y.\ z = q\text{-}Eats\ y\ y \wedge WR\ i\ y$ **using** *i*
    by (*auto simp*: *WR-def SeqWR-def*) (*metis LstSeq-trunc hmem-succ-self*)
next
  assume $\exists\, y.\ z = q\text{-}Eats\ y\ y \wedge WR\ i\ y$
  then obtain *y* **where** *z*: $z = q\text{-}Eats\ y\ y$ **and** *y*: $WR\ i\ y$
    by *blast*
  thus $WR\ (succ\ i)\ z$ **using** *i*
    apply (*auto simp*: *WR-def SeqWR-def*)
    apply (*rule-tac* x=*insf s* (*succ i*) (*q-Eats y y*) **in** *exI*)
   apply (*auto simp*: *LstSeq-imp-Seq-succ app-insf-Seq-if LstSeq-insf succ-notin-self*)
    done
qed

lemma *WR-succ*: $Ord\ i \Longrightarrow WR\ (succ\ i)\ (q\text{-}Eats\ y\ y) = WR\ i\ y$
  by (*metis WR-succ-iff q-Eats-iff*)

lemma *WR-ord-of*: $WR\ (ord\text{-}of\ i)\ [\![\lceil ORD\text{-}OF\ i\rceil]\!]e$
  by (*induct i*) (*auto simp*: *WR0-iff WR-succ-iff quot-Succ q-defs*)

Lemma 6.1

lemma *WR-quot-Var*: *WR* $[\![\ulcorner Var\ x \urcorner]\!]e$ $[\![\ulcorner\ulcorner Var\ x \urcorner\urcorner]\!]e$
  by (*auto simp*: *quot-Var quot-Succ*)
    (*metis One-nat-def Ord-ord-of WR-ord-of WR-succ htuple.simps q-Eats-def*)

lemma *ground-WRP* [*simp*]: *ground-fm* (*WRP x y*) $\longleftrightarrow$ *ground x* $\wedge$ *ground y*
  by (*auto simp*: *ground-aux-def ground-fm-aux-def supp-conv-fresh*)

lemma *prove-WRP*: {} $\vdash$ *WRP* $\ulcorner Var\ x \urcorner$ $\ulcorner\ulcorner Var\ x \urcorner\urcorner$
  by (*auto simp*: *WR-quot-Var ground-aux-def supp-conv-fresh intro*: *Sigma-fm-imp-thm*)

8.1.3 Proving that these relations are functions

lemma *SeqWRP-Zero-E*:
  assumes *insert* (*y EQ Zero*) *H* $\vdash$ *A* *H* $\vdash$ *k EQ Zero*
  shows *insert* (*SeqWRP s k y*) *H* $\vdash$ *A*
proof –
  obtain *l*::*name* **and** *sl*::*name*
    where *atom l* $\sharp$ (*s*,*k*,*sl*) *atom sl* $\sharp$ (*s*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    apply (*auto simp*: *SeqWRP.simps* [**where** *s*=*s* **and** *l*=*l* **and** *sl*=*sl*])
    apply (*rule cut-same* [**where** *A* = *LstSeqP s Zero y*])
    apply (*blast intro*: *thin1 assms LstSeqP-cong* [*OF Refl - Refl*, *THEN Iff-MP-same*])
    apply (*rule cut-same* [**where** *A* = *y EQ Zero*])
    apply (*blast intro*: *LstSeqP-EQ*)
    apply (*metis rotate2 assms*(*1*) *thin1*)
    done
qed

lemma *SeqWRP-SUCC-lemma*:
  assumes *y'*: *atom y'* $\sharp$ (*s*,*k*,*y*)
  shows {*SeqWRP s* (*SUCC k*) *y*} $\vdash$ *Ex y'* (*SeqWRP s k* (*Var y'*) *AND y EQ Q-Succ* (*Var y'*))
proof –
  obtain *l*::*name* **and** *sl*::*name*
    where *atoms*: *atom l* $\sharp$ (*s*,*k*,*y*,*y'*,*sl*) *atom sl* $\sharp$ (*s*,*k*,*y*,*y'*)
    by (*metis obtain-fresh*)
  thus *?thesis* **using** *y'*
    apply (*auto simp*: *SeqWRP.simps* [**where** *s*=*s* **and** *l*=*l* **and** *sl*=*sl*])
    apply (*rule All2-SUCC-E'* [**where** *t*=*k*, *THEN rotate2*], *auto*)
    apply (*rule Ex-I* [**where** *x* = *Var sl*], *auto*)
    apply (*blast intro*: *LstSeqP-SUCC*) — showing *SeqWRP s k* (*Var sl*)
    apply (*blast intro*: *ContraProve LstSeqP-EQ*)
    done
qed

lemma *SeqWRP-SUCC-E*:
  assumes *y'*: *atom y'* $\sharp$ (*s*,*k*,*y*) **and** *k'*: *H* $\vdash$ *k' EQ* (*SUCC k*)
  shows *insert* (*SeqWRP s k' y*) *H* $\vdash$ *Ex y'* (*SeqWRP s k* (*Var y'*) *AND y EQ*

Q-Succ (*Var y'*))
  using *SeqWRP-cong* [*OF Refl k' Refl*] *cut1* [*OF SeqWRP-SUCC-lemma* [*of y' s k y*]]
  by (*metis Assume Iff-MP-left Iff-sym y'*)

lemma *SeqWRP-unique*: {*OrdP x*, *SeqWRP s x y*, *SeqWRP s' x y'*} ⊢ *y' EQ y*
proof −
  obtain *i*::*name* **and** *j*::*name* **and** *j'*::*name* **and** *k*::*name* **and** *sl*::*name* **and** *sl'*::*name* **and** *l*::*name* **and** *pi*::*name*
    where *i*: *atom i* ♯ (*s*,*s'*,*y*,*y'*) **and** *j*: *atom j* ♯ (*s*,*s'*,*i*,*x*,*y*,*y'*) **and** *j'*: *atom j'* ♯ (*s*,*s'*,*i*,*j*,*x*,*y*,*y'*)
     and *atoms*: *atom k* ♯ (*s*,*s'*,*i*,*j*,*j'*) *atom sl* ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*) *atom sl'* ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*sl*)
       atom pi ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*sl*,*sl'*)
    by (*metis obtain-fresh*)
  have {*OrdP* (*Var i*)} ⊢ *All j* (*All j'* (*SeqWRP s* (*Var i*) (*Var j*) *IMP* (*SeqWRP s'* (*Var i*) (*Var j'*) *IMP Var j' EQ Var j*)))
    apply (*rule OrdIndH* [**where** *j*=*k*])
    using *i j j' atoms* **apply** *auto*
    apply (*rule rotate4*)
    apply (*rule OrdP-cases-E* [**where** *k*=*pi*], *simp-all*)
    — Zero case
    apply (*rule SeqWRP-Zero-E* [*THEN rotate3*])
    prefer *2* **apply** *blast*
    apply (*rule SeqWRP-Zero-E* [*THEN rotate4*])
    prefer *2* **apply** *blast*
    apply (*blast intro*: *ContraProve* [*THEN rotate4*] *Sym Trans*)
    — SUCC case
    apply (*rule Ex-I* [**where** *x* = *Var pi*], *auto*)
    apply (*metis ContraProve EQ-imp-SUBS2 Mem-SUCC-I2 Refl Subset-D*)
    apply (*rule cut-same*)
    apply (*rule SeqWRP-SUCC-E* [*of sl' s' Var pi*, *THEN rotate4*], *auto*)
    apply (*rule cut-same*)
    apply (*rule SeqWRP-SUCC-E* [*of sl s Var pi*, *THEN rotate7*], *auto*)
    apply (*rule All-E* [**where** *x* = *Var sl*, *THEN rotate5*], *simp*)
    apply (*rule All-E* [**where** *x* = *Var sl'*], *simp*)
    apply (*rule Imp-E*, *blast*)+
    apply (*rule cut-same* [*OF Q-Succ-cong* [*OF Assume*]])
    apply (*blast intro*: *Trans* [*OF Hyp Sym*] *HPair-cong*)
    done
  hence {*OrdP* (*Var i*)} ⊢ (*All j'* (*SeqWRP s* (*Var i*) (*Var j*) *IMP* (*SeqWRP s'* (*Var i*) (*Var j'*) *IMP Var j' EQ Var j*)))(*j*::=*y*)
    by (*metis All-D*)
  hence {*OrdP* (*Var i*)} ⊢ (*SeqWRP s* (*Var i*) *y IMP* (*SeqWRP s'* (*Var i*) (*Var j'*) *IMP Var j' EQ y*))(*j'*::=*y'*)
    using *j j'*
    by *simp* (*drule All-D* [**where** *x*=*y'*], *simp*)
  hence {} ⊢ *OrdP* (*Var i*) *IMP* (*SeqWRP s* (*Var i*) *y IMP* (*SeqWRP s'* (*Var i*) *y' IMP y' EQ y*))
    using *j j'*

by *simp* (*metis Imp-I*)
hence {} ⊢ (*OrdP* (*Var i*) *IMP* (*SeqWRP s* (*Var i*) *y IMP* (*SeqWRP s′* (*Var i*) *y′ IMP y′ EQ y*)))(*i*::=*x*)
by (*metis Subst emptyE*)
thus *?thesis* **using** *i*
by *simp* (*metis anti-deduction insert-commute*)
qed

theorem *WRP-unique*: {*OrdP x*, *WRP x y*, *WRP x y′*} ⊢ *y′ EQ y*
proof −
obtain *s*::*name* **and** *s′*::*name*
where *atom s* ♯ (*x*,*y*,*y′*) *atom s′* ♯ (*x*,*y*,*y′*,*s*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SeqWRP-unique* [*THEN rotate3*] *WRP.simps* [*of s - y*] *WRP.simps* [*of s′ - y′*])
qed

8.1.4 The equivalent function

definition *W* :: *hf* ⇒ *tm*
where *W* ≡ *hmemrec* (*λf z. if z=0 then Zero else Q-Eats* (*f* (*pred z*)) (*f* (*pred z*)))

lemma *W0* [*simp*]: *W 0* = *Zero*
by (*rule trans* [*OF def-hmemrec* [*OF W-def*]]) *auto*

lemma *W-succ* [*simp*]: *Ord i* ⟹ *W* (*succ i*) = *Q-Eats* (*W i*) (*W i*)
by (*rule trans* [*OF def-hmemrec* [*OF W-def*]]) (*auto simp*: *ecut-apply SUCC-def W-def*)

lemma *W-ord-of* [*simp*]: *W* (*ord-of i*) = ⌈*ORD-OF i*⌉
by (*induct i*, *auto simp*: *SUCC-def quot-simps*)

lemma *WR-iff-eq-W*: *Ord x* ⟹ *WR x y* ⟷ *y* = ⟦*W x*⟧*e*
proof (*induct x arbitrary*: *y rule*: *Ord-induct2*)
case *0* **thus** *?case*
by (*metis W0 WR0-iff eval-tm.simps*(*1*))
next
case (*succ k*) **thus** *?case*
by (*auto simp*: *WR-succ-iff q-Eats-def*)
qed

8.2 The Function HF and Lemma 6.2

definition *SeqHR* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SeqHR x x′ s k* ≡
BuildSeq2 (*λy y′. Ord y* ∧ *WR y y′*)
(*λu u′ v v′ w w′. u* = ⟨*v*,*w*⟩ ∧ *u′* = *q-HPair v′ w′*) *s k x x′*

8.2.1 Defining the syntax: quantified body

nominal-function *SeqHRP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
  where ⟦*atom l* ♯ (*s*,*k*,*sl*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*);
    atom sl ♯ (*s*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*);
    atom sl' ♯ (*s*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*);
    atom m ♯ (*s*,*n*,*sm*,*sm'*,*sn*,*sn'*);
    atom n ♯ (*s*,*sm*,*sm'*,*sn*,*sn'*);
    atom sm ♯ (*s*,*sm'*,*sn*,*sn'*);
    atom sm' ♯ (*s*,*sn*,*sn'*);
    atom sn ♯ (*s*,*sn'*);
    atom sn' ♯ (*s*)⟧ ⟹
   SeqHRP x x' s k =
    LstSeqP s k (*HPair x x'*) *AND*
    All2 l (*SUCC k*) (*Ex sl* (*Ex sl'* (*HPair* (*Var l*) (*HPair* (*Var sl*) (*Var sl'*)) *IN*
s AND
      ((*OrdP* (*Var sl*) *AND WRP* (*Var sl*) (*Var sl'*)) *OR*
       Ex m (*Ex n* (*Ex sm* (*Ex sm'* (*Ex sn* (*Ex sn'* (*Var m IN Var l AND*
Var n IN Var l AND
        HPair (*Var m*) (*HPair* (*Var sm*) (*Var sm'*)) *IN s AND*
        HPair (*Var n*) (*HPair* (*Var sn*) (*Var sn'*)) *IN s AND*
        Var sl EQ HPair (*Var sm*) (*Var sn*) *AND*
        Var sl' EQ Q-HPair (*Var sm'*) (*Var sn'*))))))))))
by (*auto simp*: *eqvt-def SeqHRP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma
 shows *SeqHRP-fresh-iff* [*simp*]:
   a ♯ *SeqHRP x x' s k* ⟷ *a* ♯ *x* ∧ *a* ♯ *x'* ∧ *a* ♯ *s* ∧ *a* ♯ *k* (**is** *?thesis1*)
  and *eval-fm-SeqHRP* [*simp*]:
   eval-fm e (*SeqHRP x x' s k*) ⟷ *SeqHR* ⟦*x*⟧*e* ⟦*x'*⟧*e* ⟦*s*⟧*e* ⟦*k*⟧*e* (**is** *?thesis2*)
  and *SeqHRP-sf* [*iff*]: *Sigma-fm* (*SeqHRP x x' s k*) (**is** *?thsf*)
  and *SeqHRP-imp-OrdP*: { *SeqHRP x y s k* } ⊢ *OrdP k* (**is** *?thord*)
proof −
  obtain *l*::*name* **and** *sl*::*name* **and** *sl'*::*name* **and** *m*::*name* **and** *n*::*name* **and**
    sm::*name* **and** *sm'*::*name* **and** *sn*::*name* **and** *sn'*::*name*
   where *atoms*:
    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*)
   by (*metis obtain-fresh*)
  thus *?thesis1 ?thsf ?thord*
   by (*auto intro*: *LstSeqP-OrdP*)
  show *?thesis2* **using** *atoms*
   by (*fastforce simp*: *LstSeq-imp-Ord SeqHR-def*
     BuildSeq2-def BuildSeq-def Builds-def

HBall-def q-HPair-def q-Eats-def
Seq-iff-app [*of* $[\![s]\!]e$, *OF LstSeq-imp-Seq-succ*]
Ord-trans [*of - - succ* $[\![k]\!]e$]
cong: *conj-cong*)
qed

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 −
obtain l::*name* **and** sl::*name* **and** sl'::*name* **and** m::*name* **and** n::*name* **and** sm::*name* **and** sm'::*name* **and** sn::*name* **and** sn'::*name*
where *atom* $l \sharp (s,k,t,i,sl,sl',m,n,sm,sm',sn,sn')$
atom $sl \sharp (s,t,i,sl',m,n,sm,sm',sn,sn')$
atom $sl' \sharp (s,t,i,m,n,sm,sm',sn,sn')$
atom $m \sharp (s,t,i,n,sm,sm',sn,sn')$ *atom* $n \sharp (s,t,i,sm,sm',sn,sn')$
atom $sm \sharp (s,t,i,sm',sn,sn')$ *atom* $sm' \sharp (s,t,i,sn,sn')$
atom $sn \sharp (s,t,i,sn')$ *atom* $sn' \sharp (s,t,i)$
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SeqHRP.simps* [*of l - - sl sl' m n sm sm' sn sn'*])
qed

lemma *SeqHRP-cong*:
assumes $H \vdash x\ EQ\ x'$ **and** $H \vdash y\ EQ\ y'$ $H \vdash s\ EQ\ s'$ **and** $H \vdash k\ EQ\ k'$
shows $H \vdash SeqHRP\ x\ y\ s\ k\ IFF\ SeqHRP\ x'\ y'\ s'\ k'$
by (*rule P4-cong* [*OF - assms*], *auto*)

8.2.2 Defining the syntax: main predicate

definition *HR* :: *hf* ⇒ *hf* ⇒ *bool*
where $HR\ x\ x' \equiv \exists\, s\ k.\ SeqHR\ x\ x'\ s\ k$

nominal-function *HRP* :: *tm* ⇒ *tm* ⇒ *fm*
where $[\![atom\ s \sharp (x,x',k);\ atom\ k \sharp (x,x')]\!] \Longrightarrow$
$HRP\ x\ x' = Ex\ s\ (Ex\ k\ (SeqHRP\ x\ x'\ (Var\ s)\ (Var\ k)))$
by (*auto simp*: *eqvt-def HRP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *HRP-fresh-iff* [*simp*]: $a \sharp HRP\ x\ x' \longleftrightarrow a \sharp x \wedge a \sharp x'$ (**is** *?thesis1*)
and *eval-fm-HRP* [*simp*]: *eval-fm* e $(HRP\ x\ x') \longleftrightarrow HR\ [\![x]\!]e\ [\![x']\!]e$ (**is** *?thesis2*)
and *HRP-sf* [*iff*]: *Sigma-fm* $(HRP\ x\ x')$ (**is** *?thsf*)
proof −
obtain s::*name* **and** k::*name* **where** *atom* $s \sharp (x,x',k)$ *atom* $k \sharp (x,x')$
by (*metis obtain-fresh*)

thus *?thesis1 ?thesis2 ?thsf*
by (*auto simp*: *HR-def q-defs*)
qed

lemma *HRP-subst* [*simp*]: (*HRP x x'*)(*i*::=*t*) = *HRP* (*subst i t x*) (*subst i t x'*)
proof −
obtain *s*::*name* **and** *k*::*name* **where** *atom s* ♯ (*x*,*x'*,*t*,*i*,*k*) *atom k* ♯ (*x*,*x'*,*t*,*i*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *HRP.simps* [*of s - - k*])
qed

8.2.3 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 −
obtain *l*::*name* **and** *sl*::*name* **and** *sl'*::*name*
where *atoms*:
atom l ♯ (*x*,*x'*,*s*,*k*,*sl*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl ♯ (*x*,*x'*,*s*,*k*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl' ♯ (*x*,*x'*,*s*,*k*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
by (*metis obtain-fresh*)
thus *?thesis* **using** *atoms assms*
apply (*simp add*: *SeqHRP.simps* [*of l s k sl sl' m n sm sm' sn sn'*])
apply (*rule Conj-E*)
apply (*rule All2-SUCC-E'* [**where** *t*=*k*, *THEN rotate2*], *simp-all*)
apply (*rule rotate2*)
apply (*rule Ex-E Conj-E*)+
apply (*rule cut-same* [**where** *A* = *HPair x x' EQ HPair* (*Var sl*) (*Var sl'*)])
apply (*metis Assume LstSeqP-EQ rotate4*, *simp-all*, *clarify*)
apply (*rule Disj-E* [*THEN rotate4*])
apply (*rule Disj-I1*)
apply (*metis Assume AssumeH*(*3*) *Sym thin1 Iff-MP-same* [*OF Conj-cong* [*OF OrdP-cong WRP-cong*] *Assume*])
— auto could be used but is VERY SLOW
apply (*rule Disj-I2*)
apply (*rule Ex-E Conj-EH*)+
apply *simp-all*

apply (*rule Ex-I* [**where** x = *Var m*], *simp*)
apply (*rule Ex-I* [**where** x = *Var n*], *simp*)
apply (*rule Ex-I* [**where** x = *Var sm*], *simp*)
apply (*rule Ex-I* [**where** x = *Var sm'*], *simp*)
apply (*rule Ex-I* [**where** x = *Var sn*], *simp*)
apply (*rule Ex-I* [**where** x = *Var sn'*], *simp*)
apply (*simp add*: *SeqHRP.simps* [*of l - - sl sl' m n sm sm' sn sn'*])
apply (*rule Conj-I*, *blast*)+
— first SeqHRP subgoal
apply (*rule Conj-I*)+
apply (*blast intro*: *LstSeqP-Mem*)
apply (*rule All2-Subset* [*OF Hyp*], *blast*)
apply (*blast intro*!: *SUCC-Subset-Ord LstSeqP-OrdP*, *blast*, *simp*)
— next SeqHRP subgoal
apply (*rule Conj-I*)+
apply (*blast intro*: *LstSeqP-Mem*)
apply (*rule All2-Subset* [*OF Hyp*], *blast*)
apply (*auto intro*!: *SUCC-Subset-Ord LstSeqP-OrdP*)
— finally, the equality pair
apply (*blast intro*: *Trans*)+
done
qed

lemma *SeqHRP-unique*: {*SeqHRP x y s u*, *SeqHRP x y' s' u'*} ⊢ *y' EQ y*
proof −
obtain *i*::*name* **and** *j*::*name* **and** *j'*::*name* **and** *k*::*name* **and** *k'*::*name* **and** *l*::*name*
and *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name* **and** *sm'*::*name* **and** *sn'*::*name*
and *m2*::*name* **and** *n2*::*name* **and** *sm2*::*name* **and** *sn2*::*name* **and** *sm2'*::*name* **and** *sn2'*::*name*
where *atoms*: *atom i* ♯ (*s*,*s'*,*y*,*y'*) *atom j* ♯ (*s*,*s'*,*i*,*x*,*y*,*y'*) *atom j'* ♯ (*s*,*s'*,*i*,*j*,*x*,*y*,*y'*)
atom k ♯ (*s*,*s'*,*x*,*y*,*y'*,*u'*,*i*,*j*,*j'*) *atom k'* ♯ (*s*,*s'*,*x*,*y*,*y'*,*k*,*i*,*j*,*j'*) *atom l* ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*)
atom m ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*,*l*) *atom n* ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*)
atom sm ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*) *atom sn* ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*)
atom sm' ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*) *atom sn'* ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*)
atom m2 ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*,*sn'*) *atom n2* ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*,*sn'*,*m2*)
atom sm2 ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*,*sn'*,*m2*,*n2*) *atom sn2* ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*,*sn'*,*m2*,*n2*,*sm2*)
atom sm2' ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*,*sn'*,*m2*,*n2*,*sm2*,*sn2*) *atom sn2'* ♯ (*s*,*s'*,*i*,*j*,*j'*,*k*,*k'*,*l*,*m*,*n*,*sm*,*sn*,*sm'*,*sn'*,*m2*,*n2*,*sm2*,*sn2*,*sm2'*)
by (*metis obtain-fresh*)
have {*OrdP* (*Var k*)}
⊢ *All i* (*All j* (*All j'* (*All k'* (*SeqHRP* (*Var i*) (*Var j*) *s* (*Var k*) *IMP* (*SeqHRP* (*Var i*) (*Var j'*) *s'* (*Var k'*) *IMP Var j' EQ Var j*)))))
apply (*rule OrdIndH* [**where** *j*=*l*])

using *atoms* **apply** *auto*
apply (*rule Swap*)
apply (*rule cut-same*)
apply (*rule cut1* [*OF SeqHRP-lemma* [*of m Var i Var j s Var k n sm sm′ sn sn′*]], *simp-all*, *blast*)
apply (*rule cut-same*)
apply (*rule cut1* [*OF SeqHRP-lemma* [*of m2 Var i Var j′ s′ Var k′ n2 sm2 sm2′ sn2 sn2′*]], *simp-all*, *blast*)
apply (*rule Disj-EH Conj-EH*)+
— case 1, both are ordinals
apply (*blast intro*: *cut3* [*OF WRP-unique*])
— case 2, *OrdP* (*Var i*) but also a pair
apply (*rule Conj-EH Ex-EH*)+
apply *simp-all*
apply (*rule cut-same* [**where** *A* = *OrdP* (*HPair* (*Var sm*) (*Var sn*))])
apply (*blast intro*: *OrdP-cong* [*OF Hyp*, *THEN Iff-MP-same*], *blast*)
— towards second two cases
apply (*rule Ex-E Disj-EH Conj-EH*)+
— case 3, *OrdP* (*Var i*) but also a pair
apply (*rule cut-same* [**where** *A* = *OrdP* (*HPair* (*Var sm2*) (*Var sn2*))])
apply (*blast intro*: *OrdP-cong* [*OF Hyp*, *THEN Iff-MP-same*], *blast*)
— case 4, two pairs
apply (*rule Ex-E Disj-EH Conj-EH*)+
apply (*rule All-E′* [*OF Hyp*, **where** *x*=*Var m*], *blast*)
apply (*rule All-E′* [*OF Hyp*, **where** *x*=*Var n*], *blast*, *simp-all*)
apply (*rule Disj-EH*, *blast intro*: *thin1 ContraProve*)+
apply (*rule All-E* [**where** *x*=*Var sm*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm2′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var m2*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn*, *THEN rotate2*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn2′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var n2*], *simp*)
apply (*rule cut-same* [**where** *A* = *HPair* (*Var sm*) (*Var sn*) *EQ HPair* (*Var sm2*) (*Var sn2*)])
apply (*blast intro*: *Sym Trans*)
apply (*rule cut-same* [**where** *A* = *SeqHRP* (*Var sn*) (*Var sn2′*) *s′* (*Var n2*)])
apply (*blast intro*: *SeqHRP-cong* [*OF Hyp Refl Refl*, *THEN Iff-MP2-same*])
apply (*rule cut-same* [**where** *A* = *SeqHRP* (*Var sm*) (*Var sm2′*) *s′* (*Var m2*)])
apply (*blast intro*: *SeqHRP-cong* [*OF Hyp Refl Refl*, *THEN Iff-MP2-same*])
apply (*rule Disj-EH*, *blast intro*: *thin1 ContraProve*)+
apply (*blast intro*: *Trans* [*OF Hyp Sym*] *intro*!: *HPair-cong*)
done
hence {*OrdP* (*Var k*)}
$\vdash$ *All j* (*All j′* (*All k′* (*SeqHRP x* (*Var j*) *s* (*Var k*)
IMP (*SeqHRP x* (*Var j′*) *s′* (*Var k′*) *IMP Var j′ EQ Var j*))))
apply (*rule All-D* [**where** *x* = *x*, *THEN cut-same*])
using *atoms* **by** *auto*

hence {*OrdP* (*Var k*)}
⊢ *All j′* (*All k′* (*SeqHRP x y s* (*Var k*) *IMP* (*SeqHRP x* (*Var j′*) *s′* (*Var k′*) *IMP Var j′ EQ y*)))
apply (*rule All-D* [**where** *x* = *y*, *THEN cut-same*])
using *atoms* **by** *auto*
hence {*OrdP* (*Var k*)}
⊢ *All k′* (*SeqHRP x y s* (*Var k*) *IMP* (*SeqHRP x y′ s′* (*Var k′*) *IMP y′ EQ y*))
apply (*rule All-D* [**where** *x* = *y′*, *THEN cut-same*])
using *atoms* **by** *auto*
hence {*OrdP* (*Var k*)} ⊢ *SeqHRP x y s* (*Var k*) *IMP* (*SeqHRP x y′ s′ u′ IMP y′ EQ y*)
apply (*rule All-D* [**where** *x* = *u′*, *THEN cut-same*])
using *atoms* **by** *auto*
hence {*SeqHRP x y s* (*Var k*)} ⊢ *SeqHRP x y s* (*Var k*) *IMP* (*SeqHRP x y′ s′ u′ IMP y′ EQ y*)
by (*metis SeqHRP-imp-OrdP cut1*)
hence {} ⊢ ((*SeqHRP x y s* (*Var k*) *IMP* (*SeqHRP x y′ s′ u′ IMP y′ EQ y*)))(*k*::=*u*)
by (*metis Subst emptyE Assume MP-same Imp-I*)
hence {} ⊢ *SeqHRP x y s u IMP* (*SeqHRP x y′ s′ u′ IMP y′ EQ y*)
using *atoms* **by** *simp*
thus *?thesis*
by (*metis anti-deduction insert-commute*)
qed

theorem *HRP-unique*: {*HRP x y*, *HRP x y′*} ⊢ *y′ EQ y*
proof −
obtain *s*::*name* **and** *s′*::*name* **and** *k*::*name* **and** *k′*::*name*
where *atom s* ♯ (*x*,*y*,*y′*) *atom s′* ♯ (*x*,*y*,*y′*,*s*)
atom k ♯ (*x*,*y*,*y′*,*s*,*s′*) *atom k′* ♯ (*x*,*y*,*y′*,*s*,*s′*,*k*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SeqHRP-unique HRP.simps* [*of s x y k*] *HRP.simps* [*of s′ x y′ k′*])
qed

8.2.4 Finally The Function HF Itself

definition *HF* :: *hf* ⇒ *tm*
where *HF* ≡ *hmemrec* (λ*f z*. *if Ord z then W z else Q-HPair* (*f* (*hfst z*)) (*f* (*hsnd z*)))

lemma *HF-Ord* [*simp*]: *Ord i* ⟹ *HF i* = *W i*
by (*rule trans* [*OF def-hmemrec* [*OF HF-def*]]) *auto*

lemma *HF-pair* [*simp*]: *HF* (*hpair x y*) = *Q-HPair* (*HF x*) (*HF y*)
by (*rule trans* [*OF def-hmemrec* [*OF HF-def*]]) (*auto simp*: *ecut-apply HF-def*)

lemma *SeqHR-hpair*: *SeqHR x1 x3 s1 k1* $\Longrightarrow$ *SeqHR x2 x4 s2 k2* $\Longrightarrow$ $\exists$ *s k*. *SeqHR* $\langle x1,x2\rangle$ (*q-HPair x3 x4*) *s k*
by (*auto simp*: *SeqHR-def intro*: *BuildSeq2-combine*)

lemma *HR-H*: *coding-hf x* $\Longrightarrow$ *HR x* $[\![HF\ x]\!]e$
proof (*induct x rule*: *hmem-rel-induct*)
case (*step x*) **show** *?case*
proof (*cases Ord x*)
case *True* **thus** *?thesis*
by (*auto simp*: *HR-def SeqHR-def Ord-not-hpair WR-iff-eq-W* [**where** *e=e*] *intro*!: *BuildSeq2-exI*)
next
case *False*
then obtain *x1 x2* **where** *x*: $x = \langle x1,x2\rangle$
by (*metis Ord-ord-of coding-hf.simps step.prems*)
then have *x12*: $(x1,\ x) \in$ *hmem-rel* $(x2,\ x) \in$ *hmem-rel*
by (*auto simp*: *hmem-rel-iff-hmem-eclose*)
have *co12*: *coding-hf x1* *coding-hf x2* **using** *False step x*
by (*metis Ord-ord-of coding-hf-hpair*)+
hence *HR x1* $[\![HF\ x1]\!]e$ *HR x2* $[\![HF\ x2]\!]e$
by (*auto simp*: *x12 step*)
thus *?thesis* **using** *x SeqHR-hpair*
by (*auto simp*: *HR-def q-defs*)
qed
qed

Lemma 6.2

lemma *HF-quot-coding-tm*: *coding-tm t* $\Longrightarrow$ *HF* $[\![t]\!]e = \lceil t\rceil$
by (*induct t rule*: *coding-tm.induct*) (*auto*, *simp add*: *HPair-def quot-Eats*)

lemma *HR-quot-fm*: **fixes** *A*::*fm* **shows** *HR* $[\![\lceil A\rceil]\!]e$ $[\![\lceil\lceil A\rceil\rceil]\!]e$
by (*metis HR-H HF-quot-coding-tm coding-tm-hf quot-fm-coding*)

lemma *prove-HRP*: **fixes** *A*::*fm* **shows** $\{\} \vdash$ *HRP* $\lceil A\rceil$ $\lceil\lceil A\rceil\rceil$
by (*auto simp*: *supp-conv-fresh Sigma-fm-imp-thm ground-aux-def ground-fm-aux-def HR-quot-fm*)

8.3 The Function K and Lemma 6.3

nominal-function *KRP* :: *tm* $\Rightarrow$ *tm* $\Rightarrow$ *tm* $\Rightarrow$ *fm*
where *atom y* $\sharp$ (v,x,x') $\Longrightarrow$
KRP v x x' = *Ex y* (*HRP x* (*Var y*) *AND SubstFormP v* (*Var y*) *x x'*)
by (*auto simp*: *eqvt-def KRP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma *KRP-fresh-iff* [*simp*]: $a \sharp$ *KRP v x x'* $\longleftrightarrow a \sharp v \wedge a \sharp x \wedge a \sharp x'$
proof –

obtain y::*name* **where** *atom* $y \sharp (v,x,x')$
  by (*metis obtain-fresh*)
thus *?thesis*
  by *auto*
qed

lemma *KRP-subst* [*simp*]: $(KRP\ v\ x\ x')(i{::=}t) = KRP\ (subst\ i\ t\ v)\ (subst\ i\ t\ x)\ (subst\ i\ t\ x')$
proof −
  obtain y::*name* **where** *atom* $y \sharp (v,x,x',t,i)$
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *KRP.simps* [*of y*])
qed

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

lemma *prove-SubstFormP*: $\{\} \vdash SubstFormP\ \lceil Var\ i\rceil\ \lceil\lceil A\rceil\rceil\ \lceil A\rceil\ \lceil A(i{::=}\lceil A\rceil)\rceil$
 by (*auto simp*: *supp-conv-fresh Sigma-fm-imp-thm ground-aux-def SubstForm-quot*)

lemma *prove-KRP*: $\{\} \vdash KRP\ \lceil Var\ i\rceil\ \lceil A\rceil\ \lceil A(i{::=}\lceil A\rceil)\rceil$
  by (*auto simp*: *KRP.simps* [*of y*]
      intro!: *Ex-I* [**where** $x{=}\lceil\lceil A\rceil\rceil$] *prove-HRP prove-SubstFormP*)

lemma *KRP-unique*: $\{KRP\ v\ x\ y,\ KRP\ v\ x\ y'\} \vdash y'\ EQ\ y$
proof −
  obtain u::*name* **and** u'::*name* **where** *atom* $u \sharp (v,x,y,y')$ *atom* $u' \sharp (v,x,y,y',u)$
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *KRP.simps* [*of u v x y*] *KRP.simps* [*of u' v x y'*]
        intro: *SubstFormP-cong* [*THEN Iff-MP2-same*]
          SubstFormP-unique [*THEN cut2*] *HRP-unique* [*THEN cut2*])
qed

lemma *KRP-subst-fm*: $\{KRP\ \lceil Var\ i\rceil\ \lceil\beta\rceil\ (Var\ j)\} \vdash Var\ j\ EQ\ \lceil\beta(i{::=}\lceil\beta\rceil)\rceil$
  by (*metis KRP-unique cut0 prove-KRP*)

8.4 The Diagonal Lemma and Gdel's Theorem

lemma *diagonal*:
  obtains δ **where** $\{\} \vdash \delta\ IFF\ \alpha(i{::=}\lceil\delta\rceil)$ $supp\ \delta = supp\ \alpha - \{atom\ i\}$
proof −
  obtain k::*name* **and** j::*name*
    where *atoms*: *atom* $k \sharp (i,j,\alpha)$ *atom* $j \sharp (i,\alpha)$
    by (*metis obtain-fresh*)
  def $\beta \equiv Ex\ j\ (KRP\ \lceil Var\ i\rceil\ (Var\ i)\ (Var\ j)\ AND\ \alpha(i ::= Var\ j))$
  hence *1*: $\{\} \vdash \beta(i ::= \lceil\beta\rceil)\ IFF\ (Ex\ j\ (KRP\ \lceil Var\ i\rceil\ (Var\ i)\ (Var\ j)\ AND\ \alpha(i ::= Var\ j)))(i ::= \lceil\beta\rceil)$
    by (*metis Iff-refl*)

have *2*: $\{\} \vdash (Ex\ j\ (KRP\ \lceil Var\ i\rceil\ (Var\ i)\ (Var\ j)\ AND\ \alpha(i ::= Var\ j)))(i ::= \lceil\beta\rceil)\ IFF$
$Ex\ j\ (Var\ j\ EQ\ \lceil\beta(i::=\lceil\beta\rceil)\rceil\ AND\ \alpha(i::=Var\ j))$
using *atoms*
apply (*auto intro*!: *Ex-cong Conj-cong KRP-subst-fm*)
apply (*rule Iff-MP-same* [*OF Var-Eq-subst-Iff*])
apply (*auto intro*: *prove-KRP thin0*)
done
have *3*: $\{\} \vdash Ex\ j\ (Var\ j\ EQ\ \lceil\beta(i::=\lceil\beta\rceil)\rceil\ AND\ \alpha(i::=Var\ j))\ IFF\ \alpha(i::=\lceil\beta(i::=\lceil\beta\rceil)\rceil)$
using *atoms*
apply *auto*
apply (*rule cut-same* [*OF Iff-MP2-same* [*OF Var-Eq-subst-Iff AssumeH*(*2*)]])
apply (*auto intro*: *Ex-I* [**where** $x=\lceil\beta(i::=\lceil\beta\rceil)\rceil$])
done
have $supp\ (\beta(i ::= \lceil\beta\rceil)) = supp\ \alpha - \{atom\ i\}$ **using** *atoms*
by (*auto simp*: *fresh-at-base ground-fm-aux-def* β*-def supp-conv-fresh*)
thus *?thesis* **using** *atoms*
by (*metis that 1 2 3 Iff-trans*)
qed

Gdel's first incompleteness theorem: If consistent, our theory is incomplete.

theorem *Goedel-I*:
assumes $\neg \{\} \vdash Fls$
obtains δ **where** $\{\} \vdash \delta\ IFF\ Neg\ (PfP\ \lceil\delta\rceil)$ $\neg \{\} \vdash \delta$ $\neg \{\} \vdash Neg\ \delta$
eval-fm e δ *ground-fm* δ
proof −
fix *i*::*name*
obtain δ **where** $\{\} \vdash \delta\ IFF\ Neg\ ((PfP\ (Var\ i))(i::=\lceil\delta\rceil))$
and *suppd*: $supp\ \delta = supp\ (Neg\ (PfP\ (Var\ i))) - \{atom\ i\}$
by (*metis SyntaxN.Neg diagonal*)
hence *diag*: $\{\} \vdash \delta\ IFF\ Neg\ (PfP\ \lceil\delta\rceil)$
by *simp*
hence *np*: $\neg \{\} \vdash \delta$
by (*metis assms Iff-MP-same Neg-D proved-iff-proved-PfP*)
hence *npn*: $\neg \{\} \vdash Neg\ \delta$ **using** *diag*
by (*metis Iff-MP-same NegNeg-D Neg-cong proved-iff-proved-PfP*)
moreover have *eval-fm e* δ **using** *hfthm-sound* [**where** *e*=*e*, *OF diag*]
by *simp* (*metis Pf-quot-imp-is-proved np*)
moreover have *ground-fm* δ **using** *suppd*
by (*simp add*: *supp-conv-fresh ground-fm-aux-def subset-eq*) (*metis fresh-ineq-at-base*)
ultimately show *?thesis*
by (*metis diag np npn that*)
qed

end

Chapter 9

Syntactic Preliminaries for the Second Incompleteness Theorem

theory *II-Prelims*
imports *Pf-Predicates*
begin

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

lemma *VarP-Var* [*intro*]: $H \vdash$ *VarP* $\lceil$*Var i*$\rceil$
proof −
  have {} $\vdash$ *VarP* $\lceil$*Var i*$\rceil$
   by (*auto simp*: *Sigma-fm-imp-thm* [*OF VarP-sf*] *ground-fm-aux-def supp-conv-fresh*)
  thus *?thesis*
    by (*rule thin0*)
qed

lemma *VarP-neq-IndP*: {*t EQ v*, *VarP v*, *IndP t*} $\vdash$ *Fls*
proof −
  obtain *m*::*name* **where** *atom m* $\sharp$ (*t*,*v*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    apply (*auto simp*: *VarP-def IndP.simps* [*of m*])
    apply (*rule cut-same* [*of - OrdP* (*Q-Ind* (*Var m*))])
    apply (*blast intro*: *Sym Trans OrdP-cong* [*THEN Iff-MP-same*])
    by (*metis OrdP-HPairE*)
qed

lemma *OrdP-ORD-OF* [*intro*]: $H \vdash$ *OrdP* (*ORD-OF n*)
proof −
  have {} $\vdash$ *OrdP* (*ORD-OF n*)
    by (*induct n*) (*auto simp*: *OrdP-SUCC-I*)

thus *?thesis*
by (*rule thin0*)
qed

lemma *Mem-HFun-Sigma-OrdP*: {*HPair t u IN f*, *HFun-Sigma f*} ⊢ *OrdP t*
proof −
obtain *x*::*name* **and** *y*::*name* **and** *z*::*name* **and** *x'*::*name* **and** *y'*::*name* **and** *z'*::*name*
where *atom z* ♯ (*f*,*t*,*u*,*z'*,*x*,*y*,*x'*,*y'*) *atom z'* ♯ (*f*,*t*,*u*,*x*,*y*,*x'*,*y'*)
atom x ♯ (*f*,*t*,*u*,*y*,*x'*,*y'*) *atom y* ♯ (*f*,*t*,*u*,*x'*,*y'*)
atom x' ♯ (*f*,*t*,*u*,*y'*) *atom y'* ♯ (*f*,*t*,*u*)
by (*metis obtain-fresh*)
thus *?thesis*
apply (*simp add*: *HFun-Sigma.simps* [*of z f z' x y x' y'*])
apply (*rule All2-E* [**where** *x*=*HPair t u*, *THEN rotate2*], *auto*)
apply (*rule All2-E* [**where** *x*=*HPair t u*], *auto intro*: *OrdP-cong* [*THEN Iff-MP2-same*])
done
qed

9.1 NotInDom

nominal-function *NotInDom* :: *tm* ⇒ *tm* ⇒ *fm*
where *atom z* ♯ (*t*, *r*) ⟹ *NotInDom t r* = *All z* (*Neg* (*HPair t* (*Var z*) *IN r*))
by (*auto simp*: *eqvt-def NotInDom-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma *NotInDom-fresh-iff* [*simp*]: *a* ♯ *NotInDom t r* ⟷ *a* ♯ (*t*, *r*)
proof −
obtain *j*::*name* **where** *atom j* ♯ (*t*,*r*)
by (*rule obtain-fresh*)
thus *?thesis*
by *auto*
qed

lemma *subst-fm-NotInDom* [*simp*]: (*NotInDom t r*)(*i*::=*x*) = *NotInDom* (*subst i x t*) (*subst i x r*)
proof −
obtain *j*::*name* **where** *atom j* ♯ (*i*,*x*,*t*,*r*)
by (*rule obtain-fresh*)
thus *?thesis*
by (*auto simp*: *NotInDom.simps* [*of j*])
qed

lemma *NotInDom-cong*: *H* ⊢ *t EQ t'* ⟹ *H* ⊢ *r EQ r'* ⟹ *H* ⊢ *NotInDom t r IFF NotInDom t' r'*
by (*rule P2-cong*) *auto*

lemma *NotInDom-Zero*: $H \vdash$ *NotInDom t Zero*
proof –
  obtain *z::name* **where** *atom z* $\sharp$ *t*
    by (*metis obtain-fresh*)
  hence {} $\vdash$ *NotInDom t Zero*
    by (*auto simp*: *fresh-Pair*)
  thus *?thesis*
    by (*rule thin0*)
qed

lemma *NotInDom-Fls*: {*HPair d d′ IN r*, *NotInDom d r*} $\vdash$ *A*
proof –
  obtain *z::name* **where** *atom z* $\sharp$ (*d*,*r*)
    by (*metis obtain-fresh*)
  hence {*HPair d d′ IN r*, *NotInDom d r*} $\vdash$ *Fls*
    by (*auto intro*!: *Ex-I* [**where** *x=d′*])
  thus *?thesis*
    by (*metis ExFalso*)
qed

lemma *NotInDom-Contra*: $H \vdash$ *NotInDom d r* $\Longrightarrow$ $H \vdash$ *HPair x y IN r* $\Longrightarrow$ *insert* (*x EQ d*) $H \vdash A$
by (*rule NotInDom-Fls* [*THEN cut2*, *THEN ExFalso*])
  (*auto intro*: *thin1 NotInDom-cong* [*OF Assume Refl*, *THEN Iff-MP2-same*])

9.2 Restriction of a Sequence to a Domain

nominal-function *RestrictedP* :: *tm* $\Rightarrow$ *tm* $\Rightarrow$ *tm* $\Rightarrow$ *fm*
  where ⟦*atom x* $\sharp$ (*y*,*f*,*k*,*g*); *atom y* $\sharp$ (*f*,*k*,*g*)⟧ $\Longrightarrow$
    RestrictedP f k g =
      g SUBS f AND
      All x (*All y* (*HPair* (*Var x*) (*Var y*) *IN g IFF*
                (*Var x*) *IN k AND HPair* (*Var x*) (*Var y*) *IN f*))
by (*auto simp*: *eqvt-def RestrictedP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma *RestrictedP-fresh-iff* [*simp*]: *a* $\sharp$ *RestrictedP f k g* $\longleftrightarrow$ *a* $\sharp$ *f* $\wedge$ *a* $\sharp$ *k* $\wedge$ *a* $\sharp$ *g*
proof –
  obtain *x::name* **and** *y::name* **where** *atom x* $\sharp$ (*y*,*f*,*k*,*g*) *atom y* $\sharp$ (*f*,*k*,*g*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by *auto*
qed

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 −
obtain x::*name* **and** y::*name* **where** *atom* $x \sharp (y,f,k,g,i,u)$ *atom* $y \sharp (f,k,g,i,u)$
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *RestrictedP.simps* [*of x y*])
qed

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* $f'\ A'\ g'$
by (*rule P3-cong*) *auto*

lemma *RestrictedP-Zero*: $H \vdash$ *RestrictedP Zero k Zero*
proof −
obtain x::*name* **and** y::*name* **where** *atom* $x \sharp (y,k)$ *atom* $y \sharp (k)$
by (*metis obtain-fresh*)
hence $\{\} \vdash$ *RestrictedP Zero k Zero*
by (*auto simp*: *RestrictedP.simps* [*of x y*])
thus *?thesis*
by (*rule thin0*)
qed

lemma *RestrictedP-Mem*: { *RestrictedP s k* s', *HPair a b IN s*, *a IN k* } $\vdash$ *HPair a b IN* s'
proof −
obtain x::*name* **and** y::*name* **where** *atom* $x \sharp (y,s,k,s',a,b)$ *atom* $y \sharp (s,k,s',a,b)$
by (*metis obtain-fresh*)
thus *?thesis*
apply (*auto simp*: *RestrictedP.simps* [*of x y*])
apply (*rule All-E* [**where** x=a, *THEN rotate2*], *auto*)
apply (*rule All-E* [**where** x=b], *auto intro*: *Iff-E2*)
done
qed

lemma *RestrictedP-imp-Subset*: {*RestrictedP s k* s'} $\vdash$ s' *SUBS s*
proof −
obtain x::*name* **and** y::*name* **where** *atom* $x \sharp (y,s,k,s')$ *atom* $y \sharp (s,k,s')$
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *RestrictedP.simps* [*of x y*])
qed

lemma *RestrictedP-Mem2*:
{ *RestrictedP s k* s', *HPair a b IN* s' } $\vdash$ *HPair a b IN s AND a IN k*
proof −
obtain x::*name* **and** y::*name* **where** *atom* $x \sharp (y,s,k,s',a,b)$ *atom* $y \sharp (s,k,s',a,b)$
by (*metis obtain-fresh*)
thus *?thesis*
apply (*auto simp*: *RestrictedP.simps* [*of x y*] *intro*: *Subset-D*)

apply (*rule All-E* [**where** *x*=*a*, *THEN rotate2*], *auto*)
apply (*rule All-E* [**where** *x*=*b*], *auto intro*: *Iff-E1*)
done
qed

lemma *RestrictedP-Mem-D*: $H \vdash$ *RestrictedP s k t* $\Longrightarrow H \vdash$ *a IN t* $\Longrightarrow$ *insert* (*a IN s*) $H \vdash A \Longrightarrow H \vdash A$
by (*metis RestrictedP-imp-Subset Subset-E cut1*)

lemma *RestrictedP-Eats*:
{ *RestrictedP s k s'*, *a IN k* } $\vdash$ *RestrictedP* (*Eats s* (*HPair a b*)) *k* (*Eats s'* (*HPair a b*))
lemma *exists-RestrictedP*:
assumes *s*: *atom s* $\sharp$ (*f*,*k*)
shows $H \vdash$ *Ex s* (*RestrictedP f k* (*Var s*))
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*)) $H \vdash A \Longrightarrow H \vdash A$
apply (*rule cut-same* [*OF exists-RestrictedP* [*of s*]])
using *assms* **apply** *auto*
done

lemma *RestrictedP-NotInDom*: { *RestrictedP s k s'*, *Neg* (*j IN k*) } $\vdash$ *NotInDom j s'*
proof –
obtain *x*::*name* **and** *y*::*name* **and** *z*::*name*
where *atom x* $\sharp$ (*y*,*s*,*j*,*k*,*s'*) *atom y* $\sharp$ (*s*,*j*,*k*,*s'*) *atom z* $\sharp$ (*s*,*j*,*k*,*s'*)
by (*metis obtain-fresh*)
thus *?thesis*
apply (*auto simp*: *RestrictedP.simps* [*of x y*] *NotInDom.simps* [*of z*])
apply (*rule All-E* [**where** *x*=*j*, *THEN rotate3*], *auto*)
apply (*rule All-E*, *auto intro*: *Conj-E1 Iff-E1*)
done
qed

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

9.3 Applications to LstSeqP

lemma *HFun-Sigma-Eats*:
assumes $H \vdash$ *HFun-Sigma r* $H \vdash$ *NotInDom d r* $H \vdash$ *OrdP d*
shows $H \vdash$ *HFun-Sigma* (*Eats r* (*HPair d d'*))
lemma *HFun-Sigma-single* [*iff*]: $H \vdash$ *OrdP d* $\Longrightarrow H \vdash$ *HFun-Sigma* (*Eats Zero* (*HPair d d'*))
by (*metis HFun-Sigma-Eats HFun-Sigma-Zero NotInDom-Zero*)

lemma *LstSeqP-single* [*iff*]: $H \vdash$ *LstSeqP* (*Eats Zero* (*HPair Zero x*)) *Zero x*
by (*auto simp*: *LstSeqP.simps intro*!: *OrdP-SUCC-I HDomain-Incl-Eats-I Mem-Eats-I2*)

lemma *NotInDom-LstSeqP-Eats*:
{ *NotInDom* (*SUCC k*) *s*, *LstSeqP s k y* } ⊢ *LstSeqP* (*Eats s* (*HPair* (*SUCC k*) *z*)) (*SUCC k*) *z*
by (*auto simp*: *LstSeqP.simps intro*: *HDomain-Incl-Eats-I Mem-Eats-I2 OrdP-SUCC-I HFun-Sigma-Eats*)

lemma *RestrictedP-HDomain-Incl*: {*HDomain-Incl s k*, *RestrictedP s k s'*} ⊢ *HDomain-Incl s' k*
proof −
  obtain *u*::*name* **and** *v*::*name* **and** *x*::*name* **and** *y*::*name* **and** *z*::*name*
    where *atom u* ♯ (*v*,*s*,*k*,*s'*) *atom v* ♯ (*s*,*k*,*s'*)
      atom x ♯ (*s*,*k*,*s'*,*u*,*v*,*y*,*z*) *atom y* ♯ (*s*,*k*,*s'*,*u*,*v*,*z*) *atom z* ♯ (*s*,*k*,*s'*,*u*,*v*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    apply (*auto simp*: *HDomain-Incl.simps* [*of x - - y z*])
    apply (*rule Ex-I* [**where** *x*=*Var x*], *auto*)
    apply (*rule Ex-I* [**where** *x*=*Var y*], *auto*)
    apply (*rule Ex-I* [**where** *x*=*Var z*], *simp*)
    apply (*rule Var-Eq-subst-Iff* [*THEN Iff-MP-same*, *THEN rotate2*])
    apply (*auto simp*: *RestrictedP.simps* [*of u v*])
    apply (*rule All-E* [**where** *x*=*Var x*, *THEN rotate2*], *auto*)
    apply (*rule All-E* [**where** *x*=*Var y*])
    apply (*auto intro*: *Iff-E ContraProve Mem-cong* [*THEN Iff-MP-same*])
    done
qed

lemma *RestrictedP-HFun-Sigma*: {*HFun-Sigma s*, *RestrictedP s k s'*} ⊢ *HFun-Sigma s'*
  by (*metis Assume RestrictedP-imp-Subset Subset-HFun-Sigma rcut2*)

lemma *RestrictedP-LstSeqP*:
{ *RestrictedP s* (*SUCC k*) *s'*, *LstSeqP s k y* } ⊢ *LstSeqP s' k y*
  by (*auto simp*: *LstSeqP.simps*
      intro: *Mem-Neg-refl cut2* [*OF RestrictedP-HDomain-Incl*]
        cut2 [*OF RestrictedP-HFun-Sigma*] *cut3* [*OF RestrictedP-Mem*])

lemma *RestrictedP-LstSeqP-Eats*:
{ *RestrictedP s* (*SUCC k*) *s'*, *LstSeqP s k y* }
⊢ *LstSeqP* (*Eats s'* (*HPair* (*SUCC k*) *z*)) (*SUCC k*) *z*
by (*blast intro*: *Mem-Neg-refl cut2* [*OF NotInDom-LstSeqP-Eats*]
      cut2 [*OF RestrictedP-NotInDom*] *cut2* [*OF RestrictedP-LstSeqP*])

9.4 Ordinal Addition

9.4.1 Predicate form, defined on sequences

nominal-function *SeqHaddP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*

where ⟦*atom l* ♯ (*sl*,*s*,*k*,*j*); *atom sl* ♯ (*s*,*j*)⟧ ⟹
SeqHaddP s j k y = *LstSeqP s k y AND*
HPair Zero j IN s AND
All2 l k (*Ex sl* (*HPair* (*Var l*) (*Var sl*) *IN s AND*
HPair (*SUCC* (*Var l*)) (*SUCC* (*Var sl*)) *IN s*))
by (*auto simp*: *eqvt-def SeqHaddP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma *SeqHaddP-fresh-iff* [*simp*]: *a* ♯ *SeqHaddP s j k y* ⟷ *a* ♯ *s* ∧ *a* ♯ *j* ∧ *a* ♯ *k* ∧ *a* ♯ *y*
proof −
obtain *l*::*name* **and** *sl*::*name* **where** *atom l* ♯ (*sl*,*s*,*k*,*j*) *atom sl* ♯ (*s*,*j*)
by (*metis obtain-fresh*)
thus *?thesis*
by *force*
qed

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 −
obtain *l*::*name* **and** *sl*::*name* **where** *atom l* ♯ (*s*,*k*,*j*,*sl*,*t*,*i*) *atom sl* ♯ (*s*,*k*,*j*,*t*,*i*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SeqHaddP.simps* [**where** *l*=*l* **and** *sl*=*sl*])
qed

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*)
by (*auto simp*: *eqvt-def HaddP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma *HaddP-fresh-iff* [*simp*]: *a* ♯ *HaddP x y z* ⟷ *a* ♯ *x* ∧ *a* ♯ *y* ∧ *a* ♯ *z*
proof −
obtain *s*::*name* **where** *atom s* ♯ (*x*,*y*,*z*)
by (*metis obtain-fresh*)
thus *?thesis*
by *force*
qed

lemma *HaddP-subst* [*simp*]: (*HaddP x y z*)(*i*::=*t*) = *HaddP* (*subst i t x*) (*subst i t y*) (*subst i t z*)

proof −
  obtain *s::name* **where** *atom s* ♯ (*x*,*y*,*z*,*t*,*i*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *HaddP.simps* [*of s*])
qed

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′*
  by (*rule P3-cong*) *auto*

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

lemma *HaddP-Zero2*: *H* ⊢ *HaddP x Zero x*
proof −
  obtain *s::name* **and** *l::name* **and** *sl::name* **where** *atom l* ♯ (*sl*,*s*,*x*) *atom sl* ♯ (*s*,*x*) *atom s* ♯ *x*
    by (*metis obtain-fresh*)
  hence {} ⊢ *HaddP x Zero x*
    by (*auto simp*: *HaddP.simps* [*of s*] *SeqHaddP.simps* [*of l sl*]
        intro!: *Mem-Eats-I2 Ex-I* [**where** *x*=*Eats Zero* (*HPair Zero x*)])
  thus *?thesis*
    by (*rule thin0*)
qed

lemma *HaddP-imp-OrdP*: {*HaddP x y z*} ⊢ *OrdP y*
proof −
  obtain *s::name* **and** *l::name* **and** *sl::name*
    where *atom l* ♯ (*sl*,*s*,*x*,*y*,*z*) *atom sl* ♯ (*s*,*x*,*y*,*z*) *atom s* ♯ (*x*,*y*,*z*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *HaddP.simps* [*of s*] *SeqHaddP.simps* [*of l sl*] *LstSeqP.simps*)
qed

lemma *HaddP-SUCC2*: {*HaddP x y z*} ⊢ *HaddP x* (*SUCC y*) (*SUCC z*)

9.4.2 Proving that these relations are functions

lemma *SeqHaddP-Zero-E*: {*SeqHaddP s w Zero z*} ⊢ *w EQ z*
proof −
  obtain *l::name* **and** *sl::name* **where** *atom l* ♯ (*s*,*w*,*z*,*sl*) *atom sl* ♯ (*s*,*w*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *SeqHaddP.simps* [*of l sl*] *LstSeqP.simps intro*: *HFun-Sigma-E*)
qed

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 −
obtain *l*::*name* **and** *sl*::*name* **where** *atom l* ♯ (*s*,*j*,*k*,*y*,*y′*,*sl*) *atom sl* ♯ (*s*,*j*,*k*,*y*,*y′*)
by (*metis obtain-fresh*)
thus *?thesis* **using** *y′*
apply (*auto simp*: *SeqHaddP.simps* [**where** *s*=*s* **and** *l*=*l* **and** *sl*=*sl*])
apply (*rule All2-SUCC-E′* [**where** *t*=*k*, *THEN rotate2*], *auto*)
apply (*auto intro*!: *Ex-I* [**where** *x*=*Var sl*])
apply (*blast intro*: *LstSeqP-SUCC*) — showing *SeqHaddP s j k* (*Var sl*)
apply (*blast intro*: *LstSeqP-EQ*)
done
qed

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′*))
by (*metis SeqHaddP-SUCC-lemma* [*THEN cut1*] *assms*)

lemma *SeqHaddP-unique*: {*OrdP x*, *SeqHaddP s w x y*, *SeqHaddP s′ w x y′*} ⊢ *y′* *EQ y*
lemma *HaddP-unique*: {*HaddP w x y*, *HaddP w x y′*} ⊢ *y′ EQ y*
proof −
obtain *s*::*name* **and** *s′*::*name* **where** *atom s* ♯ (*w*,*x*,*y*,*y′*) *atom s′* ♯ (*w*,*x*,*y*,*y′*,*s*)
by (*metis obtain-fresh*)
hence {*OrdP x*, *HaddP w x y*, *HaddP w x y′*} ⊢ *y′ EQ y*
by (*auto simp*: *HaddP.simps* [*of s - - y*] *HaddP.simps* [*of s′ - - y′*]
intro: *SeqHaddP-unique* [*THEN cut3*])
thus *?thesis*
by (*metis HaddP-imp-OrdP cut-same thin1*)
qed

lemma *HaddP-Zero1*: **assumes** *H* ⊢ *OrdP x* **shows** *H* ⊢ *HaddP Zero x x*
proof −
fix *k*::*name*
have { *OrdP* (*Var k*) } ⊢ *HaddP Zero* (*Var k*) (*Var k*)
by (*rule OrdInd2H* [**where** *i*=*k*]) (*auto intro*: *HaddP-Zero2 HaddP-SUCC2* [*THEN cut1*])
hence {} ⊢ *OrdP* (*Var k*) *IMP HaddP Zero* (*Var k*) (*Var k*)
by (*metis Imp-I*)
hence {} ⊢ (*OrdP* (*Var k*) *IMP HaddP Zero* (*Var k*) (*Var k*))(*k*::=*x*)
by (*rule Subst*) *auto*
hence {} ⊢ *OrdP x IMP HaddP Zero x x*
by *simp*
thus *?thesis* **using** *assms*
by (*metis MP-same thin0*)
qed

lemma *HaddP-Zero-D1*: *insert* (*HaddP Zero x y*) *H* ⊢ *x EQ y*
by (*metis Assume HaddP-imp-OrdP HaddP-Zero1 HaddP-unique* [*THEN cut2*]

rcut1)

lemma *HaddP-Zero-D2*: *insert* (*HaddP x Zero y*) *H* ⊢ *x EQ y*
  by (*metis Assume HaddP-Zero2 HaddP-unique* [*THEN cut2*])

lemma *HaddP-SUCC-Ex2*:
  assumes *H* ⊢ *HaddP x* (*SUCC y*) *z atom z′* ♯ (*x*,*y*,*z*)
    shows *H* ⊢ *Ex z′* (*HaddP x y* (*Var z′*) *AND z EQ SUCC* (*Var z′*))
proof −
  obtain *s*::*name* **and** *s′*::*name* **where** *atom s* ♯ (*x*,*y*,*z*,*z′*) *atom s′* ♯ (*x*,*y*,*z*,*z′*,*s*)
    by (*metis obtain-fresh*)
  hence { *HaddP x* (*SUCC y*) *z* } ⊢ *Ex z′* (*HaddP x y* (*Var z′*) *AND z EQ SUCC* (*Var z′*))
    using *assms*
    apply (*auto simp*: *HaddP.simps* [*of s* - -] *HaddP.simps* [*of s′* - -])
    apply (*rule cut-same* [*OF SeqHaddP-SUCC-lemma* [*of z′*]], *auto*)
    apply (*rule Ex-I*, *auto*)+
    done
  thus *?thesis*
    by (*metis assms*(*1*) *cut1*)
qed

lemma *HaddP-SUCC1*: { *HaddP x y z* } ⊢ *HaddP* (*SUCC x*) *y* (*SUCC z*)
lemma *HaddP-commute*: {*HaddP x y z*, *OrdP x*} ⊢ *HaddP y x z*
lemma *HaddP-SUCC-Ex1*:
  assumes *atom i* ♯ (*x*,*y*,*z*)
    shows *insert* (*HaddP* (*SUCC x*) *y z*) (*insert* (*OrdP x*) *H*)
        ⊢ *Ex i* (*HaddP x y* (*Var i*) *AND z EQ SUCC* (*Var i*))
proof −
  have { *HaddP* (*SUCC x*) *y z*, *OrdP x* } ⊢ *Ex i* (*HaddP x y* (*Var i*) *AND z EQ SUCC* (*Var i*))
    apply (*rule cut-same* [*OF HaddP-commute* [*THEN cut2*]])
    apply (*blast intro*: *OrdP-SUCC-I*)+
    apply (*rule cut-same* [*OF HaddP-SUCC-Ex2* [**where** *z′*=*i*]], *blast*)
    using *assms* **apply** *auto*
    apply (*auto intro*!: *Ex-I* [**where** *x*=*Var i*])
   by (*metis AssumeH*(*2*) *HaddP-commute* [*THEN cut2*] *HaddP-imp-OrdP rotate2 thin1*)
  thus *?thesis*
    by (*metis Assume AssumeH*(*2*) *cut2*)
qed

lemma *HaddP-inv2*: {*HaddP x y z*, *HaddP x y′ z*, *OrdP x*} ⊢ *y′ EQ y*
lemma *Mem-imp-subtract*:
lemma *HaddP-OrdP*:
  assumes *H* ⊢ *HaddP x y z H* ⊢ *OrdP x* **shows** *H* ⊢ *OrdP z*
lemma *HaddP-Mem-cancel-left*:
  assumes *H* ⊢ *HaddP x y′ z′ H* ⊢ *HaddP x y z H* ⊢ *OrdP x*
    shows *H* ⊢ *z′ IN z IFF y′ IN y*

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 –
have $H \vdash$ *OrdP* x'
by (*metis Ord-IN-Ord assms*(*3*) *assms*(*4*))
hence $H \vdash$ *HaddP* y x' z' $H \vdash$ *HaddP* y x z
by (*blast intro*: *assms HaddP-commute* [*THEN cut2*])+
thus *?thesis*
by (*blast intro*: *assms HaddP-imp-OrdP* [*THEN cut1*] *HaddP-Mem-cancel-left* [*THEN Iff-MP2-same*])
qed

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*) $H \vdash A$
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 –
obtain *i*::*name* **and** *j*::*name* **and** *k*::*name*
where *atoms*: *atom i* $\sharp$ (*x*,*y*,*z*) *atom j* $\sharp$ (*i*,*x*,*y*,*z*) *atom k* $\sharp$ (*i*,*j*,*x*,*y*,*z*)
by (*metis obtain-fresh*)
have {*OrdP* (*Var i*)} $\vdash$ *All j* (*HaddP* (*Var i*) *y* (*Var j*) *IMP Neg* ((*Var j*) *IN* (*Var i*)))
(**is** - $\vdash$ *?scheme*)
proof (*rule OrdInd2H*)
show {} $\vdash$ *?scheme*(*i*::=*Zero*)
using *atoms* **by** *auto*
next
show {} $\vdash$ *All i* (*OrdP* (*Var i*) *IMP ?scheme IMP ?scheme*(*i*::=*SUCC* (*Var i*)))
using *atoms* **apply** *auto*
apply (*rule cut-same* [*OF HaddP-SUCC-Ex1* [*of k Var i y Var j*, *THEN cut2*]], *auto*)
apply (*rule Ex-I* [**where** *x*=*Var k*], *auto*)
apply (*blast intro*: *OrdP-IN-SUCC-D Mem-cong* [*OF - Refl*, *THEN Iff-MP-same*])
done
qed
hence {*OrdP* (*Var i*)} $\vdash$ (*HaddP* (*Var i*) *y* (*Var j*) *IMP Neg* ((*Var j*) *IN* (*Var i*)))(*j*::=*z*)
by (*metis All-D*)
hence {} $\vdash$ *OrdP* (*Var i*) *IMP HaddP* (*Var i*) *y z IMP Neg* (*z IN* (*Var i*))
using *atoms* **by** *simp* (*metis Imp-I*)
hence {} $\vdash$ (*OrdP* (*Var i*) *IMP HaddP* (*Var i*) *y z IMP Neg* (*z IN* (*Var*

```
i)))(i::=x)
    by (metis Subst emptyE)
  thus ?thesis
    using atoms by simp (metis MP-same MP-null Neg-D assms)
qed

lemma exists-HaddP:
  assumes H ⊢ OrdP y atom j ♯ (x,y)
    shows H ⊢ Ex j (HaddP x y (Var j))
proof −
  obtain i::name
   where atoms: atom i ♯ (j,x,y)
    by (metis obtain-fresh)
  have {OrdP (Var i)} ⊢ Ex j (HaddP x (Var i) (Var j))
       (is - ⊢ ?scheme)
    proof (rule OrdInd2H)
      show {} ⊢ ?scheme(i::=Zero)
        using atoms assms
        by (force intro!: Ex-I [where x=x] HaddP-Zero2)
    next
      show {} ⊢ All i (OrdP (Var i) IMP ?scheme IMP ?scheme(i::=SUCC (Var
i)))
        using atoms assms
        apply auto
        apply (auto intro!: Ex-I [where x=SUCC (Var j)] HaddP-SUCC2)
        apply (metis HaddP-SUCC2 insert-commute thin1)
        done
    qed
  hence {} ⊢ OrdP (Var i) IMP Ex j (HaddP x (Var i) (Var j))
    by (metis Imp-I)
  hence {} ⊢ (OrdP (Var i) IMP Ex j (HaddP x (Var i) (Var j)))(i::=y)
    using atoms by (force intro!: Subst)
  thus ?thesis
    using atoms assms by simp (metis MP-null assms(1))
qed

lemma HaddP-Mem-I:
  assumes H ⊢ HaddP x y z H ⊢ OrdP x shows H ⊢ x IN SUCC z
proof −
  have {HaddP x y z, OrdP x} ⊢ x IN SUCC z
    apply (rule OrdP-linear [of - x SUCC z])
    apply (auto intro: OrdP-SUCC-I HaddP-OrdP)
    apply (rule HaddP-Mem-contra, blast)
     apply (metis Assume Mem-SUCC-I2 OrdP-IN-SUCC-D Sym-L thin1 thin2,
blast)
    apply (blast intro: HaddP-Mem-contra Mem-SUCC-Refl OrdP-Trans)
    done
  thus ?thesis
    by (rule cut2) (auto intro: assms)
```

qed

9.5 A Shifted Sequence

nominal-function *ShiftP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom x* ♯ (x',*y*,*z*,*f*,*del*,*g*,*k*); *atom* x' ♯ (*y*,*z*,*f*,*del*,*g*,*k*); *atom y* ♯ (*z*,*f*,*del*,*g*,*k*); *atom z* ♯ (*f*,*del*,*g*,*k*)⟧ ⟹
ShiftP f k del g =
All z (*Var z IN g IFF*
(*Ex x* (*Ex* x' (*Ex y* ((*Var z*) *EQ HPair* (*Var* x') (*Var y*) *AND*
HaddP del (*Var x*) (*Var* x') *AND*
HPair (*Var x*) (*Var y*) *IN f AND Var x IN k*)))))
by (*auto simp*: *eqvt-def ShiftP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma *ShiftP-fresh-iff* [*simp*]: *a* ♯ *ShiftP f k del g* ⟷ *a* ♯ *f* ∧ *a* ♯ *k* ∧ *a* ♯ *del* ∧ *a* ♯ *g*
proof −
obtain *x*::*name* **and** x'::*name* **and** *y*::*name* **and** *z*::*name*
where *atom x* ♯ (x',*y*,*z*,*f*,*del*,*g*,*k*) *atom* x' ♯ (*y*,*z*,*f*,*del*,*g*,*k*)
atom y ♯ (*z*,*f*,*del*,*g*,*k*) *atom z* ♯ (*f*,*del*,*g*,*k*)
by (*metis obtain-fresh*)
thus *?thesis*
by *auto*
qed

lemma *subst-fm-ShiftP* [*simp*]:
(*ShiftP f k del g*)(*i*::=*u*) = *ShiftP* (*subst i u f*) (*subst i u k*) (*subst i u del*) (*subst i u g*)
proof −
obtain *x*::*name* **and** x'::*name* **and** *y*::*name* **and** *z*::*name*
where *atom x* ♯ (x',*y*,*z*,*f*,*del*,*g*,*k*,*i*,*u*) *atom* x' ♯ (*y*,*z*,*f*,*del*,*g*,*k*,*i*,*u*)
atom y ♯ (*z*,*f*,*del*,*g*,*k*,*i*,*u*) *atom z* ♯ (*f*,*del*,*g*,*k*,*i*,*u*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *ShiftP.simps* [*of x* x' *y z*])
qed

lemma *ShiftP-Zero*: {} ⊢ *ShiftP Zero k d Zero*
proof −
obtain *x*::*name* **and** x'::*name* **and** *y*::*name* **and** *z*::*name*
where *atom x* ♯ (x',*y*,*z*,*k*,*d*) *atom* x' ♯ (*y*,*z*,*k*,*d*) *atom y* ♯ (*z*,*k*,*d*) *atom z* ♯ (*k*,*d*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *ShiftP.simps* [*of x* x' *y z*])
qed

lemma *ShiftP-Mem1*:
{*ShiftP f k del g*, *HPair a b IN f*, *HaddP del a a′*, *a IN k*} ⊢ *HPair a′ b IN g*

proof −
obtain *x*::*name* **and** *x′*::*name* **and** *y*::*name* **and** *z*::*name*
where *atom x* ♯ (*x′*,*y*,*z*,*f*,*del*,*g*,*k*,*a*,*a′*,*b*) *atom x′* ♯ (*y*,*z*,*f*,*del*,*g*,*k*,*a*,*a′*,*b*)
atom y ♯ (*z*,*f*,*del*,*g*,*k*,*a*,*a′*,*b*) *atom z* ♯ (*f*,*del*,*g*,*k*,*a*,*a′*,*b*)
by (*metis obtain-fresh*)
thus *?thesis*
apply (*auto simp*: *ShiftP.simps* [*of x x′ y z*])
apply (*rule All-E* [**where** *x*=*HPair a′ b*], *auto intro*!: *Iff-E2*)
apply (*rule Ex-I* [**where** *x*=*a*], *simp*)
apply (*rule Ex-I* [**where** *x*=*a′*], *simp*)
apply (*rule Ex-I* [**where** *x*=*b*], *auto intro*: *Mem-Eats-I1*)
done
qed

lemma *ShiftP-Mem2*:
assumes *atom u* ♯ (*f*,*k*,*del*,*g*,*a*,*b*)
shows {*ShiftP f k del g*, *HPair a b IN g*} ⊢ *Ex u* ((*Var u*) *IN k AND HaddP del* (*Var u*) *a AND HPair* (*Var u*) *b IN f*)
proof −
obtain *x*::*name* **and** *x′*::*name* **and** *y*::*name* **and** *z*::*name*
where *atoms*: *atom x* ♯ (*x′*,*y*,*z*,*f*,*del*,*g*,*k*,*a*,*u*,*b*) *atom x′* ♯ (*y*,*z*,*f*,*del*,*g*,*k*,*a*,*u*,*b*)
atom y ♯ (*z*,*f*,*del*,*g*,*k*,*a*,*u*,*b*) *atom z* ♯ (*f*,*del*,*g*,*k*,*a*,*u*,*b*)
by (*metis obtain-fresh*)
thus *?thesis* **using** *assms*
apply (*auto simp*: *ShiftP.simps* [*of x x′ y z*])
apply (*rule All-E* [**where** *x*=*HPair a b*])
apply (*auto intro*!: *Iff-E1* [*OF Assume*])
apply (*rule Ex-I* [**where** *x*=*Var x*])
apply (*auto intro*: *Mem-cong* [*OF HPair-cong Refl*, *THEN Iff-MP2-same*])
apply (*blast intro*: *HaddP-cong* [*OF Refl Refl*, *THEN Iff-MP2-same*])
done
qed

lemma *ShiftP-Mem-D*:
assumes *H* ⊢ *ShiftP f k del g* *H* ⊢ *a IN g*
atom x ♯ (*x′*,*y*,*a*,*f*,*del*,*g*,*k*) *atom x′* ♯ (*y*,*a*,*f*,*del*,*g*,*k*) *atom y* ♯ (*a*,*f*,*del*,*g*,*k*)
shows *H* ⊢ (*Ex x* (*Ex x′* (*Ex y* (*a EQ HPair* (*Var x′*) (*Var y*) *AND*
HaddP del (*Var x*) (*Var x′*) *AND*
HPair (*Var x*) (*Var y*) *IN f AND Var x IN k*))))
(**is** - ⊢ *?concl*)
proof −
obtain *z*::*name* **where** *atom z* ♯ (*x*,*x′*,*y*,*f*,*del*,*g*,*k*,*a*)
by (*metis obtain-fresh*)
hence {*ShiftP f k del g*, *a IN g*} ⊢ *?concl* **using** *assms*
by (*auto simp*: *ShiftP.simps* [*of x x′ y z*]) (*rule All-E* [**where** *x*=*a*], *auto intro*: *Iff-E1*)

thus *?thesis*
  by (*rule cut2*) (*rule assms*)+
qed

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

9.6 Union of Two Sets

nominal-function *UnionP* :: *tm* $\Rightarrow$ *tm* $\Rightarrow$ *tm* $\Rightarrow$ *fm*
  where *atom i* $\sharp$ (x,y,z) $\Longrightarrow$ *UnionP x y z* = *All i* (*Var i IN z IFF* (*Var i IN x OR Var i IN y*))
by (*auto simp*: *eqvt-def UnionP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma *UnionP-fresh-iff* [*simp*]: *a* $\sharp$ *UnionP x y z* $\longleftrightarrow$ *a* $\sharp$ *x* $\wedge$ *a* $\sharp$ *y* $\wedge$ *a* $\sharp$ *z*
proof –
  obtain *i*::*name* **where** *atom i* $\sharp$ (x,y,z)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by *auto*
qed

lemma *subst-fm-UnionP* [*simp*]:
  (*UnionP x y z*)(*i*::=*u*) = *UnionP* (*subst i u x*) (*subst i u y*) (*subst i u z*)
proof –
  obtain *j*::*name* **where** *atom j* $\sharp$ (x,y,z,i,u)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*auto simp*: *UnionP.simps* [*of j*])
qed

lemma *Union-Zero1*: *H* $\vdash$ *UnionP Zero x x*
proof –
  obtain *i*::*name* **where** *atom i* $\sharp$ *x*
    by (*metis obtain-fresh*)

hence {} ⊢ *UnionP Zero x x*
by (*auto simp*: *UnionP.simps* [*of i*] *intro*: *Disj-I2*)
thus *?thesis*
by (*metis thin0*)
qed

lemma *Union-Eats*: {*UnionP x y z*} ⊢ *UnionP* (*Eats x a*) *y* (*Eats z a*)
proof −
obtain *i*::*name* **where** *atom i* ♯ (*x*,*y*,*z*,*a*)
by (*metis obtain-fresh*)
thus *?thesis*
apply (*auto simp*: *UnionP.simps* [*of i*])
apply (*rule Ex-I* [**where** *x*=*Var i*])
apply (*auto intro*: *Iff-E1* [*THEN rotate2*] *Iff-E2* [*THEN rotate2*] *Mem-Eats-I1* *Mem-Eats-I2 Disj-I1 Disj-I2*)
done
qed

lemma *exists-Union-lemma*:
assumes *z*: *atom z* ♯ (*i*,*y*) **and** *i*: *atom i* ♯ *y*
shows {} ⊢ *Ex z* (*UnionP* (*Var i*) *y* (*Var z*))
proof −
obtain *j*::*name* **where** *j*: *atom j* ♯ (*y*,*z*,*i*)
by (*metis obtain-fresh*)
show {} ⊢ *Ex z* (*UnionP* (*Var i*) *y* (*Var z*))
apply (*rule Ind* [*of j i*]) **using** *j z i*
apply *simp-all*
apply (*rule Ex-I* [**where** *x*=*y*], *simp add*: *Union-Zero1*)
apply (*auto del*: *Ex-EH*)
apply (*rule Ex-E*)
apply (*rule NegNeg-E*)
apply (*rule Ex-E*)
apply (*auto del*: *Ex-EH*)
apply (*rule thin1*, *force intro*: *Ex-I* [**where** *x*=*Eats* (*Var z*) (*Var j*)] *Union-Eats*)
done
qed

lemma *exists-UnionP*:
assumes *z*: *atom z* ♯ (*x*,*y*) **shows** *H* ⊢ *Ex z* (*UnionP x y* (*Var z*))
proof −
obtain *i*::*name* **where** *i*: *atom i* ♯ (*y*,*z*)
by (*metis obtain-fresh*)
hence {} ⊢ *Ex z* (*UnionP* (*Var i*) *y* (*Var z*))
by (*metis exists-Union-lemma fresh-Pair fresh-at-base*(*2*) *z*)
hence {} ⊢ (*Ex z* (*UnionP* (*Var i*) *y* (*Var z*)))(*i*::=*x*)
by (*metis Subst empty-iff*)
thus *?thesis* **using** *i z*
by (*simp add*: *thin0*)
qed

lemma *UnionP-Mem1*: { *UnionP x y z*, *a IN x* } ⊢ *a IN z*
proof −
  obtain *i*::*name* **where** *atom i* ♯ (*x*,*y*,*z*,*a*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*force simp*: *UnionP.simps* [*of i*] *intro*: *All-E* [**where** *x*=*a*] *Disj-I1 Iff-E2*)
qed

lemma *UnionP-Mem2*: { *UnionP x y z*, *a IN y* } ⊢ *a IN z*
proof −
  obtain *i*::*name* **where** *atom i* ♯ (*x*,*y*,*z*,*a*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*force simp*: *UnionP.simps* [*of i*] *intro*: *All-E* [**where** *x*=*a*] *Disj-I2 Iff-E2*)
qed

lemma *UnionP-Mem*: { *UnionP x y z*, *a IN z* } ⊢ *a IN x OR a IN y*
proof −
  obtain *i*::*name* **where** *atom i* ♯ (*x*,*y*,*z*,*a*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    by (*force simp*: *UnionP.simps* [*of i*] *intro*: *All-E* [**where** *x*=*a*] *Iff-E1*)
qed

lemma *UnionP-Mem-E*:
  assumes *H* ⊢ *UnionP x y z*
      and *insert* (*a IN x*) *H* ⊢ *A*
      and *insert* (*a IN y*) *H* ⊢ *A*
    shows *insert* (*a IN z*) *H* ⊢ *A*
  using *assms*
  by (*blast intro*: *rotate2 cut-same* [*OF UnionP-Mem* [*THEN cut2*]] *thin1*)

9.7 Append on Sequences

nominal-function *SeqAppendP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
  where ⟦*atom g1* ♯ (*g2*,*f1*,*k1*,*f2*,*k2*,*g*); *atom g2* ♯ (*f1*,*k1*,*f2*,*k2*,*g*)⟧ ⟹
    SeqAppendP f1 k1 f2 k2 g =
      (*Ex g1* (*Ex g2* (*RestrictedP f1 k1* (*Var g1*) *AND*
                    ShiftP f2 k2 k1 (*Var g2*) *AND*
                    UnionP (*Var g1*) (*Var g2*) *g*)))
by (*auto simp*: *eqvt-def SeqAppendP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
  by *lexicographic-order*

lemma *SeqAppendP-fresh-iff* [*simp*]:
  a ♯ *SeqAppendP f1 k1 f2 k2 g* ⟷ *a* ♯ *f1* ∧ *a* ♯ *k1* ∧ *a* ♯ *f2* ∧ *a* ♯ *k2* ∧ *a* ♯ *g*
proof −

obtain *g1*::*name* **and** *g2*::*name*
where *atom g1* ♯ (*g2*,*f1*,*k1*,*f2*,*k2*,*g*) *atom g2* ♯ (*f1*,*k1*,*f2*,*k2*,*g*)
by (*metis obtain-fresh*)
thus *?thesis*
by *auto*
qed

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 −
obtain *g1*::*name* **and** *g2*::*name*
where *atom g1* ♯ (*g2*,*f1*,*k1*,*f2*,*k2*,*g*,*i*,*u*) *atom g2* ♯ (*f1*,*k1*,*f2*,*k2*,*g*,*i*,*u*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SeqAppendP.simps* [*of g1 g2*])
qed

lemma *exists-SeqAppendP*:
assumes *atom g* ♯ (*f1*,*k1*,*f2*,*k2*)
shows *H* ⊢ *Ex g* (*SeqAppendP f1 k1 f2 k2* (*Var g*))
proof −
obtain *g1*::*name* **and** *g2*::*name*
where *atoms*: *atom g1* ♯ (*g2*,*f1*,*k1*,*f2*,*k2*,*g*) *atom g2* ♯ (*f1*,*k1*,*f2*,*k2*,*g*)
by (*metis obtain-fresh*)
hence {} ⊢ *Ex g* (*SeqAppendP f1 k1 f2 k2* (*Var g*))
using *assms*
apply (*auto simp*: *SeqAppendP.simps* [*of g1 g2*])
apply (*rule cut-same* [*OF exists-RestrictedP* [*of g1 f1 k1*]], *auto*)
apply (*rule cut-same* [*OF exists-ShiftP* [*of g2 f2 k2 k1*]], *auto*)
apply (*rule cut-same* [*OF exists-UnionP* [*of g Var g1 Var g2*]], *auto*)
apply (*rule Ex-I* [**where** *x*=*Var g*], *simp*)
apply (*rule Ex-I* [**where** *x*=*Var g1*], *simp*)
apply (*rule Ex-I* [**where** *x*=*Var g2*], *auto*)
done
thus *?thesis* **using** *assms*
by (*metis thin0*)
qed

lemma *SeqAppendP-Mem1*: {*SeqAppendP f1 k1 f2 k2 g*, *HPair x y IN f1*, *x IN k1*} ⊢ *HPair x y IN g*
proof −
obtain *g1*::*name* **and** *g2*::*name*
where *atom g1* ♯ (*g2*,*f1*,*k1*,*f2*,*k2*,*g*,*x*,*y*) *atom g2* ♯ (*f1*,*k1*,*f2*,*k2*,*g*,*x*,*y*)
by (*metis obtain-fresh*)
thus *?thesis*
using *assms*
by (*auto simp*: *SeqAppendP.simps* [*of g1 g2*] *intro*: *UnionP-Mem1* [*THEN cut2*]

RestrictedP-Mem [*THEN cut3*])
qed

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 −
  obtain *g1*::*name* **and** *g2*::*name*
    where *atom g1* ♯ (*g2*,*f1*,*k1*,*f2*,*k2*,*g*,*x*,*x'*,*y*) *atom g2* ♯ (*f1*,*k1*,*f2*,*k2*,*g*,*x*,*x'*,*y*)
    by (*metis obtain-fresh*)
  thus *?thesis*
    using *assms*
   by (*auto simp*: *SeqAppendP.simps* [*of g1 g2*] *intro*: *UnionP-Mem2* [*THEN cut2*] *ShiftP-Mem1* [*THEN cut4*])
qed

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*

9.8 LstSeqP and SeqAppendP

lemma *HDomain-Incl-SeqAppendP*: — The And eliminates the need to prove *cut5*

  {*SeqAppendP f1 k1 f2 k2 g*, *HDomain-Incl f1 k1 AND HDomain-Incl f2 k2*,
    HaddP k1 k2 k, *OrdP k1*} ⊢ *HDomain-Incl g k*
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*
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 −
  have {*SeqAppendP f1* (*SUCC k1*) *f2* (*SUCC k2*) *g*, *LstSeqP f1 k1 y1*, *LstSeqP f2 k2 y2*, *HaddP k1 k2 k*}
   ⊢ *LstSeqP g* (*SUCC k*) *y2*
    apply (*auto simp*: *LstSeqP.simps intro*: *HaddP-OrdP OrdP-SUCC-I*)
    apply (*rule HDomain-Incl-SeqAppendP* [*THEN cut4*])
    apply (*rule AssumeH Conj-I*)+
   apply (*blast intro*: *HaddP-SUCC1* [*THEN cut1*] *HaddP-SUCC2* [*THEN cut1*])
    apply (*blast intro*: *HaddP-OrdP OrdP-SUCC-I*)
    apply (*rule HFun-Sigma-SeqAppendP* [*THEN cut4*])
    apply (*auto intro*: *HaddP-OrdP OrdP-SUCC-I*)

apply (*blast intro*: *Mem-SUCC-Refl HaddP-SUCC1* [*THEN cut1*] *HaddP-SUCC2* [*THEN cut1*]
SeqAppendP-Mem2 [*THEN cut4*])
done
thus *?thesis* **using** *assms*
by (*rule cut4*)
qed

lemma *SeqAppendP-NotInDom*: {*SeqAppendP f1 k1 f2 k2 g*, *HaddP k1 k2 k*, *OrdP k1*} ⊢ *NotInDom k g*
proof −
obtain *x*::*name* **and** *z*::*name*
where *atom x* ♯ (*z*,*f1*,*k1*,*f2*,*k2*,*g*,*k*) *atom z* ♯ (*f1*,*k1*,*f2*,*k2*,*g*,*k*)
by (*metis obtain-fresh*)
thus *?thesis*
apply (*auto simp*: *NotInDom.simps* [*of z*])
apply (*rule SeqAppendP-Mem-E* [**where** *u*=*x*])
apply (*rule AssumeH*)+
apply (*blast intro*: *HaddP-Mem-contra*, *simp-all*)
apply (*rule cut-same* [**where** *A*=(*Var x*) *EQ k2*])
apply (*blast intro*: *HaddP-inv2* [*THEN cut3*])
apply (*blast intro*: *Mem-non-refl* [**where** *x*=*k2*] *Mem-cong* [*OF - Refl*, *THEN Iff-MP-same*])
done
qed

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 −
have {*SeqAppendP f1* (*SUCC k1*) *f2* (*SUCC k2*) *g*, *LstSeqP f1 k1 y1*, *LstSeqP f2 k2 y2*, *HaddP k1 k2 k*}
⊢ *LstSeqP* (*Eats g* (*HPair* (*SUCC* (*SUCC k*)) *z*)) (*SUCC* (*SUCC k*)) *z*
apply (*rule cut2* [*OF NotInDom-LstSeqP-Eats*])
apply (*rule SeqAppendP-NotInDom* [*THEN cut3*])
apply (*rule AssumeH*)
apply (*metis HaddP-SUCC1 HaddP-SUCC2 cut1 thin1*)
apply (*metis Assume LstSeqP-OrdP OrdP-SUCC-I insert-commute*)
apply (*blast intro*: *LstSeqP-SeqAppendP*)
done
thus *?thesis* **using** *assms*
by (*rule cut4*)
qed

9.9 Substitution and Abstraction on Terms

9.9.1 Atomic cases

lemma *SeqStTermP-Var-same*:
assumes *atom s* ♯ (*k*,*v*,*i*) *atom k* ♯ (*v*,*i*)
shows {*VarP v*} ⊢ *Ex s* (*Ex k* (*SeqStTermP v i v i* (*Var s*) (*Var k*)))
proof –
obtain *l*::*name* **and** *sl*::*name* **and** *sl'*::*name* **and** *m*::*name* **and** *sm*::*name* **and** *sm'*::*name*
and *n*::*name* **and** *sn*::*name* **and** *sn'*::*name*
where *atom l* ♯ (*v*,*i*,*s*,*k*,*sl*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl ♯ (*v*,*i*,*s*,*k*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl' ♯ (*v*,*i*,*s*,*k*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom m ♯ (*v*,*i*,*s*,*k*,*n*,*sm*,*sm'*,*sn*,*sn'*) *atom n* ♯ (*v*,*i*,*s*,*k*,*sm*,*sm'*,*sn*,*sn'*)
atom sm ♯ (*v*,*i*,*s*,*k*,*sm'*,*sn*,*sn'*) *atom sm'* ♯ (*v*,*i*,*s*,*k*,*sn*,*sn'*)
atom sn ♯ (*v*,*i*,*s*,*k*,*sn'*) *atom sn'* ♯ (*v*,*i*,*s*,*k*)
by (*metis obtain-fresh*)
thus *?thesis* **using** *assms*
apply (*simp add*: *SeqStTermP.simps* [*of l - - v i sl sl' m n sm sm' sn sn'*])
apply (*rule Ex-I* [**where** *x* = *Eats Zero* (*HPair Zero* (*HPair v i*))], *simp*)
apply (*rule Ex-I* [**where** *x* = *Zero*], *auto intro*!: *Mem-SUCC-EH*)
apply (*rule Ex-I* [**where** *x* = *v*], *simp*)
apply (*rule Ex-I* [**where** *x* = *i*], *auto intro*: *Disj-I1 Mem-Eats-I2 HPair-cong*)
done
qed

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 –
obtain *l*::*name* **and** *sl*::*name* **and** *sl'*::*name* **and** *m*::*name* **and** *sm*::*name* **and** *sm'*::*name*
and *n*::*name* **and** *sn*::*name* **and** *sn'*::*name*
where *atom l* ♯ (*v*,*w*,*i*,*s*,*k*,*sl*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl ♯ (*v*,*w*,*i*,*s*,*k*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl' ♯ (*v*,*w*,*i*,*s*,*k*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom m ♯ (*v*,*w*,*i*,*s*,*k*,*n*,*sm*,*sm'*,*sn*,*sn'*) *atom n* ♯ (*v*,*w*,*i*,*s*,*k*,*sm*,*sm'*,*sn*,*sn'*)
atom sm ♯ (*v*,*w*,*i*,*s*,*k*,*sm'*,*sn*,*sn'*) *atom sm'* ♯ (*v*,*w*,*i*,*s*,*k*,*sn*,*sn'*)
atom sn ♯ (*v*,*w*,*i*,*s*,*k*,*sn'*) *atom sn'* ♯ (*v*,*w*,*i*,*s*,*k*)
by (*metis obtain-fresh*)
thus *?thesis* **using** *assms*
apply (*simp add*: *SeqStTermP.simps* [*of l - - v i sl sl' m n sm sm' sn sn'*])
apply (*rule Ex-I* [**where** *x* = *Eats Zero* (*HPair Zero* (*HPair w w*))], *simp*)
apply (*rule Ex-I* [**where** *x* = *Zero*], *auto intro*!: *Mem-SUCC-EH*)
apply (*rule rotate2* [*OF Swap*])
apply (*rule Ex-I* [**where** *x* = *w*], *simp*)
apply (*rule Ex-I* [**where** *x* = *w*], *auto simp*: *VarP-def*)
apply (*blast intro*: *HPair-cong Mem-Eats-I2*)

apply (*blast intro*: *Sym OrdNotEqP-I Disj-I1 Disj-I2*)
done
qed

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*)))
corollary *SubstTermP-Zero*: {*TermP t*} ⊢ *SubstTermP* ⌈*Var v*⌉ *t Zero Zero*
proof −
obtain *s*::*name* **and** *k*::*name* **where** *atom s* ♯ (*v*,*t*,*k*) *atom k* ♯ (*v*,*t*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SubstTermP.simps* [*of s - - - - k*] *intro*: *SeqStTermP-Zero* [*THEN cut1*])
qed

corollary *SubstTermP-Var-same*: {*VarP v*, *TermP t*} ⊢ *SubstTermP v t v t*
proof −
obtain *s*::*name* **and** *k*::*name* **where** *atom s* ♯ (*v*,*t*,*k*) *atom k* ♯ (*v*,*t*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SubstTermP.simps* [*of s - - - - k*] *intro*: *SeqStTermP-Var-same* [*THEN cut1*])
qed

corollary *SubstTermP-Var-diff*: {*VarP v*, *VarP w*, *Neg* (*v EQ w*), *TermP t*} ⊢ *SubstTermP v t w w*
proof −
obtain *s*::*name* **and** *k*::*name* **where** *atom s* ♯ (*v*,*w*,*t*,*k*) *atom k* ♯ (*v*,*w*,*t*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SubstTermP.simps* [*of s - - - - k*] *intro*: *SeqStTermP-Var-diff* [*THEN cut3*])
qed

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 −
obtain *l*::*name* **and** *sl*::*name* **and** *sl'*::*name* **and** *m*::*name* **and** *sm*::*name* **and** *sm'*::*name*
and *n*::*name* **and** *sn*::*name* **and** *sn'*::*name*
where *atom l* ♯ (*v*,*t*,*i*,*s*,*k*,*sl*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl ♯ (*v*,*t*,*i*,*s*,*k*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl' ♯ (*v*,*t*,*i*,*s*,*k*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom m ♯ (*v*,*t*,*i*,*s*,*k*,*n*,*sm*,*sm'*,*sn*,*sn'*) *atom n* ♯ (*v*,*t*,*i*,*s*,*k*,*sm*,*sm'*,*sn*,*sn'*)
atom sm ♯ (*v*,*t*,*i*,*s*,*k*,*sm'*,*sn*,*sn'*) *atom sm'* ♯ (*v*,*t*,*i*,*s*,*k*,*sn*,*sn'*)
atom sn ♯ (*v*,*t*,*i*,*s*,*k*,*sn'*) *atom sn'* ♯ (*v*,*t*,*i*,*s*,*k*)
by (*metis obtain-fresh*)

thus *?thesis* **using** *assms*
apply (*simp add*: *SeqStTermP.simps* [*of l - - v i sl sl′ m n sm sm′ sn sn′*])
apply (*rule Ex-I* [**where** *x* = *Eats Zero* (*HPair Zero* (*HPair t t*))], *simp*)
apply (*rule Ex-I* [**where** *x* = *Zero*], *auto intro*!: *Mem-SUCC-EH*)
apply (*rule Ex-I* [**where** *x* = *t*], *simp*)
apply (*rule Ex-I* [**where** *x* = *t*], *auto intro*: *HPair-cong Mem-Eats-I2*)
apply (*blast intro*: *Disj-I1 Disj-I2 VarP-neq-IndP*)
done
qed

corollary *SubstTermP-Ind*: {*VarP v*, *IndP w*, *TermP t*} ⊢ *SubstTermP v t w w*
proof −
obtain *s*::*name* **and** *k*::*name* **where** *atom s* ♯ (*v*,*w*,*t*,*k*) *atom k* ♯ (*v*,*w*,*t*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*force simp*: *SubstTermP.simps* [*of s - - - - k*]
intro: *SeqStTermP-Ind* [*THEN cut2*])
qed

9.9.2 Non-atomic cases

lemma *SeqStTermP-Eats*:
assumes *sk*: *atom s* ♯ (*k*,*s1*,*s2*,*k1*,*k2*,*t1*,*t2*,*u1*,*u2*,*v*,*i*)
atom k ♯ (*t1*,*t2*,*u1*,*u2*,*v*,*i*)
shows {*SeqStTermP v i t1 u1 s1 k1*, *SeqStTermP v i t2 u2 s2 k2*}
⊢ *Ex s* (*Ex k* (*SeqStTermP v i* (*Q-Eats t1 t2*) (*Q-Eats u1 u2*) (*Var s*) (*Var k*)))
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 −
obtain *k1*::*name* **and** *s1*::*name* **and** *k2*::*name* **and** *s2*::*name* **and** *k*::*name* **and** *s*::*name*
where *atom s1* ♯ (*v*,*i*,*t1*,*u1*,*t2*,*u2*) *atom k1* ♯ (*v*,*i*,*t1*,*u1*,*t2*,*u2*,*s1*)
atom s2 ♯ (*v*,*i*,*t1*,*u1*,*t2*,*u2*,*k1*,*s1*) *atom k2* ♯ (*v*,*i*,*t1*,*u1*,*t2*,*u2*,*s2*,*k1*,*s1*)
atom s ♯ (*v*,*i*,*t1*,*u1*,*t2*,*u2*,*k2*,*s2*,*k1*,*s1*)
atom k ♯ (*v*,*i*,*t1*,*u1*,*t2*,*u2*,*s*,*k2*,*s2*,*k1*,*s1*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto intro*!: *SeqStTermP-Eats* [*THEN cut2*]
simp: *SubstTermP.simps* [*of s - - -* (*Q-Eats u1 u2*) *k*]
SubstTermP.simps [*of s1 v i t1 u1 k1*]
SubstTermP.simps [*of s2 v i t2 u2 k2*])
qed

9.9.3 Substitution over a constant

lemma *SeqConstP-lemma*:
assumes *atom m* ♯ (*s*,*k*,*c*,*n*,*sm*,*sn*) *atom n* ♯ (*s*,*k*,*c*,*sm*,*sn*)
atom sm ♯ (*s*,*k*,*c*,*sn*) *atom sn* ♯ (*s*,*k*,*c*)

shows { *SeqConstP s k c* }
⊢ *c EQ Zero OR*
Ex m (*Ex n* (*Ex sm* (*Ex sn* (*Var m IN k AND Var n IN k AND*
SeqConstP s (*Var m*) (*Var sm*) *AND*
SeqConstP s (*Var n*) (*Var sn*) *AND*
c EQ Q-Eats (*Var sm*) (*Var sn*)))))
lemma *SeqConstP-imp-SubstTermP*: {*SeqConstP s kk c*, *TermP t*} ⊢ *SubstTermP* ⌈*Var w*⌉ *t c c*
theorem *SubstTermP-Const*: {*ConstP c*, *TermP t*} ⊢ *SubstTermP* ⌈*Var w*⌉ *t c c*
proof −
obtain *s*::*name* **and** *k*::*name* **where** *atom s* ♯ (*c*,*t*,*w*,*k*) *atom k* ♯ (*c*,*t*,*w*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *CTermP.simps* [*of k s c*] *SeqConstP-imp-SubstTermP*)
qed

9.10 Substitution on Formulas

9.10.1 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 −
obtain *t*::*name* **and** *u*::*name* **and** *t′*::*name* **and** *u′*::*name*
where *atom t* ♯ (*v*,*i*,*x*,*x′*,*y*,*y′*,*t′*,*u*,*u′*) *atom t′* ♯ (*v*,*i*,*x*,*x′*,*y*,*y′*,*u*,*u′*)
atom u ♯ (*v*,*i*,*x*,*x′*,*y*,*y′*,*u′*) *atom u′* ♯ (*v*,*i*,*x*,*x′*,*y*,*y′*)
by (*metis obtain-fresh*)
thus *?thesis*
apply (*simp add*: *SubstAtomicP.simps* [*of t - - - - t′ u u′*])
apply (*rule Ex-I* [**where** *x* = *x*], *simp*)
apply (*rule Ex-I* [**where** *x* = *y*], *simp*)
apply (*rule Ex-I* [**where** *x* = *x′*], *simp*)
apply (*rule Ex-I* [**where** *x* = *y′*], *auto intro*: *Disj-I2*)
done
qed

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 −
let *?vs* = (*s*,*k*,*x*,*y*,*x′*,*y′*,*v*,*i*)
obtain *l*::*name* **and** *sl*::*name* **and** *sl′*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sm′*::*name* **and** *sn*::*name* **and** *sn′*::*name*
where *atom l* ♯ (*?vs*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
atom sl ♯ (*?vs*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*) *atom sl′* ♯ (*?vs*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
atom m ♯ (*?vs*,*n*,*sm*,*sm′*,*sn*,*sn′*) *atom n* ♯ (*?vs*,*sm*,*sm′*,*sn*,*sn′*)

atom sm ♯ (?vs,sm′,sn,sn′) atom sm′ ♯ (?vs,sn,sn′)
atom sn ♯ (?vs,sn′) atom sn′ ♯ ?vs
by (*metis obtain-fresh*)
thus *?thesis*
using *assms*
apply (*auto simp*: *SeqSubstFormP.simps* [*of l Var s - - - sl sl′ m n sm sm′ sn sn′*])
apply (*rule Ex-I* [**where** *x = Eats Zero* (*HPair Zero* (*HPair* (*Q-Mem x y*) (*Q-Mem x′ y′*)))], *simp*)
apply (*rule Ex-I* [**where** *x = Zero*], *auto intro*!: *Mem-SUCC-EH*)
apply (*rule Ex-I* [**where** *x = Q-Mem x y*], *simp*)
apply (*rule Ex-I* [**where** *x = Q-Mem x′ y′*], *auto intro*: *Mem-Eats-I2 HPair-cong*)
apply (*blast intro*: *SubstAtomicP-Mem* [*THEN cut2*] *Disj-I1*)
done
qed

lemma *SubstFormP-Mem*:
{*SubstTermP v i x x′*, *SubstTermP v i y y′*} ⊢ *SubstFormP v i* (*Q-Mem x y*) (*Q-Mem x′ y′*)
proof −
obtain *k1*::*name* **and** *s1*::*name* **and** *k2*::*name* **and** *s2*::*name* **and** *k*::*name* **and** *s*::*name*
where *atom s1 ♯ (v,i,x,y,x′,y′) atom k1 ♯ (v,i,x,y,x′,y′,s1)*
atom s2 ♯ (v,i,x,y,x′,y′,k1,s1) atom k2 ♯ (v,i,x,y,x′,y′,s2,k1,s1)
atom s ♯ (v,i,x,y,x′,y′,k2,s2,k1,s1) atom k ♯ (v,i,x,y,x′,y′,s,k2,s2,k1,s1)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SubstFormP.simps* [*of s v i* (*Q-Mem x y*) *- k*]
SubstFormP.simps [*of s1 v i x x′ k1*]
SubstFormP.simps [*of s2 v i y y′ k2*]
intro: *SubstTermP-imp-TermP SubstTermP-imp-VarP SeqSubstFormP-Mem thin1*)
qed

9.10.2 Equality

lemma *SubstAtomicP-Eq*:
{*SubstTermP v i x x′*, *SubstTermP v i y y′*} ⊢ *SubstAtomicP v i* (*Q-Eq x y*) (*Q-Eq x′ y′*)
proof −
obtain *t*::*name* **and** *u*::*name* **and** *t′*::*name* **and** *u′*::*name*
where *atom t ♯ (v,i,x,x′,y,y′,t′,u,u′) atom t′ ♯ (v,i,x,x′,y,y′,u,u′)*
atom u ♯ (v,i,x,x′,y,y′,u′) atom u′ ♯ (v,i,x,x′,y,y′)
by (*metis obtain-fresh*)
thus *?thesis*
apply (*simp add*: *SubstAtomicP.simps* [*of t - - - - t′ u u′*])
apply (*rule Ex-I* [**where** *x = x*], *simp*)
apply (*rule Ex-I* [**where** *x = y*], *simp*)
apply (*rule Ex-I* [**where** *x = x′*], *simp*)

apply (*rule Ex-I* [**where** $x = y'$], *auto intro*: *Disj-I1*)
done
qed

lemma *SeqSubstFormP-Eq*:
assumes *sk*: *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-Eq x y*) (*Q-Eq x' y'*) (*Var s*) (*Var k*)))
proof −
let *?vs* = (s,k,x,y,x',y',v,i)
obtain *l*::*name* **and** *sl*::*name* **and** *sl'*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sm'*::*name* **and** *sn*::*name* **and** *sn'*::*name*
where *atom l* ♯ (*?vs*,*sl*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom sl ♯ (*?vs*,*sl'*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*) *atom sl'* ♯ (*?vs*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom m ♯ (*?vs*,*n*,*sm*,*sm'*,*sn*,*sn'*) *atom n* ♯ (*?vs*,*sm*,*sm'*,*sn*,*sn'*)
atom sm ♯ (*?vs*,*sm'*,*sn*,*sn'*) *atom sm'* ♯ (*?vs*,*sn*,*sn'*)
atom sn ♯ (*?vs*,*sn'*) *atom sn'* ♯ *?vs*
by (*metis obtain-fresh*)
thus *?thesis*
using *sk*
apply (*auto simp*: *SeqSubstFormP.simps* [*of l Var s - - - sl sl' m n sm sm' sn sn'*])
apply (*rule Ex-I* [**where** *x* = *Eats Zero* (*HPair Zero* (*HPair* (*Q-Eq x y*) (*Q-Eq x' y'*)))], *simp*)
apply (*rule Ex-I* [**where** *x* = *Zero*], *auto intro*!: *Mem-SUCC-EH*)
apply (*rule Ex-I* [**where** *x* = *Q-Eq x y*], *simp*)
apply (*rule Ex-I* [**where** *x* = *Q-Eq x' y'*], *auto*)
apply (*metis Mem-Eats-I2 Assume HPair-cong Refl*)
apply (*blast intro*: *SubstAtomicP-Eq* [*THEN cut2*] *Disj-I1*)
done
qed

lemma *SubstFormP-Eq*:
{*SubstTermP v i x x'*, *SubstTermP v i y y'*} ⊢ *SubstFormP v i* (*Q-Eq x y*) (*Q-Eq x' y'*)
proof −
obtain *k1*::*name* **and** *s1*::*name* **and** *k2*::*name* **and** *s2*::*name* **and** *k*::*name* **and** *s*::*name*
where *atom s1* ♯ (v,i,x,y,x',y') *atom k1* ♯ (v,i,x,y,x',y',*s1*)
atom s2 ♯ (v,i,x,y,x',y',*k1*,*s1*) *atom k2* ♯ (v,i,x,y,x',y',*s2*,*k1*,*s1*)
atom s ♯ (v,i,x,y,x',y',*k2*,*s2*,*k1*,*s1*) *atom k* ♯ (v,i,x,y,x',y',*s*,*k2*,*s2*,*k1*,*s1*)

by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *SubstFormP.simps* [*of s v i* (*Q-Eq x y*) - *k*]
SubstFormP.simps [*of s1 v i x x' k1*]
SubstFormP.simps [*of s2 v i y y' k2*]
intro: *SeqSubstFormP-Eq SubstTermP-imp-TermP SubstTermP-imp-VarP*

thin1)
qed

9.10.3 Negation

lemma *SeqSubstFormP-Neg*:
assumes *atom s* ♯ (*k*,*s1*,*k1*,*x*,*x*′,*v*,*i*) *atom k* ♯ (*s1*,*k1*,*x*,*x*′,*v*,*i*)
shows {*SeqSubstFormP v i x x*′ *s1 k1*, *TermP i*, *VarP v*}
⊢ *Ex s* (*Ex k* (*SeqSubstFormP v i* (*Q-Neg x*) (*Q-Neg x*′) (*Var s*) (*Var k*)))

theorem *SubstFormP-Neg*: {*SubstFormP v i x x*′} ⊢ *SubstFormP v i* (*Q-Neg x*) (*Q-Neg x*′)
proof −
obtain *k1*::*name* **and** *s1*::*name* **and** *k*::*name* **and** *s*::*name*
where *atom s1* ♯ (*v*,*i*,*x*,*x*′) *atom k1* ♯ (*v*,*i*,*x*,*x*′,*s1*)
atom s ♯ (*v*,*i*,*x*,*x*′,*k1*,*s1*) *atom k* ♯ (*v*,*i*,*x*,*x*′,*s*,*k1*,*s1*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*force simp*: *SubstFormP*.*simps* [*of s v i Q-Neg x - k*] *SubstFormP*.*simps* [*of s1 v i x x*′ *k1*]
intro: *SeqSubstFormP-Neg* [*THEN cut3*])
qed

9.10.4 Disjunction

lemma *SeqSubstFormP-Disj*:
assumes *atom s* ♯ (*k*,*s1*,*s2*,*k1*,*k2*,*x*,*y*,*x*′,*y*′,*v*,*i*) *atom k* ♯ (*s1*,*s2*,*k1*,*k2*,*x*,*y*,*x*′,*y*′,*v*,*i*)
shows {*SeqSubstFormP v i x x*′ *s1 k1*,
SeqSubstFormP v i y y′ *s2 k2*, *TermP i*, *VarP v*}
⊢ *Ex s* (*Ex k* (*SeqSubstFormP v i* (*Q-Disj x y*) (*Q-Disj x*′ *y*′) (*Var s*) (*Var k*)))
theorem *SubstFormP-Disj*:
{*SubstFormP v i x x*′, *SubstFormP v i y y*′} ⊢ *SubstFormP v i* (*Q-Disj x y*) (*Q-Disj x*′ *y*′)
proof −
obtain *k1*::*name* **and** *s1*::*name* **and** *k2*::*name* **and** *s2*::*name* **and** *k*::*name* **and** *s*::*name*
where *atom s1* ♯ (*v*,*i*,*x*,*y*,*x*′,*y*′) *atom k1* ♯ (*v*,*i*,*x*,*y*,*x*′,*y*′,*s1*)
atom s2 ♯ (*v*,*i*,*x*,*y*,*x*′,*y*′,*k1*,*s1*) *atom k2* ♯ (*v*,*i*,*x*,*y*,*x*′,*y*′,*s2*,*k1*,*s1*)
atom s ♯ (*v*,*i*,*x*,*y*,*x*′,*y*′,*k2*,*s2*,*k1*,*s1*) *atom k* ♯ (*v*,*i*,*x*,*y*,*x*′,*y*′,*s*,*k2*,*s2*,*k1*,*s1*)

by (*metis obtain-fresh*)
thus *?thesis*
by (*force simp*: *SubstFormP*.*simps* [*of s v i Q-Disj x y - k*]
SubstFormP.*simps* [*of s1 v i x x*′ *k1*]
SubstFormP.*simps* [*of s2 v i y y*′ *k2*]
intro: *SeqSubstFormP-Disj* [*THEN cut4*])
qed

9.10.5 Existential

lemma *SeqSubstFormP-Ex*:
 assumes *atom s* ♯ (*k*,*s1*,*k1*,*x*,*x*′,*v*,*i*) *atom k* ♯ (*s1*,*k1*,*x*,*x*′,*v*,*i*)
  shows {*SeqSubstFormP v i x x*′ *s1 k1*, *TermP i*, *VarP v*}
   ⊢ *Ex s* (*Ex k* (*SeqSubstFormP v i* (*Q-Ex x*) (*Q-Ex x*′) (*Var s*) (*Var k*)))
theorem *SubstFormP-Ex*: {*SubstFormP v i x x*′} ⊢ *SubstFormP v i* (*Q-Ex x*) (*Q-Ex x*′)
proof −
 obtain *k1*::*name* **and** *s1*::*name* **and** *k*::*name* **and** *s*::*name*
  where *atom s1* ♯ (*v*,*i*,*x*,*x*′)   *atom k1* ♯ (*v*,*i*,*x*,*x*′,*s1*)
   atom s ♯ (*v*,*i*,*x*,*x*′,*k1*,*s1*) *atom k* ♯ (*v*,*i*,*x*,*x*′,*s*,*k1*,*s1*)
  by (*metis obtain-fresh*)
 thus *?thesis*
  by (*force simp*: *SubstFormP.simps* [*of s v i Q-Ex x - k*] *SubstFormP.simps* [*of s1 v i x x*′ *k1*]
   intro: *SeqSubstFormP-Ex* [*THEN cut3*])
qed

9.11 Constant Terms

lemma *ConstP-Zero*: {} ⊢ *ConstP Zero*
 by (*auto intro*: *Sigma-fm-imp-thm* [*OF CTermP-sf*] *simp*: *Const-0 ground-fm-aux-def supp-conv-fresh*)

lemma *SeqConstP-Eats*:
 assumes *atom s* ♯ (*k*,*s1*,*s2*,*k1*,*k2*,*t1*,*t2*) *atom k* ♯ (*s1*,*s2*,*k1*,*k2*,*t1*,*t2*)
  shows {*SeqConstP s1 k1 t1*, *SeqConstP s2 k2 t2*}
   ⊢ *Ex s* (*Ex k* (*SeqConstP* (*Var s*) (*Var k*) (*Q-Eats t1 t2*)))
theorem *ConstP-Eats*: {*ConstP t1*, *ConstP t2*} ⊢ *ConstP* (*Q-Eats t1 t2*)
proof −
 obtain *k1*::*name* **and** *s1*::*name* **and** *k2*::*name* **and** *s2*::*name* **and** *k*::*name* **and** *s*::*name*
  where *atom s1* ♯ (*t1*,*t2*) *atom k1* ♯ (*t1*,*t2*,*s1*)
   atom s2 ♯ (*t1*,*t2*,*k1*,*s1*) *atom k2* ♯ (*t1*,*t2*,*s2*,*k1*,*s1*)
   atom s ♯ (*t1*,*t2*,*k2*,*s2*,*k1*,*s1*) *atom k* ♯ (*t1*,*t2*,*s*,*k2*,*s2*,*k1*,*s1*)
  by (*metis obtain-fresh*)
 thus *?thesis*
  by (*auto simp*: *CTermP.simps* [*of k s* (*Q-Eats t1 t2*)]
    CTermP.simps [*of k1 s1 t1*] *CTermP.simps* [*of k2 s2 t2*]
   intro!: *SeqConstP-Eats* [*THEN cut2*])
qed

9.12 Proofs

lemma *PrfP-inference*:
 assumes *atom s* ♯ (*k*,*s1*,*s2*,*k1*,*k2*,*α1*,*α2*,*β*) *atom k* ♯ (*s1*,*s2*,*k1*,*k2*,*α1*,*α2*,*β*)
  shows {*PrfP s1 k1 α1*, *PrfP s2 k2 α2*, *ModPonP α1 α2 β OR ExistsP α1 β OR SubstP α1 β*}

⊢ *Ex k* (*Ex s* (*PrfP* (*Var s*) (*Var k*) β))
corollary *PfP-inference*: {*PfP* α*1*, *PfP* α*2*, *ModPonP* α*1* α*2* β *OR ExistsP* α*1* β *OR SubstP* α*1* β} ⊢ *PfP* β
proof −
obtain *k1*::*name* **and** *s1*::*name* **and** *k2*::*name* **and** *s2*::*name* **and** *k*::*name* **and** *s*::*name*
where *atom s1* ♯ (α*1*,α*2*,β) *atom k1* ♯ (α*1*,α*2*,β,*s1*)
atom s2 ♯ (α*1*,α*2*,β,*k1*,*s1*)*atom k2* ♯ (α*1*,α*2*,β,*s2*,*k1*,*s1*)
atom s ♯ (α*1*,α*2*,β,*k2*,*s2*,*k1*,*s1*)
atom k ♯ (α*1*,α*2*,β,*s*,*k2*,*s2*,*k1*,*s1*)
by (*metis obtain-fresh*)
thus *?thesis*
apply (*simp add*: *PfP.simps* [*of k s* β] *PfP.simps* [*of k1 s1* α*1*] *PfP.simps* [*of k2 s2* α*2*])
apply (*auto intro*!: *PrfP-inference* [*of s k Var s1 Var s2*, *THEN cut3*] *del*: *Disj-EH*)
done
qed

theorem *PfP-implies-SubstForm-PfP*:
assumes *H* ⊢ *PfP y H* ⊢ *SubstFormP x t y z*
shows *H* ⊢ *PfP z*
proof −
obtain *u*::*name* **and** *v*::*name*
where *atoms*: *atom u* ♯ (*t*,*x*,*y*,*z*,*v*) *atom v* ♯ (*t*,*x*,*y*,*z*)
by (*metis obtain-fresh*)
show *?thesis*
apply (*rule PfP-inference* [*of y*, *THEN cut3*])
apply (*rule assms*)+
using *atoms*
apply (*auto simp*: *SubstP.simps* [*of u - - v*] *intro*!: *Disj-I2*)
apply (*rule Ex-I* [**where** *x*=*x*], *simp*)
apply (*rule Ex-I* [**where** *x*=*t*], *simp add*: *assms*)
done
qed

theorem *PfP-implies-ModPon-PfP*: ⟦*H* ⊢ *PfP* (*Q-Imp x y*); *H* ⊢ *PfP x*⟧ ⟹ *H* ⊢ *PfP y*
by (*force intro*: *PfP-inference* [*of x*, *THEN cut3*] *Disj-I1 simp add*: *ModPonP-def*)

corollary *PfP-implies-ModPon-PfP-quot*: ⟦*H* ⊢ *PfP* ⌈α *IMP* β⌉; *H* ⊢ *PfP* ⌈α⌉⟧ ⟹ *H* ⊢ *PfP* ⌈β⌉
by (*auto simp*: *quot-fm-def intro*: *PfP-implies-ModPon-PfP*)

end

Chapter 10

Pseudo-Coding: Section 7 Material

theory *Pseudo-Coding*
imports *II-Prelims*
begin

10.1 General Lemmas

lemma *Collect-disj-Un*: $\{f\ i\ |i.\ P\ i \lor Q\ i\} = \{f\ i\ |i.\ P\ i\} \cup \{f\ i\ |i.\ Q\ i\}$
by *auto*

abbreviation *Q-Subset* :: $tm \Rightarrow tm \Rightarrow tm$
  where *Q-Subset t u* $\equiv$ (*Q-All* (*Q-Imp* (*Q-Mem* (*Q-Ind Zero*) *t*) (*Q-Mem* (*Q-Ind Zero*) *u*)))

lemma *NEQ-quot-tm*: $i \neq j \Longrightarrow \{\} \vdash \lceil Var\ i \rceil\ NEQ\ \lceil Var\ j \rceil$
  by (*auto intro*: *Sigma-fm-imp-thm* [*OF OrdNotEqP-sf*]
        simp: *ground-fm-aux-def supp-conv-fresh quot-tm-def*)

lemma *EQ-quot-tm-Fls*: $i \neq j \Longrightarrow$ *insert* ($\lceil Var\ i \rceil\ EQ\ \lceil Var\ j \rceil$) $H \vdash Fls$
  by (*metis* (*full-types*) *NEQ-quot-tm Assume OrdNotEqP-E cut2 thin0*)

lemma *perm-commute*: $a \sharp p \Longrightarrow a' \sharp p \Longrightarrow (a \rightleftharpoons a') + p = p + (a \rightleftharpoons a')$
  by (*rule plus-perm-eq*) (*simp add*: *supp-swap fresh-def*)

lemma *perm-self-inverseI*: $[\![-p = q;\ a \sharp p;\ a' \sharp p]\!] \Longrightarrow -((a \rightleftharpoons a') + p) = (a \rightleftharpoons a') + q$
  by (*simp-all add*: *perm-commute fresh-plus-perm minus-add*)

lemma *fresh-image*:
  fixes $f :: 'a \Rightarrow 'b::fs$ **shows** *finite A* $\Longrightarrow i \sharp f\ `\ A \longleftrightarrow (\forall x \in A.\ i \sharp f\ x)$
  by (*induct rule*: *finite-induct*) (*auto simp*: *fresh-finite-insert*)

lemma *atom-in-atom-image* [*simp*]: *atom* $j \in$ *atom* ‘ $V \longleftrightarrow j \in V$
by *auto*

lemma *fresh-star-empty* [*simp*]: $\{\} \sharp* bs$
by (*simp add*: *fresh-star-def*)

declare *fresh-star-insert* [*simp*]

lemma *fresh-star-finite-insert*:
fixes S :: (′*a*::*fs*) *set* **shows** *finite* $S \Longrightarrow a \sharp*$ *insert* $x\ S \longleftrightarrow a \sharp* x \wedge a \sharp* S$
by (*auto simp*: *fresh-star-def fresh-finite-insert*)

lemma *fresh-finite-Diff-single* [*simp*]:
fixes V :: *name set* **shows** *finite* $V \Longrightarrow a \sharp (V - \{j\}) \longleftrightarrow (a \sharp j \longrightarrow a \sharp V)$
apply (*auto simp*: *fresh-finite-insert*)
apply (*metis finite-Diff fresh-finite-insert insert-Diff-single*)
apply (*metis Diff-iff finite-Diff fresh-atom fresh-atom-at-base fresh-finite-set-at-base insertI1*)
apply (*metis Diff-idemp Diff-insert-absorb finite-Diff fresh-finite-insert insert-Diff-single insert-absorb*)
done

lemma *fresh-image-atom* [*simp*]: *finite* $A \Longrightarrow i \sharp$ *atom* ‘ $A \longleftrightarrow i \sharp A$
by (*induct rule*: *finite-induct*) (*auto simp*: *fresh-finite-insert*)

lemma *atom-fresh-star-atom-set-conv*: ⟦*atom* $i \sharp bs$; *finite bs*⟧ $\Longrightarrow bs \sharp* i$
by (*metis fresh-finite-atom-set fresh-ineq-at-base fresh-star-def*)

lemma *notin-V*:
assumes p: *atom* $i \sharp p$ **and** V: *finite* V *atom* ‘ $(p \cdot V) \sharp* V$
shows $i \notin V\ i \notin p \cdot V$
using V
apply (*auto simp*: *fresh-def fresh-star-def supp-finite-set-at-base*)
apply (*metis p mem-permute-iff fresh-at-base-permI*)+
done

10.2 Simultaneous Substitution

definition *ssubst* :: *tm* ⇒ *name set* ⇒ (*name* ⇒ *tm*) ⇒ *tm*
where *ssubst t V F* = *Finite-Set.fold* (λi. *subst i* (*F i*)) *t V*

definition *make-F* :: *name set* ⇒ *perm* ⇒ *name* ⇒ *tm*
where *make-F Vs p* ≡ λi. *if* $i \in Vs$ *then Var* $(p \cdot i)$ *else Var i*

lemma *ssubst-empty* [*simp*]: *ssubst t* {} *F* = *t*
by (*simp add*: *ssubst-def*)

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* $\Rightarrow$ *tm*
assumes p: *atom* ' $(p \cdot Vs)$ $\sharp*$ *Vs*
and *pinv*: $-p = p$
and *Vs*: *finite Vs*
defines $F \equiv$ *make-F Vs p*
begin

lemma *F-unfold*: $F\ i = ($*if* $i \in Vs$ *then Var* $(p \cdot i)$ *else Var* $i)$
by (*simp add*: *F-def make-F-def*)

lemma *finite-V* [*simp*]: $V \subseteq Vs \Longrightarrow$ *finite* V
by (*metis Vs finite-subset*)

lemma *perm-exits-Vs*: $i \in Vs \Longrightarrow (p \cdot i) \notin Vs$
by (*metis Vs fresh-finite-set-at-base imageI fresh-star-def mem-permute-iff p*)

lemma *atom-fresh-perm*: $[\![x \in Vs;\ y \in Vs]\!] \Longrightarrow$ *atom* $x \sharp p \cdot y$
by (*metis imageI Vs p fresh-finite-set-at-base fresh-star-def mem-permute-iff fresh-at-base*(*2*))

lemma *fresh-pj*: $[\![a \sharp p;\ j \in Vs]\!] \Longrightarrow a \sharp p \cdot j$
by (*metis atom-fresh-perm fresh-at-base*(*2*) *fresh-perm fresh-permute-left pinv*)

lemma *fresh-Vs*: $a \sharp p \Longrightarrow a \sharp Vs$
by (*metis Vs fresh-def fresh-perm fresh-permute-iff fresh-star-def p permute-finite supp-finite-set-at-base*)

lemma *fresh-pVs*: $a \sharp p \Longrightarrow a \sharp p \cdot Vs$
by (*metis fresh-Vs fresh-perm fresh-permute-left pinv*)

lemma assumes $V \subseteq Vs$ $a \sharp p$
shows *fresh-pV* [*simp*]: $a \sharp p \cdot V$ **and** *fresh-V* [*simp*]: $a \sharp V$
using *fresh-pVs fresh-Vs assms*
apply (*auto simp*: *fresh-def*)
apply (*metis* (*full-types*) *Vs finite-V permute-finite set-mp subset-Un-eq supp-of-finite-union union-eqvt*)
by (*metis Vs finite-V set-mp subset-Un-eq supp-of-finite-union*)

lemma *qp-insert*:
fixes i::*name* **and** i'::*name*
assumes *atom* $i \sharp p$ *atom* $i' \sharp (i,p)$
shows *quote-perm* $(($*atom* $i \rightleftharpoons$ *atom* $i') + p)$ (*insert* i *Vs*)
using *p pinv Vs assms*
by (*auto simp*: *quote-perm-def fresh-at-base-permI atom-fresh-star-atom-set-conv swap-fresh-fresh*
fresh-star-finite-insert fresh-finite-insert perm-self-inverseI)

lemma *subst-F-left-commute*: *subst* x $(F\ x)$ (*subst* y $(F\ y)$ t) = *subst* y $(F\ y)$ (*subst* x $(F\ x)$ t)
by (*metis subst-tm-commute2 F-unfold subst-tm-id F-unfold atom-fresh-perm*

tm.fresh(*2*))

lemma
assumes *finite V i* $\notin$ *V*
shows *ssubst-insert*: *ssubst t* (*insert i V*) *F* = *subst i* (*F i*) (*ssubst t V F*) (**is** *?thesis1*)
and *ssubst-insert2*: *ssubst t* (*insert i V*) *F* = *ssubst* (*subst i* (*F i*) *t*) *V F* (**is** *?thesis2*)
proof –
interpret *comp-fun-commute* (λ*i*. *subst i* (*F i*))
proof qed (*simp add*: *subst-F-left-commute fun-eq-iff*)
show *?thesis1* **using** *assms Vs*
by (*simp add*: *ssubst-def*)
show *?thesis2* **using** *assms Vs*
by (*simp add*: *ssubst-def fold-insert2 del*: *fold-insert*)
qed

lemma *ssubst-insert-if*:
finite V $\Longrightarrow$
ssubst t (*insert i V*) *F* = (*if i* $\in$ *V then ssubst t V F*
else subst i (*F i*) (*ssubst t V F*))
by (*simp add*: *ssubst-insert insert-absorb*)

lemma *ssubst-single* [*simp*]: *ssubst t* {*i*} *F* = *subst i* (*F i*) *t*
by (*simp add*: *ssubst-insert*)

lemma *ssubst-Var-if* [*simp*]:
assumes *finite V*
shows *ssubst* (*Var i*) *V F* = (*if i* $\in$ *V then F i else Var i*)
using *assms*
apply (*induction V*, *auto*)
apply (*metis ssubst-insert subst.simps*(*2*))
apply (*metis ssubst-insert2 subst.simps*(*2*))+
done

lemma *ssubst-Zero* [*simp*]: *finite V* $\Longrightarrow$ *ssubst Zero V F* = *Zero*
by (*induct V rule*: *finite-induct*) (*auto simp*: *ssubst-insert*)

lemma *ssubst-Eats* [*simp*]: *finite V* $\Longrightarrow$ *ssubst* (*Eats t u*) *V F* = *Eats* (*ssubst t V F*) (*ssubst u V F*)
by (*induct V rule*: *finite-induct*) (*auto simp*: *ssubst-insert*)

lemma *ssubst-SUCC* [*simp*]: *finite V* $\Longrightarrow$ *ssubst* (*SUCC t*) *V F* = *SUCC* (*ssubst t V F*)
by (*metis SUCC-def ssubst-Eats*)

lemma *ssubst-ORD-OF* [*simp*]: *finite V* $\Longrightarrow$ *ssubst* (*ORD-OF n*) *V F* = *ORD-OF n*
by (*induction n*) *auto*

lemma *ssubst-HPair* [*simp*]:
finite V $\Longrightarrow$ *ssubst* (*HPair t u*) *V F* = *HPair* (*ssubst t V F*) (*ssubst u V F*)
by (*simp add*: *HPair-def*)

lemma *ssubst-HTuple* [*simp*]: *finite V* $\Longrightarrow$ *ssubst* (*HTuple n*) *V F* = (*HTuple n*)
by (*induction n*) (*auto simp*: *HTuple.simps*)

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 −
obtain *i*::*name* **where** *atom i* ♯ (*t*,*u*)
by (*rule obtain-fresh*)
thus *?thesis* **using** *assms*
by (*auto simp*: *Subset.simps* [*of i*] *vquot-fm-def vquot-tm-def trans-tm-forget*)
qed

lemma *fresh-ssubst*:
assumes *finite V a* ♯ *p* · *V a* ♯ *t*
shows *a* ♯ *ssubst t V F*
using *assms*
by (*induct V*)
(*auto simp*: *ssubst-insert-if fresh-finite-insert F-unfold intro*: *fresh-ineq-at-base*)

lemma *fresh-ssubst'*:
assumes *finite V atom i* ♯ *t atom* (*p* · *i*) ♯ *t*
shows *atom i* ♯ *ssubst t V F*
using *assms*
by (*induct t rule*: *tm.induct*) (*auto simp*: *F-unfold fresh-permute-left pinv*)

lemma *ssubst-vquot-Ex*:
⟦*finite V*; *atom i* ♯ *p* · *V*⟧
$\Longrightarrow$ *ssubst* ⌊*Ex i A*⌋(*insert i V*) (*insert i V*) *F* = *ssubst* ⌊*Ex i A*⌋*V V F*
by (*simp add*: *ssubst-insert-if insert-absorb vquot-fm-insert fresh-ssubst*)

lemma *ground-ssubst-eq*: ⟦*finite V*; *supp t* = {}⟧ $\Longrightarrow$ *ssubst t V F* = *t*
by (*induct V rule*: *finite-induct*) (*auto simp*: *ssubst-insert fresh-def*)

lemma *ssubst-quot-tm* [*simp*]:
fixes *t*::*tm* **shows** *finite V* $\Longrightarrow$ *ssubst* ⌈*t*⌉ *V F* = ⌈*t*⌉
by (*simp add*: *ground-ssubst-eq supp-conv-fresh*)

lemma *ssubst-quot-fm* [*simp*]:
fixes *A*::*fm* **shows** *finite V* $\Longrightarrow$ *ssubst* ⌈*A*⌉ *V F* = ⌈*A*⌉
by (*simp add*: *ground-ssubst-eq supp-conv-fresh*)

lemma *atom-in-p-Vs*: ⟦*i* ∈ *p* · *V*; *V* ⊆ *Vs*⟧ $\Longrightarrow$ *i* ∈ *p* · *Vs*
by (*metis* (*full-types*) *True-eqvt set-mp subset-eqvt*)

10.3 The Main Theorems of Section 7

lemma *SubstTermP-vquot-dbtm*:
  assumes *w*: *w* ∈ *Vs* − *V* **and** *V*: *V* ⊆ *Vs V ′* = *p* • *V*
    and *s*: *supp dbtm* ⊆ *atom ‘ Vs*
  shows
  insert (*ConstP* (*F w*)) {*ConstP* (*F i*) | *i*. *i* ∈ *V*}
  ⊢ *SubstTermP* ⌈*Var w*⌉ (*F w*)
        (*ssubst* (*vquot-dbtm V dbtm*) *V F*)
        (*subst w* (*F w*) (*ssubst* (*vquot-dbtm* (*insert w V*) *dbtm*) *V F*))
using *s*
proof (*induct dbtm rule*: *dbtm.induct*)
  case *DBZero* **thus** *?case* **using** *V w*
    by (*auto intro*: *SubstTermP-Zero* [*THEN cut1*] *ConstP-imp-TermP* [*THEN cut1*])
next
  case (*DBInd n*) **thus** *?case* **using** *V*
    apply *auto*
    apply (*rule thin* [*of* {*ConstP* (*F w*)}])
    apply (*rule SubstTermP-Ind* [*THEN cut3*])
    apply (*auto simp*: *IndP-Q-Ind OrdP-ORD-OF ConstP-imp-TermP*)
    done
next
  case (*DBVar i*) **show** *?case*
  proof (*cases i* ∈ *V ′*)
    case *True* **hence** *i* ∉ *Vs* **using** *assms*
      by (*metis p Vs atom-in-atom-image atom-in-p-Vs fresh-finite-set-at-base fresh-star-def*)
    thus *?thesis* **using** *DBVar True V*
      by *auto*
  next
    case *False* **thus** *?thesis* **using** *DBVar V w*
      apply (*auto simp*: *quot-Var* [*symmetric*])
        apply (*blast intro*: *thin* [*of* {*ConstP* (*F w*)}] *ConstP-imp-TermP*
                SubstTermP-Var-same [*THEN cut2*])
       apply (*subst forget-subst-tm*, *metis F-unfold atom-fresh-perm tm.fresh*(*2*))
       apply (*blast intro*: *Hyp thin* [*of* {*ConstP* (*F w*)}] *ConstP-imp-TermP*
                SubstTermP-Const [*THEN cut2*])
        apply (*blast intro*: *Hyp thin* [*of* {*ConstP* (*F w*)}] *ConstP-imp-TermP EQ-quot-tm-Fls*
                SubstTermP-Var-diff [*THEN cut4*])
      done
  qed
next
  case (*DBEats tm1 tm2*) **thus** *?case* **using** *V*
    by (*auto simp*: *SubstTermP-Eats* [*THEN cut2*])
qed

lemma *SubstFormP-vquot-dbfm*:

assumes w: $w \in Vs - V$ **and** V: $V \subseteq Vs$ $V' = p \cdot V$
and s: $supp\ dbfm \subseteq atom$ ` Vs
shows
insert (*ConstP* (*F w*)) {*ConstP* (*F i*) | *i*. $i \in V$}
$\vdash$ *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*))
using *w s*
proof (*induct dbfm rule*: *dbfm.induct*)
case (*DBMem t u*) **thus** *?case* **using** *V*
by (*auto intro*: *SubstTermP-vquot-dbtm SubstFormP-Mem* [*THEN cut2*])
next
case (*DBEq t u*) **thus** *?case* **using** *V*
by (*auto intro*: *SubstTermP-vquot-dbtm SubstFormP-Eq* [*THEN cut2*])
next
case (*DBDisj A B*) **thus** *?case* **using** *V*
by (*auto intro*: *SubstFormP-Disj* [*THEN cut2*])
next
case (*DBNeg A*) **thus** *?case* **using** *V*
by (*auto intro*: *SubstFormP-Neg* [*THEN cut1*])
next
case (*DBEx A*) **thus** *?case* **using** *V*
by (*auto intro*: *SubstFormP-Ex* [*THEN cut1*])
qed

Lemmas 7.5 and 7.6

lemma *ssubst-SubstFormP*:
fixes *A*::*fm*
assumes w: $w \in Vs - V$ **and** V: $V \subseteq Vs$ $V' = p \cdot V$
and s: $supp\ A \subseteq atom$ ` Vs
shows
insert (*ConstP* (*F w*)) {*ConstP* (*F i*) | *i*. $i \in V$}
$\vdash$ *SubstFormP* ⌈*Var w*⌉ (*F w*)
(*ssubst* ⌊*A*⌋ *V V F*)
(*ssubst* ⌊*A*⌋(*insert w V*) (*insert w V*) *F*)
proof −
have $w \notin V$ **using** *assms*
by *auto*
thus *?thesis* **using** *assms*
by (*simp add*: *vquot-fm-def supp-conv-fresh ssubst-insert-if SubstFormP-vquot-dbfm*)
qed

Theorem 7.3

theorem *PfP-implies-PfP-ssubst*:
fixes β::*fm*
assumes β: {} $\vdash$ *PfP* ⌈β⌉
and V: $V \subseteq Vs$
and s: $supp\ \beta \subseteq atom$ ` Vs
shows {*ConstP* (*F i*) | *i*. $i \in V$} $\vdash$ *PfP* (*ssubst* ⌊β⌋ *V V F*)

proof –
  show *?thesis* **using** *finite-V* [*OF V*] *V*
  proof *induction*
    case *empty* **thus** *?case*
      by (*auto simp*: β)
  next
    case (*insert i V*)
    thus *?case* **using** *assms*
      by (*auto simp*: *Collect-disj-Un fresh-finite-set-at-base*
            intro: *PfP-implies-SubstForm-PfP thin1 ssubst-SubstFormP*)
  qed
qed

end

end

Chapter 11

Quotations of the Free Variables

theory *Quote*
imports *Pseudo-Coding*
begin

11.1 Sequence version of the "Special p-Function, F*"

The definition below describes a relation, not a function. This material relates to Section 8, but omits the ordering of the universe.

definition *SeqQuote* :: *hf* ⇒ *hf* ⇒ *hf* ⇒ *hf* ⇒ *bool*
where *SeqQuote x x′ s k* ≡
  BuildSeq2 (λ*y y′*. *y*=*0* ∧ *y′* = *0*)
    (λ*u u′ v v′ w w′*. *u* = *v* ◁ *w* ∧ *u′* = *q-Eats v′ w′*) *s k x x′*

11.1.1 Defining the syntax: quantified body

nominal-function *SeqQuoteP* :: *tm* ⇒ *tm* ⇒ *tm* ⇒ *tm* ⇒ *fm*
 where ⟦*atom l* ♯ (*s*,*k*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*);
   atom sl ♯ (*s*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*); *atom sl′* ♯ (*s*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*);
   atom m ♯ (*s*,*n*,*sm*,*sm′*,*sn*,*sn′*); *atom n* ♯ (*s*,*sm*,*sm′*,*sn*,*sn′*);
   atom sm ♯ (*s*,*sm′*,*sn*,*sn′*); *atom sm′* ♯ (*s*,*sn*,*sn′*);
   atom sn ♯ (*s*,*sn′*); *atom sn′* ♯ *s*⟧ ⟹
  SeqQuoteP t u s k =
   LstSeqP s k (*HPair t u*) *AND*
   All2 l (*SUCC k*) (*Ex sl* (*Ex sl′* (*HPair* (*Var l*) (*HPair* (*Var sl*) (*Var sl′*)) *IN s AND*
     ((*Var sl EQ Zero AND Var sl′ EQ Zero*) *OR*
     Ex m (*Ex n* (*Ex sm* (*Ex sm′* (*Ex sn* (*Ex sn′* (*Var m IN Var l AND Var n IN Var l AND*
       HPair (*Var m*) (*HPair* (*Var sm*) (*Var sm′*)) *IN s AND*

HPair (*Var n*) (*HPair* (*Var sn*) (*Var sn′*)) *IN s AND*
Var sl EQ Eats (*Var sm*) (*Var sn*) *AND*
Var sl′ EQ Q-Eats (*Var sm′*) (*Var sn′*)))))))))))
by (*auto simp*: *eqvt-def SeqQuoteP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *SeqQuoteP-fresh-iff* [*simp*]:
$a \sharp$ *SeqQuoteP t u s k* $\longleftrightarrow a \sharp t \wedge a \sharp u \wedge a \sharp s \wedge a \sharp k$ (**is** *?thesis1*)
and *eval-fm-SeqQuoteP* [*simp*]:
eval-fm e (*SeqQuoteP t u s k*) $\longleftrightarrow$ *SeqQuote* $[\![t]\!]e$ $[\![u]\!]e$ $[\![s]\!]e$ $[\![k]\!]e$ (**is** *?thesis2*)
and *SeqQuoteP-sf* [*iff*]:
Sigma-fm (*SeqQuoteP t u s k*) (**is** *?thsf*)
and *SeqQuoteP-imp-OrdP*:
{ *SeqQuoteP t u s k* } $\vdash$ *OrdP k* (**is** *?thord*)
and *SeqQuoteP-imp-LstSeqP*:
{ *SeqQuoteP t u s k* } $\vdash$ *LstSeqP s k* (*HPair t u*) (**is** *?thlstseq*)
proof −
obtain *l*::*name* **and** *sl*::*name* **and** *sl′*::*name* **and** *m*::*name* **and** *n*::*name* **and**
sm::*name* **and** *sm′*::*name* **and** *sn*::*name* **and** *sn′*::*name*
where *atoms*:
atom l $\sharp$ (*s*,*k*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
atom sl $\sharp$ (*s*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*) *atom sl′* $\sharp$ (*s*,*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*
by (*metis obtain-fresh*)
thus *?thesis1 ?thsf ?thord ?thlstseq*
by *auto* (*auto simp*: *LstSeqP.simps*)
show *?thesis2* **using** *atoms*
by (*force simp add*: *LstSeq-imp-Ord SeqQuote-def*
BuildSeq2-def BuildSeq-def Builds-def HBall-def q-Eats-def
Seq-iff-app [*of* $[\![s]\!]e$, *OF LstSeq-imp-Seq-succ*]
Ord-trans [*of - - succ* $[\![k]\!]e$]
cong: *conj-cong*)
qed

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 −
obtain *l*::*name* **and** *sl*::*name* **and** *sl′*::*name* **and** *m*::*name* **and** *n*::*name* **and**
sm::*name* **and** *sm′*::*name* **and** *sn*::*name* **and** *sn′*::*name*
where *atom l* $\sharp$ (*s*,*k*,*w*,*j*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
atom sl $\sharp$ (*s*,*w*,*j*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*) *atom sl′* $\sharp$ (*s*,*w*,*j*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
atom m $\sharp$ (*s*,*w*,*j*,*n*,*sm*,*sm′*,*sn*,*sn′*) *atom n* $\sharp$ (*s*,*w*,*j*,*sm*,*sm′*,*sn*,*sn′*)
atom sm $\sharp$ (*s*,*w*,*j*,*sm′*,*sn*,*sn′*) *atom sm′* $\sharp$ (*s*,*w*,*j*,*sn*,*sn′*)

atom sn ♯ (*s*,*w*,*j*,*sn′*) *atom sn′* ♯ (*s*,*w*,*j*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*force simp add*: *SeqQuoteP.simps* [*of l - - sl sl′ m n sm sm′ sn sn′*])
qed

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

11.1.2 Correctness properties

lemma *SeqQuoteP-lemma*:
fixes *m*::*name* **and** *sm*::*name* **and** *sm′*::*name* **and** *n*::*name* **and** *sn*::*name* **and** *sn′*::*name*
assumes *atom m* ♯ (*t*,*u*,*s*,*k*,*n*,*sm*,*sm′*,*sn*,*sn′*) *atom n* ♯ (*t*,*u*,*s*,*k*,*sm*,*sm′*,*sn*,*sn′*)
atom sm ♯ (*t*,*u*,*s*,*k*,*sm′*,*sn*,*sn′*) *atom sm′* ♯ (*t*,*u*,*s*,*k*,*sn*,*sn′*)
atom sn ♯ (*t*,*u*,*s*,*k*,*sn′*) *atom sn′* ♯ (*t*,*u*,*s*,*k*)
shows { *SeqQuoteP t u s k* }
⊢ (*t EQ Zero AND u EQ Zero*) *OR*
Ex m (*Ex n* (*Ex sm* (*Ex sm′* (*Ex sn* (*Ex sn′* (*Var m IN k AND Var n IN k AND*
SeqQuoteP (*Var sm*) (*Var sm′*) *s* (*Var m*) *AND*
SeqQuoteP (*Var sn*) (*Var sn′*) *s* (*Var n*) *AND*
t EQ Eats (*Var sm*) (*Var sn*) *AND*
u EQ Q-Eats (*Var sm′*) (*Var sn′*)))))))
proof −
obtain *l*::*name* **and** *sl*::*name* **and** *sl′*::*name*
where *atom l* ♯ (*t*,*u*,*s*,*k*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
atom sl ♯ (*t*,*u*,*s*,*k*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
atom sl′ ♯ (*t*,*u*,*s*,*k*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
by (*metis obtain-fresh*)
thus *?thesis* **using** *assms*
apply (*simp add*: *SeqQuoteP.simps* [*of l s k sl sl′ m n sm sm′ sn sn′*])
apply (*rule Conj-EH Ex-EH All2-SUCC-E* [*THEN rotate2*] | *simp*)+
apply (*rule cut-same* [**where** *A* = *HPair t u EQ HPair* (*Var sl*) (*Var sl′*)])
apply (*metis Assume AssumeH*(*4*) *LstSeqP-EQ*)
apply *clarify*
apply (*rule Disj-EH*)
apply (*rule Disj-I1*)
apply (*rule anti-deduction*)
apply (*rule Var-Eq-subst-Iff* [*THEN Sym-L*, *THEN Iff-MP-same*])
apply (*rule rotate2*)
apply (*rule Var-Eq-subst-Iff* [*THEN Sym-L*, *THEN Iff-MP-same*], *force*)
— now the quantified case
apply (*rule Ex-EH Conj-EH*)+
apply *simp-all*
apply (*rule Disj-I2*)
apply (*rule Ex-I* [**where** *x* = *Var m*], *simp*)
apply (*rule Ex-I* [**where** *x* = *Var n*], *simp*)
apply (*rule Ex-I* [**where** *x* = *Var sm*], *simp*)

apply (*rule Ex-I* [**where** x = *Var sm'*], *simp*)
apply (*rule Ex-I* [**where** x = *Var sn*], *simp*)
apply (*rule Ex-I* [**where** x = *Var sn'*], *simp*)
apply (*simp-all add*: *SeqQuoteP.simps* [*of l s - sl sl′ m n sm sm′ sn sn'*])
apply ((*rule Conj-I*)+, *blast intro*: *LstSeqP-Mem*)+
— first SeqQuoteP subgoal
apply (*rule All2-Subset* [*OF Hyp*])
apply (*blast intro*!: *SUCC-Subset-Ord LstSeqP-OrdP*)+
apply *simp*
— next SeqQuoteP subgoal
apply ((*rule Conj-I*)+, *blast intro*: *LstSeqP-Mem*)+
apply (*rule All2-Subset* [*OF Hyp*], *blast*)
apply (*auto intro*!: *SUCC-Subset-Ord LstSeqP-OrdP intro*: *Trans*)
done
qed

11.2 The "special function" itself

definition *Quote* :: *hf* ⇒ *hf* ⇒ *bool*
where *Quote x x′* ≡ ∃ *s k*. *SeqQuote x x′ s k*

11.2.1 Defining the syntax

nominal-function *QuoteP* :: *tm* ⇒ *tm* ⇒ *fm*
where ⟦*atom s* ♯ (*t*,*u*,*k*); *atom k* ♯ (*t*,*u*)⟧ ⟹
QuoteP t u = *Ex s* (*Ex k* (*SeqQuoteP t u* (*Var s*) (*Var k*)))
by (*auto simp*: *eqvt-def QuoteP-graph-aux-def flip-fresh-fresh*) (*metis obtain-fresh*)

nominal-termination (*eqvt*)
by *lexicographic-order*

lemma
shows *QuoteP-fresh-iff* [*simp*]: *a* ♯ *QuoteP t u* ⟷ *a* ♯ *t* ∧ *a* ♯ *u* (**is** *?thesis1*)
and *eval-fm-QuoteP* [*simp*]: *eval-fm e* (*QuoteP t u*) ⟷ *Quote* ⟦*t*⟧*e* ⟦*u*⟧*e* (**is** *?thesis2*)
and *QuoteP-sf* [*iff*]: *Sigma-fm* (*QuoteP t u*) (**is** *?thsf*)
proof −
obtain *s*::*name* **and** *k*::*name* **where** *atom s* ♯ (*t*,*u*,*k*) *atom k* ♯ (*t*,*u*)
by (*metis obtain-fresh*)
thus *?thesis1 ?thesis2 ?thsf*
by (*auto simp*: *Quote-def*)
qed

lemma *QuoteP-subst* [*simp*]:
(*QuoteP t u*)(*j*::=*w*) = *QuoteP* (*subst j w t*) (*subst j w u*)
proof −
obtain *s*::*name* **and** *k*::*name* **where** *atom s* ♯ (*t*,*u*,*w*,*j*,*k*) *atom k* ♯ (*t*,*u*,*w*,*j*)
by (*metis obtain-fresh*)

thus *?thesis*
  by (*simp add*: *QuoteP.simps* [*of s - - k*])
qed

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

11.2.2 Correctness properties

lemma *Quote-0*: *Quote 0 0*
  by (*auto simp*: *Quote-def SeqQuote-def intro*: *BuildSeq2-exI*)

lemma *QuoteP-Zero*: {} ⊢ *QuoteP Zero Zero*
  by (*auto intro*: *Sigma-fm-imp-thm* [*OF QuoteP-sf*]
      simp: *ground-fm-aux-def supp-conv-fresh Quote-0*)

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 −
 obtain *km*::*name* **and** *kn*::*name* **and** *j*::*name* **and** *k′*::*name* **and** *l*::*name*
  and *sl*::*name* **and** *sl′*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name*
  and *sm′*::*name* **and** *sn*::*name* **and** *sn′*::*name*
 where *atoms2*:
    atom km ♯ (*kn*,*j*,*k′*,*l*,*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
    atom kn ♯ (*j*,*k′*,*l*,*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
    atom j ♯ (*k′*,*l*,*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
  and *atoms*: *atom k′* ♯ (*l*,*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
    atom l ♯ (*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*,*sl*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
    atom sl ♯ (*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*,*sl′*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
    atom sl′ ♯ (*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*,*m*,*n*,*sm*,*sm′*,*sn*,*sn′*)
    atom m ♯ (*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*,*n*,*sm*,*sm′*,*sn*,*sn′*)
    atom n ♯ (*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*,*sm*,*sm′*,*sn*,*sn′*)
    atom sm ♯ (*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*,*sm′*,*sn*,*sn′*)
    atom sm′ ♯ (*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*,*sn*,*sn′*)
    atom sn ♯ (*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*,*sn′*)
    atom sn′ ♯ (*s1*,*s2*,*s*,*k1*,*k2*,*k*,*t1*,*t2*,*u1*,*u2*)
  by (*metis obtain-fresh*)
 show *?thesis*
  using *assms atoms*
  apply (*auto simp*: *SeqQuoteP.simps* [*of l Var s - sl sl′ m n sm sm′ sn sn′*])
  apply (*rule cut-same* [**where** *A*=*OrdP k1 AND OrdP k2*])
  apply (*metis Conj-I SeqQuoteP-imp-OrdP thin1 thin2*)
  apply (*rule cut-same* [*OF exists-SeqAppendP* [*of s s1 SUCC k1 s2 SUCC k2*]])
  apply (*rule AssumeH Ex-EH Conj-EH* | *simp*)+
  apply (*rule cut-same* [*OF exists-HaddP* [**where** *j*=*k′* **and** *x*=*k1* **and** *y*=*k2*]])
  apply (*rule AssumeH Ex-EH Conj-EH* | *simp*)+
  apply (*rule Ex-I* [**where** *x*=*Eats* (*Var s*) (*HPair* (*SUCC* (*SUCC* (*Var k′*)))
(*HPair* (*Eats t1 t2*) (*Q-Eats u1 u2*)))])

```
  apply (simp-all (no-asm-simp))
  apply (rule Ex-I [where x=SUCC (SUCC (Var k'))])
  apply simp
  apply (rule Conj-I [OF LstSeqP-SeqAppendP-Eats])
  apply (blast intro: SeqQuoteP-imp-LstSeqP [THEN cut1])+
  proof (rule All2-SUCC-I, simp-all)
   show {HaddP k1 k2 (Var k'), OrdP k1, OrdP k2, SeqAppendP s1 (SUCC
k1) s2 (SUCC k2) (Var s),
       SeqQuoteP t1 u1 s1 k1, SeqQuoteP t2 u2 s2 k2}
      ⊢ Ex sl (Ex sl'
         (HPair (SUCC (SUCC (Var k'))) (HPair (Var sl) (Var sl')) IN
         Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats t1 t2)
(Q-Eats u1 u2))) 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 SUCC (SUCC (Var k')) AND
             Var n IN SUCC (SUCC (Var k')) AND
             HPair (Var m) (HPair (Var sm) (Var sm')) IN
             Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats t1
t2) (Q-Eats u1 u2))) AND
             HPair (Var n) (HPair (Var sn) (Var sn')) IN
             Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats t1
t2) (Q-Eats u1 u2))) AND
             Var sl EQ Eats (Var sm) (Var sn) AND Var sl' EQ Q-Eats (Var
sm') (Var sn')))))))))
    — verifying the final values
    apply (rule Ex-I [where x=Eats t1 t2])
    using assms atoms apply simp
    apply (rule Ex-I [where x=Q-Eats u1 u2], simp)
    apply (rule Conj-I [OF Mem-Eats-I2 [OF Refl]])
    apply (rule Disj-I2)
    apply (rule Ex-I [where x=k1], simp)
    apply (rule Ex-I [where x=SUCC (Var k')], simp)
    apply (rule Ex-I [where x=t1], simp)
    apply (rule Ex-I [where x=u1], simp)
    apply (rule Ex-I [where x=t2], simp)
    apply (rule Ex-I [where x=u2], simp)
    apply (rule Conj-I)
    apply (blast intro: HaddP-Mem-I Mem-SUCC-I1)
    apply (rule Conj-I [OF Mem-SUCC-Refl])
    apply (rule Conj-I)
  apply (blast intro: Mem-Eats-I1 SeqAppendP-Mem1 [THEN cut3] Mem-SUCC-Refl

         SeqQuoteP-imp-LstSeqP [THEN cut1] LstSeqP-imp-Mem)
  apply (blast intro: Mem-Eats-I1 SeqAppendP-Mem2 [THEN cut4] Mem-SUCC-Refl

       SeqQuoteP-imp-LstSeqP [THEN cut1] LstSeqP-imp-Mem HaddP-SUCC1
[THEN cut1])
    done
```

```
next
  show {HaddP k1 k2 (Var k'), OrdP k1, OrdP k2, SeqAppendP s1 (SUCC
k1) s2 (SUCC k2) (Var s),
        SeqQuoteP t1 u1 s1 k1, SeqQuoteP t2 u2 s2 k2}
       ⊢ All2 l (SUCC (SUCC (Var k')))
          (Ex sl (Ex sl'
            (HPair (Var l) (HPair (Var sl) (Var sl')) IN
             Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats t1
t2) (Q-Eats u1 u2))) 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
                  Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats
t1 t2) (Q-Eats u1 u2))) AND
                 HPair (Var n) (HPair (Var sn) (Var sn')) IN
                  Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats
t1 t2) (Q-Eats u1 u2))) AND
                    Var sl EQ Eats (Var sm) (Var sn) AND Var sl' EQ Q-Eats
(Var sm') (Var sn'))))))))))
  — verifying the sequence buildup
   apply (rule cut-same [where A=HaddP (SUCC k1) (SUCC k2) (SUCC
(SUCC (Var k')))])
  apply (blast intro: HaddP-SUCC1 [THEN cut1] HaddP-SUCC2 [THEN cut1])
  apply (rule All-I Imp-I)+
  apply (rule HaddP-Mem-cases [where i=j])
  using assms atoms atoms2 apply simp-all
  apply (rule AssumeH)
  apply (blast intro: OrdP-SUCC-I)
  — ... the sequence buildup via s1
  apply (simp add: SeqQuoteP.simps [of l s1 - sl sl' m n sm sm' sn sn'])
  apply (rule AssumeH Ex-EH Conj-EH)+
  apply (rule All2-E [THEN rotate2])
  apply (simp | rule AssumeH Ex-EH Conj-EH)+
  apply (rule Ex-I [where x=Var sl], simp)
  apply (rule Ex-I [where x=Var sl'], simp)
  apply (rule Conj-I)
  apply (rule Mem-Eats-I1)
  apply (metis SeqAppendP-Mem1 rotate3 thin2 thin4)
  apply (rule AssumeH Disj-IE1H Ex-EH Conj-EH)+
  apply (rule Ex-I [where x=Var m], simp)
  apply (rule Ex-I [where x=Var n], simp)
  apply (rule Ex-I [where x=Var sm], simp)
  apply (rule Ex-I [where x=Var sm'], simp)
  apply (rule Ex-I [where x=Var sn], simp)
  apply (rule Ex-I [where x=Var sn'], simp-all)
  apply (rule Conj-I, rule AssumeH)+
 apply (blast intro: OrdP-Trans [OF OrdP-SUCC-I] Mem-Eats-I1 [OF SeqAppendP-Mem1
```

[*THEN cut3*]] *Hyp*)
— ... the sequence buildup via s2
apply (*simp add*: *SeqQuoteP.simps* [*of l s2 - sl sl′ m n sm sm′ sn sn′*])
apply (*rule AssumeH Ex-EH Conj-EH*)+
apply (*rule All2-E* [*THEN rotate2*])
apply (*simp* | *rule AssumeH Ex-EH Conj-EH*)+
apply (*rule Ex-I* [**where** *x*=*Var sl*], *simp*)
apply (*rule Ex-I* [**where** *x*=*Var sl′*], *simp*)
apply (*rule cut-same* [**where** *A*=*OrdP* (*Var j*)])
apply (*metis HaddP-imp-OrdP rotate2 thin2*)
apply (*rule Conj-I*)
apply (*blast intro*: *Mem-Eats-I1 SeqAppendP-Mem2* [*THEN cut4*] *del*: *Disj-EH*)
apply (*rule AssumeH Disj-IE1H Ex-EH Conj-EH*)+
apply (*rule cut-same* [*OF exists-HaddP* [**where** *j*=*km* **and** *x*=*SUCC k1* **and** *y*=*Var m*]])
apply (*blast intro*: *Ord-IN-Ord*, *simp*)
apply (*rule cut-same* [*OF exists-HaddP* [**where** *j*=*kn* **and** *x*=*SUCC k1* **and** *y*=*Var n*]])
apply (*metis AssumeH*(*6*) *Ord-IN-Ord0 rotate8*, *simp*)
apply (*rule AssumeH Ex-EH Conj-EH* | *simp*)+
apply (*rule Ex-I* [**where** *x*=*Var km*], *simp*)
apply (*rule Ex-I* [**where** *x*=*Var kn*], *simp*)
apply (*rule Ex-I* [**where** *x*=*Var sm*], *simp*)
apply (*rule Ex-I* [**where** *x*=*Var sm′*], *simp*)
apply (*rule Ex-I* [**where** *x*=*Var sn*], *simp*)
apply (*rule Ex-I* [**where** *x*=*Var sn′*], *simp-all*)
apply (*rule Conj-I* [*OF - Conj-I*])
apply (*blast intro*: *Hyp OrdP-SUCC-I HaddP-Mem-cancel-left* [*THEN Iff-MP2-same*])
apply (*blast intro*: *Hyp OrdP-SUCC-I HaddP-Mem-cancel-left* [*THEN Iff-MP2-same*])
apply (*blast intro*: *Hyp Mem-Eats-I1 SeqAppendP-Mem2* [*THEN cut4*] *OrdP-Trans HaddP-imp-OrdP* [*THEN cut1*])
done
qed
qed

lemma *QuoteP-Eats*: {*QuoteP t1 u1*, *QuoteP t2 u2*} ⊢ *QuoteP* (*Eats t1 t2*) (*Q-Eats u1 u2*)
proof −
obtain *k1*::*name* **and** *s1*::*name* **and** *k2*::*name* **and** *s2*::*name* **and** *k*::*name* **and** *s*::*name*
where *atom s1* ♯ (*t1*,*u1*,*t2*,*u2*) *atom k1* ♯ (*t1*,*u1*,*t2*,*u2*,*s1*)
atom s2 ♯ (*t1*,*u1*,*t2*,*u2*,*k1*,*s1*) *atom k2* ♯ (*t1*,*u1*,*t2*,*u2*,*s2*,*k1*,*s1*)
atom s ♯ (*t1*,*u1*,*t2*,*u2*,*k2*,*s2*,*k1*,*s1*) *atom k* ♯ (*t1*,*u1*,*t2*,*u2*,*s*,*k2*,*s2*,*k1*,*s1*)
by (*metis obtain-fresh*)
thus *?thesis*
by (*auto simp*: *QuoteP.simps* [*of s* - (*Q-Eats u1 u2*) *k*]
QuoteP.simps [*of s1 t1 u1 k1*] *QuoteP.simps* [*of s2 t2 u2 k2*]

intro!: *SeqQuoteP-Eats* [*THEN cut2*])
qed

lemma *exists-QuoteP*:
assumes *j*: *atom j* ♯ *x* **shows** {} ⊢ *Ex j* (*QuoteP x* (*Var j*))
proof −
obtain *i*::*name* **and** *j'*::*name* **and** *k*::*name*
where *atoms*: *atom i* ♯ (*j*,*x*) *atom j'* ♯ (*i*,*j*,*x*) *atom* (*k*::*name*) ♯ (*i*,*j*,*j'*,*x*)
by (*metis obtain-fresh*)
have {} ⊢ *Ex j* (*QuoteP* (*Var i*) (*Var j*)) (**is** {} ⊢ *?scheme*)
proof (*rule Ind* [*of k*])
show *atom k* ♯ (*i*, *?scheme*) **using** *atoms*
by *simp*
next
show {} ⊢ *?scheme*(*i*::=*Zero*) **using** *j atoms*
by (*auto intro*: *Ex-I* [**where** *x*=*Zero*] *simp add*: *QuoteP-Zero*)
next
show {} ⊢ *All i* (*All k* (*?scheme IMP ?scheme*(*i*::=*Var k*) *IMP ?scheme*(*i*::=*Eats* (*Var i*) (*Var k*))))
apply (*rule All-I Imp-I*)+
using *atoms assms*
apply *simp-all*
apply (*rule Ex-E*)
apply (*rule Ex-E-with-renaming* [**where** *i'*=*j'*, *THEN rotate2*], *auto*)
apply (*rule Ex-I* [**where** *x*= *Q-Eats* (*Var j'*) (*Var j*)], *auto intro*: *QuoteP-Eats*)
done
qed
hence {} ⊢ (*Ex j* (*QuoteP* (*Var i*) (*Var j*))) (*i*::= *x*)
by (*rule Subst*) *auto*
thus *?thesis*
using *atoms j* **by** *auto*
qed

lemma *QuoteP-imp-ConstP*: { *QuoteP x y* } ⊢ *ConstP y*
proof −
obtain *j*::*name* **and** *j'*::*name* **and** *l*::*name* **and** *s*::*name* **and** *k*::*name*
and *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name* **and** *sm'*::*name* **and** *sn'*::*name*
where *atoms*: *atom j* ♯ (*x*,*y*,*s*,*k*,*j'*,*l*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom j' ♯ (*x*,*y*,*s*,*k*,*l*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom l ♯ (*s*,*k*,*m*,*n*,*sm*,*sm'*,*sn*,*sn'*)
atom m ♯ (*s*,*k*,*n*,*sm*,*sm'*,*sn*,*sn'*) *atom n* ♯ (*s*,*k*,*sm*,*sm'*,*sn*,*sn'*)
atom sm ♯ (*s*,*k*,*sm'*,*sn*,*sn'*) *atom sm'* ♯ (*s*,*k*,*sn*,*sn'*)
atom sn ♯ (*s*,*k*,*sn'*) *atom sn'* ♯ (*s*,*k*) *atom s* ♯ (*k*,*x*,*y*) *atom k* ♯ (*x*,*y*)
by (*metis obtain-fresh*)
have {*OrdP* (*Var k*)}
⊢ *All j* (*All j'* (*SeqQuoteP* (*Var j*) (*Var j'*) (*Var s*) (*Var k*) *IMP ConstP* (*Var j'*)))
(**is** - ⊢ *?scheme*)

proof (*rule OrdIndH* [**where** *j*=*l*])
show *atom l* ♯ (*k*, *?scheme*) **using** *atoms assms*
by *simp*
next
show {} ⊢ *All k* (*OrdP* (*Var k*) *IMP* (*All2 l* (*Var k*) (*?scheme*(*k*::= *Var l*)) *IMP ?scheme*))
apply (*rule All-I Imp-I*)+
using *atoms assms*
apply (*simp-all add*: *fresh-at-base fresh-finite-set-at-base*)
— freshness finally proved!
apply (*rule cut-same*)
apply (*rule cut1* [*OF SeqQuoteP-lemma* [*of m Var j Var j′ Var s Var k n sm sm′ sn sn′*]], *simp-all*, *blast*)
apply (*rule Imp-I Disj-EH Conj-EH*)+
— case 1, Var j EQ Zero
apply (*rule thin1*)
apply (*rule Var-Eq-subst-Iff* [*THEN Iff-MP-same*], *simp*)
apply (*metis thin0 ConstP-Zero*)
— case 2, *Var j EQ Eats* (*Var sm*) (*Var sn*)
apply (*rule Imp-I Conj-EH Ex-EH*)+
apply *simp-all*
apply (*rule Var-Eq-subst-Iff* [*THEN Iff-MP-same*, *THEN rotate2*], *simp*)
apply (*rule ConstP-Eats* [*THEN cut2*])
— Operand 1. IH for sm
apply (*rule All2-E* [**where** *x*=*Var m*, *THEN rotate8*], *auto*)
apply (*rule All-E* [**where** *x*=*Var sm*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm′*], *auto*)
— Operand 2. IH for sm
apply (*rule All2-E* [**where** *x*=*Var n*, *THEN rotate8*], *auto*)
apply (*rule All-E* [**where** *x*=*Var sn*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn′*], *auto*)
done
qed
hence {*OrdP*(*Var k*)}
⊢ (*All j′* (*SeqQuoteP* (*Var j*) (*Var j′*) (*Var s*) (*Var k*) *IMP ConstP* (*Var j′*))) (*j*::=*x*)
by (*metis All-D*)
hence {*OrdP*(*Var k*)} ⊢ *All j′* (*SeqQuoteP x* (*Var j′*) (*Var s*) (*Var k*) *IMP ConstP* (*Var j′*))
using *atoms* **by** *simp*
hence {*OrdP*(*Var k*)} ⊢ (*SeqQuoteP x* (*Var j′*) (*Var s*) (*Var k*) *IMP ConstP* (*Var j′*)) (*j′*::=*y*)
by (*metis All-D*)
hence {*OrdP*(*Var k*)} ⊢ *SeqQuoteP x y* (*Var s*) (*Var k*) *IMP ConstP y*
using *atoms* **by** *simp*
hence { *SeqQuoteP x y* (*Var s*) (*Var k*) } ⊢ *ConstP y*
by (*metis Imp-cut SeqQuoteP-imp-OrdP anti-deduction*)
thus { *QuoteP x y* } ⊢ *ConstP y* **using** *atoms*
by (*auto simp*: *QuoteP.simps* [*of s - - k*])

qed

lemma *SeqQuoteP-imp-QuoteP*: $\{SeqQuoteP\ t\ u\ s\ k\} \vdash QuoteP\ t\ u$
proof −
  obtain s'::*name* **and** k'::*name* **where** *atom* $s' \sharp (k',t,u,s,k)$ *atom* $k' \sharp (t,u,s,k)$
    by (*metis obtain-fresh*)
  thus *?thesis*
    apply (*simp add*: *QuoteP.simps* [*of* s' - - k'])
    apply (*rule Ex-I* [**where** $x = s$], *simp*)
    apply (*rule Ex-I* [**where** $x = k$], *auto*)
    done
qed

lemmas *QuoteP-I* = *SeqQuoteP-imp-QuoteP* [*THEN cut1*]

11.3 The Operator *quote-all*

11.3.1 Definition and basic properties

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

lemma *quote-all-empty* [*simp*]: *quote-all p* {} = {}
  by (*simp add*: *quote-all-def*)

lemma *quote-all-insert* [*simp*]:
  quote-all p (*insert i V*) = *insert* (*QuoteP* (*Var i*) (*Var* $(p \cdot i)$)) (*quote-all p V*)
  by (*auto simp*: *quote-all-def*)

lemma *finite-quote-all* [*simp*]: *finite V* $\Longrightarrow$ *finite* (*quote-all p V*)
  by (*induct rule*: *finite-induct*) *auto*

lemma *fresh-quote-all* [*simp*]: *finite V* $\Longrightarrow i \sharp$ *quote-all p V* $\longleftrightarrow i \sharp V \wedge i \sharp p \cdot V$
  by (*induct rule*: *finite-induct*) (*auto simp*: *fresh-finite-insert*)

lemma *fresh-quote-all-mem*: $[\![A \in$ *quote-all p V*; *finite V*; $i \sharp V$; $i \sharp p \cdot V]\!] \Longrightarrow$
$i \sharp A$
 by (*metis Set.set-insert finite-insert finite-quote-all fresh-finite-insert fresh-quote-all*)

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 −
  { fix *W*
    assume *w*: $W \subseteq V$
    have *finite W*
      by (*metis* ‹*finite V*› *finite-subset w*)
    hence *quote-all* ((*atom* $i \rightleftharpoons$ *atom* i') + *p*) *W* = *quote-all p W* **using** *w*
        apply *induction* **using** *assms*

apply (*auto simp*: *fresh-Pair perm-commute*)
apply (*metis fresh-finite-set-at-base swap-at-base-simps*(*3*))+
done}
thus *?thesis*
by (*metis order-refl*)
qed

11.3.2 Transferring theorems to the level of derivability

context *quote-perm*
begin

lemma *QuoteP-imp-ConstP-F-hyps*:
assumes $Us \subseteq Vs$ $\{ConstP\ (F\ i) \mid i.\ i \in Us\} \vdash A$ **shows** *quote-all p Us* $\vdash A$
proof −
show *?thesis* **using** *finite-V* [*OF* ‹$Us \subseteq Vs$›] *assms*
proof (*induction arbitrary*: *A rule*: *finite-induct*)
case *empty* **thus** *?case* **by** *simp*
next
case (*insert v Us*) **thus** *?case*
by (*auto simp*: *Collect-disj-Un*)
(*metis* (*lifting*) *anti-deduction Imp-cut* [*OF* - *QuoteP-imp-ConstP*] *Disj-I2*
F-unfold)
qed
qed

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 −
have $\{\} \vdash$ *PfP* $\lceil\beta\rceil$
by (*metis* β *proved-iff-proved-PfP*)
hence $\{ConstP\ (F\ i) \mid i.\ i \in V\} \vdash$ *PfP* (*ssubst* $\lfloor\beta\rfloor$ *V V F*)
by (*simp add*: *PfP-implies-PfP-ssubst V s*)
thus *?thesis*
by (*rule QuoteP-imp-ConstP-F-hyps* [*OF V*])
qed

Lemma 8.4

corollary *quote-all-MonPon-PfP-ssubst*:
assumes A: $\{\} \vdash \alpha$ *IMP* β
and V: $V \subseteq Vs$
and s: $supp\ \alpha \subseteq atom$ ‘ Vs $supp\ \beta \subseteq atom$ ‘ Vs
shows *quote-all p V* $\vdash$ *PfP* (*ssubst* $\lfloor\alpha\rfloor$ *V V F*) *IMP PfP* (*ssubst* $\lfloor\beta\rfloor$ *V V F*)
using *quote-all-PfP-ssubst* [*OF A V*] *s*
by (*auto simp*: *V vquot-fm-def intro*: *PfP-implies-ModPon-PfP thin1*)

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*)
using *quote-all-PfP-ssubst* [*OF A V*] *s*
 by (*force simp*: *V vquot-fm-def intro*: *PfP-implies-ModPon-PfP* [*OF PfP-implies-ModPon-PfP*] *thin1*)

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 −
  have {} ⊢ *α IMP* (*α OR β*)
   by (*blast intro*: *Disj-I1*)
  hence *quote-all p V* ⊢ *PfP* (*ssubst* ⌊*α*⌋*V V F*) *IMP PfP* (*ssubst* ⌊*α OR β*⌋*V V F*)
   using *assms* **by** (*auto simp*: *quote-all-MonPon-PfP-ssubst*)
  thus *?thesis*
   by (*metis MP-same prems thin*)
qed

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 −
  have {} ⊢ *β IMP* (*α OR β*)
   by (*blast intro*: *Disj-I2*)
  hence *quote-all p V* ⊢ *PfP* (*ssubst* ⌊*β*⌋*V V F*) *IMP PfP* (*ssubst* ⌊*α OR β*⌋*V V F*)
   using *assms* **by** (*auto simp*: *quote-all-MonPon-PfP-ssubst*)
  thus *?thesis*
   by (*metis MP-same prems thin*)
qed

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 −
  have {} ⊢ *α IMP β IMP* (*α AND β*)
   by *blast*
  hence *quote-all p V*
     ⊢ *PfP* (*ssubst* ⌊*α*⌋*V V F*) *IMP PfP* (*ssubst* ⌊*β*⌋*V V F*) *IMP PfP* (*ssubst*

$\lfloor\alpha\ AND\ \beta\rfloor\ V\ V\ F)$
using *assms* **by** (*auto simp*: *quote-all-MonPon2-PfP-ssubst*)
thus *?thesis*
by (*metis MP-same prems thin*)
qed

lemma *quote-all-Contra-PfP-ssubst*:
assumes $V \subseteq Vs$ *supp* $\alpha \subseteq$ *atom* ' *Vs*
shows *quote-all p V*
$\vdash PfP\ (ssubst\ \lfloor\alpha\rfloor\ V\ V\ F)\ IMP\ PfP\ (ssubst\ \lfloor Neg\ \alpha\rfloor\ V\ V\ F)\ IMP\ PfP\ (ssubst\ \lfloor Fls\rfloor\ V\ V\ F)$
proof −
have $\{\} \vdash \alpha\ IMP\ Neg\ \alpha\ IMP\ Fls$
by *blast*
thus *?thesis*
using *assms* **by** (*auto simp*: *quote-all-MonPon2-PfP-ssubst supp-conv-fresh*)
qed

lemma *fresh-ssubst-dbtm*: $[\![atom\ i\ \sharp\ p \bullet V;\ V \subseteq Vs]\!] \Longrightarrow atom\ i\ \sharp\ ssubst\ (vquot\text{-}dbtm\ V\ t)\ V\ F$
by (*induct t rule*: *dbtm.induct*) (*auto simp*: *F-unfold fresh-image permute-set-eq-image*)

lemma *fresh-ssubst-dbfm*: $[\![atom\ i\ \sharp\ p \bullet V;\ V \subseteq Vs]\!] \Longrightarrow atom\ i\ \sharp\ ssubst\ (vquot\text{-}dbfm\ V\ A)\ V\ F$
by (*nominal-induct A rule*: *dbfm.strong-induct*) (*auto simp*: *fresh-ssubst-dbtm*)

lemma *fresh-ssubst-fm*:
fixes *A*::*fm* **shows** $[\![atom\ i\ \sharp\ p \bullet V;\ V \subseteq Vs]\!] \Longrightarrow atom\ i\ \sharp\ ssubst\ (\lfloor A\rfloor\ V)\ V\ F$
by (*simp add*: *fresh-ssubst-dbfm vquot-fm-def*)

end

11.4 Star Property. Equality and Membership: Lemmas 9.3 and 9.4

lemma *SeqQuoteP-Mem-imp-QMem-and-Subset*:
assumes $atom\ i\ \sharp\ (j,j',i',si,ki,sj,kj)$ $atom\ i'\ \sharp\ (j,j',si,ki,sj,kj)$
$atom\ j\ \sharp\ (j',si,ki,sj,kj)$ $atom\ j'\ \sharp\ (si,ki,sj,kj)$
$atom\ si\ \sharp\ (ki,sj,kj)$ $atom\ sj\ \sharp\ (ki,kj)$
shows $\{SeqQuoteP\ (Var\ i)\ (Var\ i')\ (Var\ si)\ ki,\ SeqQuoteP\ (Var\ j)\ (Var\ j')\ (Var\ sj)\ kj\}$
$\vdash (Var\ i\ IN\ Var\ j\ IMP\ PfP\ (Q\text{-}Mem\ (Var\ i')\ (Var\ j')))\ AND$
$(Var\ i\ SUBS\ Var\ j\ IMP\ PfP\ (Q\text{-}Subset\ (Var\ i')\ (Var\ j')))$
proof −
obtain *k*::*name* **and** *l*::*name* **and** *li*::*name* **and** *lj*::*name*
and *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name* **and** *sm'*::*name* **and** *sn'*::*name*
where *atoms*: $atom\ lj\ \sharp\ (li,l,i,j,j',i',si,ki,sj,kj,i,i',k,m,n,sm,sm',sn,sn')$

```
            atom li ♯ (l,j,j',i,i',si,ki,sj,kj,i,i',k,m,n,sm,sm',sn,sn')
            atom l ♯ (j,j',i,i',si,ki,sj,kj,i,i',k,m,n,sm,sm',sn,sn')
            atom k ♯ (j,j',i,i',si,ki,sj,kj,m,n,sm,sm',sn,sn')
            atom m ♯ (j,j',i,i',si,ki,sj,kj,n,sm,sm',sn,sn')
            atom n ♯ (j,j',i,i',si,ki,sj,kj,sm,sm',sn,sn')
            atom sm ♯ (j,j',i,i',si,ki,sj,kj,sm',sn,sn')
            atom sm' ♯ (j,j',i,i',si,ki,sj,kj,sn,sn')
            atom sn ♯ (j,j',i,i',si,ki,sj,kj,sn')
            atom sn' ♯ (j,j',i,i',si,ki,sj,kj)
   by (metis obtain-fresh)
  have {OrdP(Var k)}
     ⊢ All i (All i' (All si (All li (All j (All j' (All sj (All lj
        (SeqQuoteP (Var i) (Var i') (Var si) (Var li) IMP
         SeqQuoteP (Var j) (Var j') (Var sj) (Var lj) IMP
         HaddP (Var li) (Var lj) (Var k) IMP
          ( (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'))))))))))))
      (is - ⊢ ?scheme)
   proof (rule OrdIndH [where j=l])
     show atom l ♯ (k, ?scheme) using atoms
       by simp
   next
     def V ≡ {i,j,sm,sn}
     and p ≡ (atom i ⇌ atom i') + (atom j ⇌ atom j') +
             (atom sm ⇌ atom sm') + (atom sn ⇌ atom sn')
     def F ≡ make-F V p
     interpret qp: quote-perm p V F
       proof unfold-locales
         show finite V by (simp add: V-def)
         show atom ‘ (p · V) ♯* V
           using atoms assms
        by (auto simp: p-def V-def F-def make-F-def fresh-star-def fresh-finite-insert)
         show −p = p  using assms atoms
        by (simp add: p-def add.assoc perm-self-inverseI fresh-swap fresh-plus-perm)
         show F ≡ make-F V p
           by (rule F-def)
       qed
     have V-mem: i ∈ V j ∈ V sm ∈ V sn ∈ V
       by (auto simp: V-def)  — Part of (2) from page 32
     have Mem1: {} ⊢ (Var i IN Var sm) IMP (Var i IN Eats (Var sm) (Var sn))
       by (blast intro: Mem-Eats-I1)
     have Q-Mem1: quote-all p V
                  ⊢ PfP (Q-Mem (Var i') (Var sm')) IMP
                    PfP (Q-Mem (Var i') (Q-Eats (Var sm') (Var sn')))
        using qp.quote-all-MonPon-PfP-ssubst [OF Mem1 subset-refl] assms atoms
V-mem
        by (simp add: vquot-fm-def ‹finite V›) (simp add: qp.F-unfold p-def)
     have Mem2: {} ⊢ (Var i EQ Var sn) IMP (Var i IN Eats (Var sm) (Var sn))
       by (blast intro: Mem-Eats-I2)
```

have *Q-Mem2*: *quote-all p V*
⊢ *PfP* (*Q-Eq* (*Var i′*) (*Var sn′*)) *IMP*
PfP (*Q-Mem* (*Var i′*) (*Q-Eats* (*Var sm′*) (*Var sn′*)))
using *qp.quote-all-MonPon-PfP-ssubst* [*OF Mem2 subset-refl*] *assms atoms V-mem*
by (*simp add*: *vquot-fm-def* ‹*finite V*›) (*simp add*: *qp.F-unfold p-def*)
have *Subs1*: {} ⊢ *Zero SUBS Var j*
by *blast*
have *Q-Subs1*: {*QuoteP* (*Var j*) (*Var j′*)} ⊢ *PfP* (*Q-Subset Zero* (*Var j′*))
using *qp.quote-all-PfP-ssubst* [*OF Subs1*, *of* {*j*}] *assms atoms*
by (*simp add*: *qp.ssubst-Subset vquot-tm-def supp-conv-fresh fresh-at-base del*: *qp.ssubst-single*)
(*simp add*: *qp.F-unfold p-def V-def*)
have *Subs2*: {} ⊢ *Var sm SUBS Var j IMP Var sn IN Var j IMP Eats* (*Var sm*) (*Var sn*) *SUBS Var j*
by *blast*
have *Q-Subs2*: *quote-all p V*
⊢ *PfP* (*Q-Subset* (*Var sm′*) (*Var j′*)) *IMP*
PfP (*Q-Mem* (*Var sn′*) (*Var j′*)) *IMP*
PfP (*Q-Subset* (*Q-Eats* (*Var sm′*) (*Var sn′*)) (*Var j′*))
using *qp.quote-all-MonPon2-PfP-ssubst* [*OF Subs2 subset-refl*] *assms atoms V-mem*
by (*simp add*: *qp.ssubst-Subset vquot-tm-def supp-conv-fresh subset-eq fresh-at-base*)
(*simp add*: *vquot-fm-def qp.F-unfold p-def V-def*)
have *Ext*: {} ⊢ *Var i SUBS Var sn IMP Var sn SUBS Var i IMP Var i EQ Var sn*
by (*blast intro*: *Equality-I*)
have *Q-Ext*: {*QuoteP* (*Var i*) (*Var i′*), *QuoteP* (*Var sn*) (*Var sn′*)}
⊢ *PfP* (*Q-Subset* (*Var i′*) (*Var sn′*)) *IMP*
PfP (*Q-Subset* (*Var sn′*) (*Var i′*)) *IMP*
PfP (*Q-Eq* (*Var i′*) (*Var sn′*))
using *qp.quote-all-MonPon2-PfP-ssubst* [*OF Ext*, *of* {*i*,*sn*}] *assms atoms*
by (*simp add*: *qp.ssubst-Subset vquot-tm-def supp-conv-fresh subset-eq fresh-at-base*
del: *qp.ssubst-single*)
(*simp add*: *vquot-fm-def qp.F-unfold p-def V-def*)
show {} ⊢ *All k* (*OrdP* (*Var k*) *IMP* (*All2 l* (*Var k*) (*?scheme*(*k*::= *Var l*)) *IMP ?scheme*))
apply (*rule All-I Imp-I*)+
using *atoms assms*
apply *simp-all*
apply (*rule cut-same* [**where** *A* = *QuoteP* (*Var i*) (*Var i′*)])
apply (*blast intro*: *QuoteP-I*)
apply (*rule cut-same* [**where** *A* = *QuoteP* (*Var j*) (*Var j′*)])
apply (*blast intro*: *QuoteP-I*)
apply (*rule rotate6*)
apply (*rule Conj-I*)
— *Var i IN Var j IMP PfP* (*Q-Mem* (*Var i′*) (*Var j′*))

apply (*rule cut-same*)
apply (*rule cut1* [*OF SeqQuoteP-lemma* [*of m Var j Var j′ Var sj Var lj n sm sm′ sn sn′*]], *simp-all*, *blast*)
apply (*rule Imp-I Disj-EH Conj-EH*)+
— case 1, *Var j EQ Zero*
apply (*rule cut-same* [**where** *A* = *Var i IN Zero*])
apply (*blast intro*: *Mem-cong* [*THEN Iff-MP-same*], *blast*)
— case 2, *Var j EQ Eats* (*Var sm*) (*Var sn*)
apply (*rule Imp-I Conj-EH Ex-EH*)+
apply *simp-all*
apply (*rule Var-Eq-subst-Iff* [*THEN rotate2*, *THEN Iff-MP-same*], *simp*)
apply (*rule cut-same* [**where** *A* = *QuoteP* (*Var sm*) (*Var sm′*)])
apply (*blast intro*: *QuoteP-I*)
apply (*rule cut-same* [**where** *A* = *QuoteP* (*Var sn*) (*Var sn′*)])
apply (*blast intro*: *QuoteP-I*)
apply (*rule cut-same* [**where** *A* = *Var i IN Eats* (*Var sm*) (*Var sn*)])
apply (*rule Mem-cong* [*OF Refl*, *THEN Iff-MP-same*])
apply (*rule AssumeH Mem-Eats-E*)+
— Eats case 1. IH for sm
apply (*rule cut-same* [**where** *A* = *OrdP* (*Var m*)])
apply (*blast intro*: *Hyp Ord-IN-Ord SeqQuoteP-imp-OrdP* [*THEN cut1*])
apply (*rule cut-same* [*OF exists-HaddP* [**where** *j*=*l* **and** *x*=*Var li* **and** *y*=*Var m*]])
apply *auto*
apply (*rule All2-E* [**where** *x*=*Var l*, *THEN rotate13*], *simp-all*)
apply (*blast intro*: *Hyp HaddP-Mem-cancel-left* [*THEN Iff-MP2-same*] *SeqQuoteP-imp-OrdP* [*THEN cut1*])
apply (*rule All-E* [**where** *x*=*Var i*], *simp*)
apply (*rule All-E* [**where** *x*=*Var i′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var si*], *simp*)
apply (*rule All-E* [**where** *x*=*Var li*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm′*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sj*], *simp*)
apply (*rule All-E* [**where** *x*=*Var m*], *simp*)
apply (*force intro*: *MP-thin* [*OF Q-Mem1*] *simp add*: *V-def p-def*)
— Eats case 2
apply (*rule rotate13*)
apply (*rule cut-same* [**where** *A* = *OrdP* (*Var n*)])
apply (*blast intro*: *Hyp Ord-IN-Ord SeqQuoteP-imp-OrdP* [*THEN cut1*])
apply (*rule cut-same* [*OF exists-HaddP* [**where** *j*=*l* **and** *x*=*Var li* **and** *y*=*Var n*]])
apply *auto*
apply (*rule MP-same*)
apply (*rule Q-Mem2* [*THEN thin*])
apply (*simp add*: *V-def p-def*)
apply (*rule MP-same*)
apply (*rule MP-same*)
apply (*rule Q-Ext* [*THEN thin*])

apply (*simp add*: *V-def p-def*)
— *PfP* (*Q-Subset* (*Var i'*) (*Var sn'*))
apply (*rule All2-E* [**where** *x*=*Var l*, *THEN rotate14*], *simp-all*)
apply (*blast intro*: *Hyp HaddP-Mem-cancel-left* [*THEN Iff-MP2-same*] *SeqQuoteP-imp-OrdP* [*THEN cut1*])
apply (*rule All-E* [**where** *x*=*Var i*], *simp*)
apply (*rule All-E* [**where** *x*=*Var i'*], *simp*)
apply (*rule All-E* [**where** *x*=*Var si*], *simp*)
apply (*rule All-E* [**where** *x*=*Var li*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn'*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sj*], *simp*)
apply (*rule All-E* [**where** *x*=*Var n*], *simp*)
apply (*rule Imp-E*, *blast intro*: *Hyp*)+
apply (*rule Conj-E*)
apply (*rule thin1*)
apply (*blast intro*!: *Imp-E EQ-imp-SUBS* [*THEN cut1*])
— *PfP* (*Q-Subset* (*Var sn'*) (*Var i'*))
apply (*rule All2-E* [**where** *x*=*Var l*, *THEN rotate14*], *simp-all*)
apply (*blast intro*: *Hyp HaddP-Mem-cancel-left* [*THEN Iff-MP2-same*] *SeqQuoteP-imp-OrdP* [*THEN cut1*])
apply (*rule All-E* [**where** *x*=*Var sn*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn'*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sj*], *simp*)
apply (*rule All-E* [**where** *x*=*Var n*], *simp*)
apply (*rule All-E* [**where** *x*=*Var i*], *simp*)
apply (*rule All-E* [**where** *x*=*Var i'*], *simp*)
apply (*rule All-E* [**where** *x*=*Var si*], *simp*)
apply (*rule All-E* [**where** *x*=*Var li*], *simp*)
apply (*rule Imp-E*, *blast intro*: *Hyp*)+
apply (*rule Imp-E*)
apply (*blast intro*: *Hyp HaddP-commute* [*THEN cut2*] *SeqQuoteP-imp-OrdP* [*THEN cut1*])
apply (*rule Conj-E*)
apply (*rule thin1*)
apply (*blast intro*!: *Imp-E EQ-imp-SUBS2* [*THEN cut1*])
— *Var i SUBS Var j IMP PfP* (*Q-Subset* (*Var i'*) (*Var j'*))
apply (*rule cut-same*)
apply (*rule cut1* [*OF SeqQuoteP-lemma* [*of m Var i Var i' Var si Var li n sm sm' sn sn'*]], *simp-all*, *blast*)
apply (*rule Imp-I Disj-EH Conj-EH*)+
— case 1, Var i EQ Zero
apply (*rule cut-same* [**where** *A* = *PfP* (*Q-Subset Zero* (*Var j'*))])
apply (*blast intro*: *Q-Subs1* [*THEN cut1*] *SeqQuoteP-imp-QuoteP* [*THEN cut1*])
apply (*force intro*: *Var-Eq-subst-Iff* [*THEN Iff-MP-same*, *THEN rotate3*])
— case 2, *Var i EQ Eats* (*Var sm*) (*Var sn*)
apply (*rule Conj-EH Ex-EH*)+
apply *simp-all*

apply (*rule cut-same* [**where** *A* = *OrdP* (*Var lj*)])
apply (*blast intro*: *Hyp SeqQuoteP-imp-OrdP* [*THEN cut1*])
apply (*rule Var-Eq-subst-Iff* [*THEN Iff-MP-same*, *THEN rotate3*], *simp*)
apply (*rule cut-same* [**where** *A* = *QuoteP* (*Var sm*) (*Var sm'*)])
apply (*blast intro*: *QuoteP-I*)
apply (*rule cut-same* [**where** *A* = *QuoteP* (*Var sn*) (*Var sn'*)])
apply (*blast intro*: *QuoteP-I*)
apply (*rule cut-same* [**where** *A* = *Eats* (*Var sm*) (*Var sn*) *SUBS Var j*])
apply (*rule Subset-cong* [*OF* - *Refl*, *THEN Iff-MP-same*])
apply (*rule AssumeH Mem-Eats-E*)+
— Eats case split
apply (*rule Eats-Subset-E*)
apply (*rule rotate15*)
apply (*rule MP-same* [*THEN MP-same*])
apply (*rule Q-Subs2* [*THEN thin*])
apply (*simp add*: *V-def p-def*)
— Eats case 1: *PfP* (*Q-Subset* (*Var sm'*) (*Var j'*))
apply (*rule cut-same* [*OF exists-HaddP* [**where** *j*=*l* **and** *x*=*Var m* **and** *y*=*Var lj*]])
apply (*rule AssumeH Ex-EH Conj-EH* | *simp*)+
— IH for sm
apply (*rule All2-E* [**where** *x*=*Var l*, *THEN rotate15*], *simp-all*)
apply (*blast intro*: *Hyp HaddP-Mem-cancel-right-Mem SeqQuoteP-imp-OrdP* [*THEN cut1*])
apply (*rule All-E* [**where** *x*=*Var sm*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm'*], *simp*)
apply (*rule All-E* [**where** *x*=*Var si*], *simp*)
apply (*rule All-E* [**where** *x*=*Var m*], *simp*)
apply (*rule All-E* [**where** *x*=*Var j*], *simp*)
apply (*rule All-E* [**where** *x*=*Var j'*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sj*], *simp*)
apply (*rule All-E* [**where** *x*=*Var lj*], *simp*)
apply (*blast intro*: *thin1 Imp-E*)
— Eats case 2: *PfP* (*Q-Mem* (*Var sn'*) (*Var j'*))
apply (*rule cut-same* [*OF exists-HaddP* [**where** *j*=*l* **and** *x*=*Var n* **and** *y*=*Var lj*]])
apply (*rule AssumeH Ex-EH Conj-EH* | *simp*)+
— IH for sn
apply (*rule All2-E* [**where** *x*=*Var l*, *THEN rotate15*], *simp-all*)
apply (*blast intro*: *Hyp HaddP-Mem-cancel-right-Mem SeqQuoteP-imp-OrdP* [*THEN cut1*])
apply (*rule All-E* [**where** *x*=*Var sn*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sn'*], *simp*)
apply (*rule All-E* [**where** *x*=*Var si*], *simp*)
apply (*rule All-E* [**where** *x*=*Var n*], *simp*)
apply (*rule All-E* [**where** *x*=*Var j*], *simp*)
apply (*rule All-E* [**where** *x*=*Var j'*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sj*], *simp*)
apply (*rule All-E* [**where** *x*=*Var lj*], *simp*)

apply (*blast intro*: *Hyp Imp-E*)
done
qed
hence *p1*: {*OrdP*(*Var k*)}
⊢ (*All i′* (*All si* (*All li*
(*All j* (*All j′* (*All sj* (*All lj*
(*SeqQuoteP* (*Var i*) (*Var i′*) (*Var si*) (*Var li*) *IMP*
SeqQuoteP (*Var j*) (*Var j′*) (*Var sj*) (*Var lj*) *IMP*
HaddP (*Var li*) (*Var lj*) (*Var k*) *IMP*
(*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′*))))))))))))
(*i*::= *Var i*)
by (*metis All-D*)
have *p2*: {*OrdP*(*Var k*)}
⊢ (*All si* (*All li*
(*All j* (*All j′* (*All sj* (*All lj*
(*SeqQuoteP* (*Var i*) (*Var i′*) (*Var si*) (*Var li*) *IMP*
SeqQuoteP (*Var j*) (*Var j′*) (*Var sj*) (*Var lj*) *IMP*
HaddP (*Var li*) (*Var lj*) (*Var k*) *IMP*
(*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′*))))))))))))(*i′*::= *Var i′*)
apply (*rule All-D*)
using *atoms p1* **by** *simp*
have *p3*: {*OrdP*(*Var k*)}
⊢ (*All li*
(*All j* (*All j′* (*All sj* (*All lj*
(*SeqQuoteP* (*Var i*) (*Var i′*) (*Var si*) (*Var li*) *IMP*
SeqQuoteP (*Var j*) (*Var j′*) (*Var sj*) (*Var lj*) *IMP*
HaddP (*Var li*) (*Var lj*) (*Var k*) *IMP*
(*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′*))))))))))
(*si*::= *Var si*)
apply (*rule All-D*)
using *atoms p2* **by** *simp*
have *p4*: {*OrdP*(*Var k*)}
⊢ (*All j* (*All j′* (*All sj* (*All lj*
(*SeqQuoteP* (*Var i*) (*Var i′*) (*Var si*) (*Var li*) *IMP*
SeqQuoteP (*Var j*) (*Var j′*) (*Var sj*) (*Var lj*) *IMP*
HaddP (*Var li*) (*Var lj*) (*Var k*) *IMP*
(*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′*))))))))
(*li*::= *ki*)
apply (*rule All-D*)
using *atoms p3* **by** *simp*
have *p5*: {*OrdP*(*Var k*)}
⊢ (*All j′* (*All sj* (*All lj*
(*SeqQuoteP* (*Var i*) (*Var i′*) (*Var si*) *ki IMP*
SeqQuoteP (*Var j*) (*Var j′*) (*Var sj*) (*Var lj*) *IMP*

HaddP ki (*Var lj*) (*Var k*) *IMP*
(*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'*))))))) (*j*::=
Var j)
apply (*rule All-D*)
using *atoms assms p4* **by** *simp*
have *p6*: {*OrdP*(*Var k*)}
⊢ (*All sj* (*All lj*
(*SeqQuoteP* (*Var i*) (*Var i'*) (*Var si*) *ki IMP*
SeqQuoteP (*Var j*) (*Var j'*) (*Var sj*) (*Var lj*) *IMP*
HaddP ki (*Var lj*) (*Var k*) *IMP*
(*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'*)))))) (*j'*::=
Var j')
apply (*rule All-D*)
using *atoms p5* **by** *simp*
have *p7*: {*OrdP*(*Var k*)}
⊢ (*All lj* (*SeqQuoteP* (*Var i*) (*Var i'*) (*Var si*) *ki IMP*
SeqQuoteP (*Var j*) (*Var j'*) (*Var sj*) (*Var lj*) *IMP*
HaddP ki (*Var lj*) (*Var k*) *IMP*
(*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'*)))))
(*sj*::= *Var sj*)
apply (*rule All-D*)
using *atoms p6* **by** *simp*
have *p8*: {*OrdP*(*Var k*)}
⊢ (*SeqQuoteP* (*Var i*) (*Var i'*) (*Var si*) *ki IMP*
SeqQuoteP (*Var j*) (*Var j'*) (*Var sj*) (*Var lj*) *IMP*
HaddP ki (*Var lj*) (*Var k*) *IMP*
(*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'*)))) (*lj*::= *kj*)
apply (*rule All-D*)
using *atoms p7* **by** *simp*
hence *p9*: {*OrdP*(*Var k*)}
⊢ *SeqQuoteP* (*Var i*) (*Var i'*) (*Var si*) *ki IMP*
SeqQuoteP (*Var j*) (*Var j'*) (*Var sj*) *kj IMP*
HaddP ki kj (*Var k*) *IMP*
(*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'*)))
using *assms atoms* **by** *simp*
have *p10*: { *HaddP ki kj* (*Var k*),
SeqQuoteP (*Var i*) (*Var i'*) (*Var si*) *ki*,
SeqQuoteP (*Var j*) (*Var j'*) (*Var sj*) *kj*, *OrdP* (*Var k*) }
⊢ (*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'*)))
apply (*rule MP-same* [*THEN MP-same* [*THEN MP-same*]])
apply (*rule p9* [*THEN thin*])
apply (*auto intro*: *MP-same*)
done

show *?thesis*
apply (*rule cut-same* [*OF exists-HaddP* [**where** *j*=*k* **and** *x*=*ki* **and** *y*=*kj*]])
apply (*metis SeqQuoteP-imp-OrdP thin1*)
prefer *2*
apply (*rule Ex-E*)
apply (*rule p10* [*THEN cut4*])
using *assms atoms*
apply (*auto intro*: *HaddP-OrdP SeqQuoteP-imp-OrdP* [*THEN cut1*])
done
qed

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 −
obtain *si*::*name* **and** *ki*::*name* **and** *sj*::*name* **and** *kj*::*name*
where *atoms*: *atom si* ♯ (*ki*,*sj*,*kj*,*i*,*j*,*j'*,*i'*) *atom ki* ♯ (*sj*,*kj*,*i*,*j*,*j'*,*i'*)
atom sj ♯ (*kj*,*i*,*j*,*j'*,*i'*) *atom kj* ♯ (*i*,*j*,*j'*,*i'*)
by (*metis obtain-fresh*)
hence *C*: {*QuoteP* (*Var i*) (*Var i'*), *QuoteP* (*Var j*) (*Var j'*)}
⊢ (*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'*)))
using *assms*
by (*auto simp*: *QuoteP.simps* [*of si Var i - ki*] *QuoteP.simps* [*of sj Var j - kj*]
intro!: *SeqQuoteP-Mem-imp-QMem-and-Subset del*: *Conj-I*)
show *?thesis1*
by (*best intro*: *Conj-E1* [*OF C*, *THEN MP-thin*])
show *?thesis2*
by (*best intro*: *Conj-E2* [*OF C*, *THEN MP-thin*])
qed

11.5 Star Property. Universal Quantifier: Lemma 9.7

lemma (**in** *quote-perm*) *SeqQuoteP-Mem-imp-All2*:
assumes *IH*: *insert* (*QuoteP* (*Var i*) (*Var i'*)) (*quote-all p Vs*)
⊢ *α IMP PfP* (*ssubst* ⌊*α*⌋(*insert i Vs*) (*insert i Vs*) *Fi*)
and *sp*: *supp α* − {*atom i*} ⊆ *atom* ‘ *Vs*
and *j*: *j* ∈ *Vs* **and** *j'*: *p* · *j* = *j'*
and *pi*: *pi* = (*atom i* ⇌ *atom i'*) + *p*
and *Fi*: *Fi* = *make-F* (*insert i Vs*) *pi*
and *atoms*: *atom i* ♯ (*j*,*j'*,*s*,*k*,*p*) *atom i'* ♯ (*i*,*p*,*α*)

atom j ♯ (*j*′,*s*,*k*,*α*) *atom j*′ ♯ (*s*,*k*,*α*)
atom s ♯ (*k*,*α*) *atom k* ♯ (*α*,*p*)
shows *insert* (*SeqQuoteP* (*Var j*) (*Var j*′) (*Var s*) (*Var k*)) (*quote-all p* (*Vs*−{*j*}))
⊢ *All2 i* (*Var j*) *α IMP PfP* (*ssubst* ⌊*All2 i* (*Var j*) *α*⌋ *Vs Vs F*)
proof −
have *pj*′ [*simp*]: *p* · *j*′ = *j* **using** *pinv j*′
by (*metis permute-minus-cancel*(*2*))
have [*simp*]: *F j* = *Var j*′ **using** *j j*′
by (*auto simp*: *F-unfold*)
hence *i*′: *atom i*′ ♯ *Vs* **using** *atoms*
by (*auto simp*: *Vs*)
have *fresh-ss* [*simp*]: ⋀*i A*::*fm*. *atom i* ♯ *p* ⟹ *atom i* ♯ *ssubst* (⌊*A*⌋ *Vs*) *Vs F*
by (*simp add*: *vquot-fm-def fresh-ssubst-dbfm*)
obtain *l*::*name* **and** *m*::*name* **and** *n*::*name* **and** *sm*::*name* **and** *sn*::*name* **and** *sm*′::*name* **and** *sn*′::*name*
where *atoms*′: *atom l* ♯ (*p*,*α*,*i*,*j*,*j*′,*s*,*k*,*m*,*n*,*sm*,*sm*′,*sn*,*sn*′)
atom m ♯ (*p*,*α*,*i*,*j*,*j*′,*s*,*k*,*n*,*sm*,*sm*′,*sn*,*sn*′) *atom n* ♯ (*p*,*α*,*i*,*j*,*j*′,*s*,*k*,*sm*,*sm*′,*sn*,*sn*′)
atom sm ♯ (*p*,*α*,*i*,*j*,*j*′,*s*,*k*,*sm*′,*sn*,*sn*′) *atom sm*′ ♯ (*p*,*α*,*i*,*j*,*j*′,*s*,*k*,*sn*,*sn*′)
atom sn ♯ (*p*,*α*,*i*,*j*,*j*′,*s*,*k*,*sn*′) *atom sn*′ ♯ (*p*,*α*,*i*,*j*,*j*′,*s*,*k*)
by (*metis obtain-fresh*)
def *V*′ ≡ {*sm*,*sn*} ∪ *Vs* **and** *p*′ ≡ (*atom sm* ⇌ *atom sm*′) + (*atom sn* ⇌ *atom sn*′) + *p*
def *F*′ ≡ *make-F V*′ *p*′
interpret *qp*′: *quote-perm p*′ *V*′ *F*′
proof *unfold-locales*
show *finite V*′ **by** (*simp add*: *V*′*-def*)
show *atom* ‘ (*p*′ · *V*′) ♯∗ *V*′
using *atoms atoms*′ *p*
by (*auto simp*: *p*′*-def V*′*-def swap-fresh-fresh fresh-at-base-permI*
fresh-star-finite-insert fresh-finite-insert atom-fresh-star-atom-set-conv)
show *F*′ ≡ *make-F V*′ *p*′
by (*rule F*′*-def*)
show − *p*′ = *p*′ **using** *atoms atoms*′ *pinv*
by (*simp add*: *p*′*-def add.assoc perm-self-inverseI fresh-swap fresh-plus-perm*)
qed
have *All2-Zero*: {} ⊢ *All2 i Zero α*
by *auto*
have *Q-All2-Zero*:
quote-all p Vs ⊢ *PfP* (*Q-All* (*Q-Imp* (*Q-Mem* (*Q-Ind Zero*) *Zero*)
(*ssubst* (*vquot-dbfm Vs* (*trans-fm* [*i*] *α*)) *Vs F*)))
using *quote-all-PfP-ssubst* [*OF All2-Zero*] *assms*
by (*force simp add*: *vquot-fm-def supp-conv-fresh*)
have *All2-Eats*: {} ⊢ *All2 i* (*Var sm*) *α IMP α*(*i*::=*Var sn*) *IMP All2 i* (*Eats* (*Var sm*) (*Var sn*)) *α*
using *atoms*′ **apply** *auto*
apply (*rule Ex-I* [**where** *x* = *Var i*], *auto*)
apply (*rule rotate2*)
apply (*blast intro*: *ContraProve Var-Eq-imp-subst-Iff* [*THEN Iff-MP-same*])
done

have [*simp*]: F' *sm* = *Var sm'* F' *sn* = *Var sn'* **using** *atoms'*
by (*auto simp*: *V'-def p'-def qp'.F-unfold swap-fresh-fresh fresh-at-base-permI*)
have *smn'* [*simp*]: *sm* ∈ *V' sn* ∈ *V' sm* ∉ *Vs sn* ∉ *Vs* **using** *atoms'*
by (*auto simp*: *V'-def fresh-finite-set-at-base* [*symmetric*])
hence *Q-All2-Eats*: *quote-all p' V'*
⊢ *PfP* (*ssubst* ⌊*All2 i* (*Var sm*) α⌋ *V' V' F'*) *IMP*
PfP (*ssubst* ⌊α(*i*::=*Var sn*)⌋ *V' V' F'*) *IMP*
PfP (*ssubst* ⌊*All2 i* (*Eats* (*Var sm*) (*Var sn*)) α⌋ *V' V' F'*)
using *sp qp'.quote-all-MonPon2-PfP-ssubst* [*OF All2-Eats subset-refl*]
by (*simp add*: *supp-conv-fresh subset-eq V'-def*)
(*metis Diff-iff empty-iff fresh-ineq-at-base insertE mem-Collect-eq*)
interpret *qpi*: *quote-perm pi insert i Vs Fi*
unfolding *pi*
apply (*rule qp-insert*) **using** *atoms*
apply (*auto simp*: *Fi pi*)
done
have *F'-eq-F*: ⋀*name*. *name* ∈ *Vs* ⟹ *F' name* = *F name*
using *atoms'*
by (*auto simp*: *F-unfold qp'.F-unfold p'-def swap-fresh-fresh V'-def fresh-pj*)
{ fix *t*::*dbtm*
assume *supp t* ⊆ *atom* ‘ *V' supp t* ⊆ *atom* ‘ *Vs*
hence *ssubst* (*vquot-dbtm V' t*) *V' F'* = *ssubst* (*vquot-dbtm Vs t*) *Vs F*
by (*induction t rule*: *dbtm.induct*) (*auto simp*: *F'-eq-F*)
} note *ssubst-v-tm* = *this*
{ fix *A*::*dbfm*
assume *supp A* ⊆ *atom* ‘ *V' supp A* ⊆ *atom* ‘ *Vs*
hence *ssubst* (*vquot-dbfm V' A*) *V' F'* = *ssubst* (*vquot-dbfm Vs A*) *Vs F*
by (*induction A rule*: *dbfm.induct*) (*auto simp*: *ssubst-v-tm F'-eq-F*)
} note *ssubst-v-fm* = *this*
have *ss-noprimes*: *ssubst* (*vquot-dbfm V'* (*trans-fm* [*i*] α)) *V' F'* =
ssubst (*vquot-dbfm Vs* (*trans-fm* [*i*] α)) *Vs F*
apply (*rule ssubst-v-fm*)
using *sp* **apply** (*auto simp*: *V'-def supp-conv-fresh*)
done
{ fix *t*::*dbtm*
assume *supp t* − {*atom i*} ⊆ *atom* ‘ *Vs*
hence *subst i'* (*Var sn'*) (*ssubst* (*vquot-dbtm* (*insert i Vs*) *t*) (*insert i Vs*) *Fi*)
=
ssubst (*vquot-dbtm V'* (*subst-dbtm* (*DBVar sn*) *i t*)) *V' F'*
apply (*induction t rule*: *dbtm.induct*)
using *atoms atoms'*
apply (*auto simp*: *vquot-tm-def pi V'-def qpi.F-unfold qp'.F-unfold p'-def*
fresh-pj swap-fresh-fresh fresh-at-base-permI)
done
} note *perm-v-tm* = *this*
{ fix *A*::*dbfm*
assume *supp A* − {*atom i*} ⊆ *atom* ‘ *Vs*
hence *subst i'* (*Var sn'*) (*ssubst* (*vquot-dbfm* (*insert i Vs*) *A*) (*insert i Vs*) *Fi*)
=

ssubst (*vquot-dbfm* V' (*subst-dbfm* (*DBVar sn*) *i A*)) V' F'
by (*induct A rule*: *dbfm.induct*) (*auto simp*: *Un-Diff perm-v-tm*)
} **note** *perm-v-fm* = *this*
have *quote-all p Vs* $\vdash$ *QuoteP* (*Var i*) (*Var* i') *IMP*
(α *IMP PfP* (*ssubst* $\lfloor\alpha\rfloor$(*insert i Vs*) (*insert i Vs*) *Fi*))
using *IH* **by** *auto*
hence *quote-all p Vs*
$\vdash$ (*QuoteP* (*Var i*) (*Var* i') *IMP*
(α *IMP PfP* (*ssubst* $\lfloor\alpha\rfloor$(*insert i Vs*) (*insert i Vs*) *Fi*))) (i'::=*Var*
sn$'$)
using *atoms IH*
by (*force intro*!: *Subst elim*!: *fresh-quote-all-mem*)
hence *quote-all p Vs*
$\vdash$ *QuoteP* (*Var i*) (*Var sn*$'$) *IMP*
(α *IMP PfP* (*subst* i' (*Var sn*$'$) (*ssubst* $\lfloor\alpha\rfloor$(*insert i Vs*) (*insert i Vs*) *Fi*)))
using *atoms* **by** *simp*
moreover have *subst* i' (*Var sn*$'$) (*ssubst* $\lfloor\alpha\rfloor$(*insert i Vs*) (*insert i Vs*) *Fi*)
= *ssubst* $\lfloor\alpha(i$::=*Var sn*$)\rfloor$ V' V' F'
using *sp*
by (*auto simp*: *vquot-fm-def perm-v-fm supp-conv-fresh subst-fm-trans-commute*
[*symmetric*])
ultimately
have *quote-all p Vs*
$\vdash$ *QuoteP* (*Var i*) (*Var sn*$'$) *IMP* (α *IMP PfP* (*ssubst* $\lfloor\alpha(i$::=*Var sn*$)\rfloor$ V'
V' F'))
by *simp*
hence *quote-all p Vs*
$\vdash$ (*QuoteP* (*Var i*) (*Var sn*$'$) *IMP* (α *IMP PfP* (*ssubst* $\lfloor\alpha(i$::=*Var sn*$)\rfloor$ V'
V' F'))) (i::=*Var sn*)
using ‹*atom i* ♯ -›
by (*force intro*!: *Subst elim*!: *fresh-quote-all-mem*)
hence *quote-all p Vs*
$\vdash$ (*QuoteP* (*Var sn*) (*Var sn*$'$) *IMP*
($\alpha(i$::=*Var sn*$)$ *IMP PfP* (*subst i* (*Var sn*) (*ssubst* $\lfloor\alpha(i$::=*Var sn*$)\rfloor$ V' V'
F'))))
using *atoms atoms*$'$ **by** *simp*
moreover have *subst i* (*Var sn*) (*ssubst* $\lfloor\alpha(i$::=*Var sn*$)\rfloor$ V' V' F')
= *ssubst* $\lfloor\alpha(i$::=*Var sn*$)\rfloor$ V' V' F'
using *atoms atoms*$'$ i'
by (*auto simp*: *swap-fresh-fresh fresh-at-base-permI p*$'$*-def*
intro!: *forget-subst-tm* [*OF qp*$'$*.fresh-ssubst*$'$])
ultimately
have *quote-all p Vs*
$\vdash$ *QuoteP* (*Var sn*) (*Var sn*$'$) *IMP* ($\alpha(i$::=*Var sn*$)$ *IMP PfP* (*ssubst*
$\lfloor\alpha(i$::=*Var sn*$)\rfloor$ V' V' F'))
using *atoms atoms*$'$ **by** *simp*
hence *star0*: *insert* (*QuoteP* (*Var sn*) (*Var sn*$'$)) (*quote-all p Vs*)
$\vdash$ $\alpha(i$::=*Var sn*$)$ *IMP PfP* (*ssubst* $\lfloor\alpha(i$::=*Var sn*$)\rfloor$ V' V' F')
by (*rule anti-deduction*)

have *subst-i-star*: *quote-all* p' V' $\vdash$ $\alpha(i::=Var\ sn)$ *IMP PfP* (*ssubst* $\lfloor\alpha(i::=Var\ sn)\rfloor$ V' V' F')
  apply (*rule thin* [*OF star0*])
  using *atoms'*
  apply (*force simp*: V'*-def* p'*-def fresh-swap fresh-plus-perm fresh-at-base-permI add.assoc*
    quote-all-perm-eq)
  done
have *insert* (*OrdP* (*Var k*)) (*quote-all p* (*Vs*$-\{j\}$))
    $\vdash$ *All j* (*All j'* (*SeqQuoteP* (*Var j*) (*Var j'*) (*Var s*) (*Var k*) *IMP*
      All2 i (*Var j*) α *IMP PfP* (*ssubst* $\lfloor$*All2 i* (*Var j*) $\alpha\rfloor$ *Vs Vs F*)))
    (**is** - $\vdash$ *?scheme*)
  proof (*rule OrdIndH* [**where** *j*=*l*])
    show *atom l* $\sharp$ (*k*, *?scheme*) **using** *atoms atoms' j j' fresh-pVs*
      by (*simp add*: *fresh-Pair F-unfold*)
  next
    have *substj*: $\bigwedge t\ j.$ *atom j* $\sharp$ α $\Longrightarrow$ *atom* ($p \cdot j$) $\sharp$ α $\Longrightarrow$
        subst j t (*ssubst* (*vquot-dbfm Vs* (*trans-fm* [*i*] α)) *Vs F*) =
        ssubst (*vquot-dbfm Vs* (*trans-fm* [*i*] α)) *Vs F*
      by (*auto simp*: *fresh-ssubst'*)
    { fix *W*
      assume *W*: *W* $\subseteq$ *Vs*
      hence *finite W* **by** (*metis Vs infinite-super*)
      hence *quote-all* p' *W* = *quote-all p W* **using** *W*
      proof (*induction*)
        case *empty* **thus** *?case*
          by *simp*
      next
        case (*insert w W*)
        hence $w \in$ *Vs atom sm* $\sharp$ $p \cdot$ *Vs atom sm'* $\sharp$ $p \cdot$ *Vs atom sn* $\sharp$ $p \cdot$ *Vs atom sn'* $\sharp$ $p \cdot$ *Vs*
          using *atoms' Vs* **by** (*auto simp*: *fresh-pVs*)
        hence *atom sm* $\sharp$ $p \cdot w$ *atom sm'* $\sharp$ $p \cdot w$ *atom sn* $\sharp$ $p \cdot w$ *atom sn'* $\sharp$ $p \cdot w$
          by (*metis Vs fresh-at-base*(*2*) *fresh-finite-set-at-base fresh-permute-left*)+
        thus *?case* **using** *insert*
        by (*simp add*: p'*-def swap-fresh-fresh*)
      qed
    }
    hence *quote-all* p' *Vs* = *quote-all p Vs*
      by (*metis subset-refl*)
    also have ... = *insert* (*QuoteP* (*Var j*) (*Var j'*)) (*quote-all p* (*Vs* $-$ $\{j\}$))
      using *j j'* **by** (*auto simp*: *quote-all-def*)
    finally have *quote-all* p' V' =
        $\{$*QuoteP* (*Var sn*) (*Var sn'*), *QuoteP* (*Var sm*) (*Var sm'*)$\}$ $\cup$
        insert (*QuoteP* (*Var j*) (*Var j'*)) (*quote-all p* (*Vs* $-$ $\{j\}$))
      using *atoms'*
      by (*auto simp*: p'*-def* V'*-def fresh-at-base-permI Collect-disj-Un*)
    also have ... = $\{$*QuoteP* (*Var sn*) (*Var sn'*), *QuoteP* (*Var sm*) (*Var sm'*), *QuoteP* (*Var j*) (*Var j'*)$\}$

$\cup$ *quote-all p* $(Vs - \{j\})$
by *blast*
finally have *quote-all′-eq*:
quote-all p' V' =
$\{$*QuoteP* (*Var sn*) (*Var sn′*), *QuoteP* (*Var sm*) (*Var sm′*), *QuoteP* (*Var* *j*) (*Var j′*)$\}$
$\cup$ *quote-all p* $(Vs - \{j\})$ **.**
have *pjV*: $p \cdot j \notin Vs$
by (*metis j perm-exits-Vs*)
hence *jpV*: *atom j* ♯ $p \cdot Vs$
by (*simp add*: *fresh-permute-left pinv fresh-finite-set-at-base*)
show *quote-all p* $(Vs-\{j\}) \vdash$ *All k* (*OrdP* (*Var k*) *IMP* (*All2 l* (*Var k*) (*?scheme*(*k*::= *Var l*)) *IMP ?scheme*))
apply (*rule All-I Imp-I*)+
using *atoms atoms′ j jpV pjV*
apply (*auto simp*: *fresh-at-base fresh-finite-set-at-base j′ elim*!: *fresh-quote-all-mem*)
apply (*rule cut-same* [**where** *A* = *QuoteP* (*Var j*) (*Var j′*)])
apply (*blast intro*: *QuoteP-I*)
apply (*rule cut-same*)
apply (*rule cut1* [*OF SeqQuoteP-lemma* [*of m Var j Var j′ Var s Var k n sm sm′ sn sn′*]], *simp-all*, *blast*)
apply (*rule Imp-I Disj-EH Conj-EH*)+
— case 1, Var j EQ Zero
apply (*simp add*: *vquot-fm-def*)
apply (*rule thin1*)
apply (*rule Var-Eq-subst-Iff* [*THEN Iff-MP-same*], *simp*)
apply (*simp add*: *substj*)
apply (*rule Q-All2-Zero* [*THEN thin*])
using *assms*
apply (*simp add*: *quote-all-def*, *blast*)
— case 2, *Var j EQ Eats* (*Var sm*) (*Var sn*)
apply (*rule Imp-I Conj-EH Ex-EH*)+
using *atoms* **apply** (*auto elim*!: *fresh-quote-all-mem*)
apply (*rule cut-same* [**where** *A* = *QuoteP* (*Var sm*) (*Var sm′*)])
apply (*blast intro*: *QuoteP-I*)
apply (*rule cut-same* [**where** *A* = *QuoteP* (*Var sn*) (*Var sn′*)])
apply (*blast intro*: *QuoteP-I*)
— Eats case. IH for sm
apply (*rule All2-E* [**where** *x*=*Var m*, *THEN rotate12*], *simp-all*, *blast*)
apply (*rule All-E* [**where** *x*=*Var sm*], *simp*)
apply (*rule All-E* [**where** *x*=*Var sm′*], *simp*)
apply (*rule Imp-E*, *blast*)
— Setting up the subgoal
apply (*rule cut-same* [**where** *A* = *PfP* (*ssubst* ⌊*All2 i* (*Eats* (*Var sm*) (*Var sn*)) α⌋ *V′ V′ F′*)])
defer *1*
apply (*rule rotate6*)
apply (*simp add*: *vquot-fm-def*)
apply (*rule Var-Eq-subst-Iff* [*THEN Iff-MP-same*], *force simp add*: *substj*

ss-noprimes j')
  apply (*rule cut-same* [**where** $A = All2\ i\ (Eats\ (Var\ sm)\ (Var\ sn))\ \alpha$])
   apply (*rule All2-cong* [*OF Hyp Iff-refl*, *THEN Iff-MP-same*], *blast*)
    apply (*force elim*!: *fresh-quote-all-mem*
      simp add: *fresh-at-base fresh-finite-set-at-base*, *blast*)
  apply (*rule All2-Eats-E*, *simp*)
  apply (*rule MP-same* [*THEN MP-same*])
  apply (*rule Q-All2-Eats* [*THEN thin*])
  apply (*force simp add*: *quote-all'-eq*)
  — Proving *PfP* (*ssubst* $\lfloor$*All2 i* (*Var sm*) $\alpha\rfloor$ *V' V' F'*)
  apply (*force intro*!: *Imp-E* [*THEN rotate3*] *simp add*: *vquot-fm-def substj j'*
ss-noprimes)
  — Proving *PfP* (*ssubst* $\lfloor\alpha(i{::=}Var\ sn)\rfloor$ *V' V' F'*)
  apply (*rule MP-same* [*OF subst-i-star* [*THEN thin*]])
  apply (*force simp add*: *quote-all'-eq*, *blast*)
  done
 qed
 hence *p1*: *insert* (*OrdP* (*Var k*)) (*quote-all p* (*Vs*−{*j*}))
   $\vdash$ (*All j'* (*SeqQuoteP* (*Var j*) (*Var j'*) (*Var s*) (*Var k*) *IMP*
    All2 i (*Var j*) α *IMP PfP* (*ssubst* $\lfloor$*All2 i* (*Var j*) $\alpha\rfloor$ *Vs Vs F*)))
($j{::=}Var\ j$)
 by (*metis All-D*)
 have *insert* (*OrdP* (*Var k*)) (*quote-all p* (*Vs*−{*j*}))
   $\vdash$ (*SeqQuoteP* (*Var j*) (*Var j'*) (*Var s*) (*Var k*) *IMP*
    All2 i (*Var j*) α *IMP PfP* (*ssubst* $\lfloor$*All2 i* (*Var j*) $\alpha\rfloor$ *Vs Vs F*)) ($j'{::=}Var$
j')
 apply (*rule All-D*)
 using *p1 atoms* **by** *simp*
 thus *?thesis*
 using *atoms*
 by *simp* (*metis SeqQuoteP-imp-OrdP Imp-cut anti-deduction*)
qed

lemma (**in** *quote-perm*) *quote-all-Mem-imp-All2*:
 assumes *IH*: *insert* (*QuoteP* (*Var i*) (*Var i'*)) (*quote-all p Vs*)
   $\vdash$ α *IMP PfP* (*ssubst* $\lfloor\alpha\rfloor$(*insert i Vs*) (*insert i Vs*) *Fi*)
  and *supp* (*All2 i* (*Var j*) α) $\subseteq$ *atom* ‘ *Vs*
  and *j*: *atom j* $\sharp$ (i,α) **and** *i*: *atom i* $\sharp$ *p* **and** *i'*: *atom i'* $\sharp$ (i,p,α)
  and *pi*: $pi = (atom\ i \rightleftharpoons atom\ i') + p$
  and *Fi*: *Fi* = *make-F* (*insert i Vs*) *pi*
 shows *insert* (*All2 i* (*Var j*) α) (*quote-all p Vs*) $\vdash$ *PfP* (*ssubst* $\lfloor$*All2 i* (*Var j*)
$\alpha\rfloor$ *Vs Vs F*)
proof −
 have *sp*: *supp* α − {*atom i*} $\subseteq$ *atom* ‘ *Vs* **and** *jV*: $j \in Vs$
 using *assms*
 by (*auto simp*: *fresh-def supp-Pair*)
 obtain *s*::*name* **and** *k*::*name*
 where *atoms*: *atom s* $\sharp$ (k,i,j,$p{\bullet}j$,α,p) *atom k* $\sharp$ (i,j,$p{\bullet}j$,α,p)
 by (*metis obtain-fresh*)

hence *ii*: *atom i* ♯ (*j*, *p* • *j*, *s*, *k*, *p*) **using** *i j*
by (*simp add*: *fresh-Pair*) (*metis fresh-at-base*(*2*) *fresh-perm fresh-permute-left pinv*)
have *jj*: *atom j* ♯ (*p* • *j*, *s*, *k*, α) **using** *atoms j*
by (*auto simp*: *fresh-Pair*) (*metis atom-fresh-perm jV*)
have *pj*: *atom* (*p* • *j*) ♯ (*s*, *k*, α) **using** *atoms ii sp jV*
by (*simp add*: *fresh-Pair*) (*auto simp*: *fresh-def perm-exits-Vs dest*!: *subsetD*)
show *?thesis*
apply (*rule cut-same* [**where** *A* = *QuoteP* (*Var j*) (*Var* (*p* • *j*))])
apply (*force intro*: *jV Hyp simp add*: *quote-all-def*)
using *atoms*
apply (*auto simp*: *QuoteP.simps* [*of s - - k*] *elim*!: *fresh-quote-all-mem*)
apply (*rule MP-same*)
apply (*rule SeqQuoteP-Mem-imp-All2* [*OF IH sp jV refl pi Fi ii i' jj pj*, *THEN thin*])
apply (*auto simp*: *fresh-at-base-permI quote-all-def intro*!: *fresh-ssubst'*)
done
qed

11.6 The Derivability Condition, Theorem 9.1

lemma *SpecI*: *H* ⊢ *A IMP Ex i A*
by (*metis Imp-I Assume Ex-I subst-fm-id*)

lemma *star*:
fixes *p* :: *perm* **and** *F* :: *name* ⇒ *tm*
assumes *C*: *ss-fm* α
and *p*: *atom* ' (*p* • *V*) ♯* *V* −*p* = *p*
and *V*: *finite V supp* α ⊆ *atom* ' *V*
and *F*: *F* = *make-F V p*
shows *insert* α (*quote-all p V*) ⊢ *PfP* (*ssubst* ⌊α⌋*V V F*)
using *C V p F*
proof (*nominal-induct avoiding*: *p arbitrary*: *V F rule*: *ss-fm.strong-induct*)
case (*MemI i j*) **show** *?case*
proof (*cases i=j*)
case *True* **thus** *?thesis*
by *auto*
next
case *False*
hence *ij*: *atom i* ♯ *j* {*i*, *j*} ⊆ *V* **using** *MemI*
by *auto*
interpret *qp*: *quote-perm p V F*
by *unfold-locales* (*auto simp*: *image-iff F make-F-def p MemI*)
have *insert* (*Var i IN Var j*) (*quote-all p V*) ⊢ *PfP* (*Q-Mem* (*Var* (*p* • *i*)) (*Var* (*p* • *j*)))
apply (*rule QuoteP-Mem-imp-QMem* [*of i j*, *THEN cut3*])
using *ij* **apply** (*auto simp*: *quote-all-def qp.atom-fresh-perm intro*: *Hyp*)
apply (*metis atom-eqvt fresh-Pair fresh-at-base*(*2*) *fresh-permute-iff qp.atom-fresh-perm*)
done

thus *?thesis*
  apply (*simp add*: *vquot-fm-def*)
  using *MemI* **apply** (*auto simp*: *make-F-def*)
  done
qed
next
  case (*DisjI A B*)
    interpret *qp*: *quote-perm p V F*
      by *unfold-locales* (*auto simp*: *image-iff DisjI*)
  show *?case*
    apply *auto*
    apply (*rule-tac* [*2*] *qp.quote-all-Disj-I2-PfP-ssubst*)
    apply (*rule qp.quote-all-Disj-I1-PfP-ssubst*)
    using *DisjI* **by** *auto*
next
  case (*ConjI A B*)
    interpret *qp*: *quote-perm p V F*
      by *unfold-locales* (*auto simp*: *image-iff ConjI*)
  show *?case*
    apply (*rule qp.quote-all-Conj-I-PfP-ssubst*)
    using *ConjI* **by** (*auto intro*: *thin1 thin2*)
next
  case (*ExI A i*)
  interpret *qp*: *quote-perm p V F*
    by *unfold-locales* (*auto simp*: *image-iff ExI*)
  obtain i'::*name* **where** i': *atom* i' ♯ (i,p,A)
    by (*metis obtain-fresh*)
  def $p' \equiv (atom\ i \rightleftharpoons atom\ i') + p$
  def $F' \equiv$ *make-F* (*insert* i V) p'
  have p'*-apply* [*simp*]: !!v. $p' \cdot v$ = (*if* v=i *then* i' *else if* v=i' *then* i *else* $p \cdot v$)
    using ‹*atom* i ♯ p› i'
    by (*auto simp*: p'*-def fresh-Pair fresh-at-base-permI*)
      (*metis atom-eq-iff fresh-at-base-permI permute-eq-iff swap-at-base-simps*(*3*))

  have $p'V$: $p' \cdot V = p \cdot V$
    by (*metis* i' p'*-def permute-plus fresh-Pair qp.fresh-pVs swap-fresh-fresh* ‹*atom* i ♯ p›)
  have i: $i \notin V$ $i \notin p \cdot V$ *atom* i ♯ V *atom* i ♯ $p \cdot V$ *atom* i ♯ $p' \cdot V$ **using** *ExI*
    by (*auto simp*: $p'V$ *fresh-finite-set-at-base notin-V*)
  interpret qp': *quote-perm* p' *insert* i V F'
    by (*auto simp*: *qp.qp-insert* i' p'*-def* F'*-def* ‹*atom* i ♯ p›)
  { fix W t **assume** W: $W \subseteq V$ $i \notin W$ $i' \notin W$
    hence *finite* W **by** (*metis* ‹*finite* V› *infinite-super*)
    hence *ssubst* t W F' = *ssubst* t W F **using** W
      by *induct* (*auto simp*: *qp.ssubst-insert-if* qp'*.ssubst-insert-if qp.F-unfold* qp'*.F-unfold*)
  }
  hence *ss-simp*: *ssubst* ⌊*Ex* i A⌋(*insert* i V) (*insert* i V) F' = *ssubst* ⌊*Ex* i A⌋ V V F **using** i

by (*metis equalityE insertCI p′-apply qp′.perm-exits-Vs qp′.ssubst-vquot-Ex qp.Vs*)
have *qa-p′*: *quote-all p′ V* = *quote-all p V* **using** *i i′ ExI.hyps*(*1*)
by (*auto simp*: *p′-def quote-all-perm-eq*)
have *ss*: (*quote-all p′* (*insert i V*))
⊢ *PfP* (*ssubst* ⌊*A*⌋(*insert i V*) (*insert i V*) *F′*) *IMP*
PfP (*ssubst* ⌊*Ex i A*⌋(*insert i V*) (*insert i V*) *F′*)
apply (*rule qp′.quote-all-MonPon-PfP-ssubst* [*OF SpecI*])
using *ExI* **apply** *auto*
done
hence *insert A* (*quote-all p′* (*insert i V*))
⊢ *PfP* (*ssubst* ⌊*Ex i A*⌋(*insert i V*) (*insert i V*) *F′*)
apply (*rule MP-thin*)
apply (*rule ExI*(*3*) [*of insert i V p′ F′*])
apply (*metis* ‹*finite V*› *finite-insert*)
using ‹*supp* (*Ex i A*) ⊆ -› *qp′.p qp′.pinv i′*
apply (*auto simp*: *F′-def fresh-finite-insert*)
done
hence *insert* (*QuoteP* (*Var i*) (*Var i′*)) (*insert A* (*quote-all p V*))
⊢ *PfP* (*ssubst* ⌊*Ex i A*⌋ *V V F*)
by (*auto simp*: *insert-commute ss-simp qa-p′*)
hence *Exi′*: *insert* (*Ex i′* (*QuoteP* (*Var i*) (*Var i′*))) (*insert A* (*quote-all p V*))

⊢ *PfP* (*ssubst* ⌊*Ex i A*⌋ *V V F*)
by (*auto intro*!: *qp.fresh-ssubst-fm*) (*auto simp*: *ExI i′ fresh-quote-all-mem*)
have *insert A* (*quote-all p V*) ⊢ *PfP* (*ssubst* ⌊*Ex i A*⌋ *V V F*)
using *i′* **by** (*auto intro*: *cut0* [*OF exists-QuoteP Exi′*])
thus *insert* (*Ex i A*) (*quote-all p V*) ⊢ *PfP* (*ssubst* ⌊*Ex i A*⌋ *V V F*)
apply (*rule Ex-E*, *simp*)
apply (*rule qp.fresh-ssubst-fm*) **using** *i ExI*
apply (*auto simp*: *fresh-quote-all-mem*)
done
next
case (*All2I A j i p V F*)
interpret *qp*: *quote-perm p V F*
by *unfold-locales* (*auto simp*: *image-iff All2I*)
obtain *i′*::*name* **where** *i′*: *atom i′* ♯ (*i*,*p*,*A*)
by (*metis obtain-fresh*)
def *p′* ≡ (*atom i* ⇌ *atom i′*) + *p*
def *F′* ≡ *make-F* (*insert i V*) *p′*
interpret *qp′*: *quote-perm p′ insert i V F′*
using ‹*atom i* ♯ *p*› *i′*
by (*auto simp*: *qp.qp-insert p′-def F′-def*)
have *p′-apply* [*simp*]: *p′* · *i* = *i′*
using ‹*atom i* ♯ *p*› **by** (*auto simp*: *p′-def fresh-at-base-permI*)
have *qa-p′*: *quote-all p′ V* = *quote-all p V* **using** *i′ All2I*
by (*auto simp*: *p′-def quote-all-perm-eq*)
have *insert A* (*quote-all p′* (*insert i V*))
⊢ *PfP* (*ssubst* ⌊*A*⌋(*insert i V*) (*insert i V*) *F′*)

apply (*rule All2I.hyps*)
using ‹*supp* (*All2 i* - *A*) ⊆ -› *qp'.p qp'.pinv*
apply (*auto simp*: *F'-def fresh-finite-insert*)
done
hence *insert* (*QuoteP* (*Var i*) (*Var i'*)) (*quote-all p V*)
⊢ *A IMP PfP* (*ssubst* ⌊*A*⌋(*insert i V*) (*insert i V*) (*make-F* (*insert i V*) *p'*))
by (*auto simp*: *insert-commute qa-p' F'-def*)
thus *insert* (*All2 i* (*Var j*) *A*) (*quote-all p V*) ⊢ *PfP* (*ssubst* ⌊*All2 i* (*Var j*) *A*⌋ *V V F*)
using *All2I i' qp.quote-all-Mem-imp-All2* **by** (*simp add*: *p'-def*)
qed

theorem *Provability*:
assumes *Sigma-fm α ground-fm α*
shows {*α*} ⊢ *PfP* ⌈*α*⌉
proof −
obtain *β* **where** *β*: *ss-fm β ground-fm β* {} ⊢ *α IFF β* **using** *assms*
by (*auto simp*: *Sigma-fm-def ground-fm-aux-def*)
hence {*β*} ⊢ *PfP* ⌈*β*⌉ **using** *star* [*of β 0* {}]
by (*auto simp*: *ground-fm-aux-def fresh-star-def*)
then have {*α*} ⊢ *PfP* ⌈*β*⌉ **using** *β*
by (*metis Iff-MP-left'*)
moreover have {} ⊢ *PfP* ⌈*β IMP α*⌉ **using** *β*
by (*metis Conj-E2 Iff-def proved-imp-proved-PfP*)
ultimately show *?thesis*
by (*metis PfP-implies-ModPon-PfP-quot thin0*)
qed

end

Chapter 12

Gdel's Second Incompleteness Theorem

theory *Goedel-II*
imports *Goedel-I Quote*
begin

The connection between *Quote* and *HR* (for interest only).

lemma *Quote-q-Eats* [*intro*]:
  $Quote\ y\ y' \Longrightarrow Quote\ z\ z' \Longrightarrow Quote\ (y \lhd z)\ (q\text{-}Eats\ y'\ z')$
  by (*auto simp*: *Quote-def SeqQuote-def intro*: *BuildSeq2-combine*)

lemma *Quote-q-Succ* [*intro*]: $Quote\ y\ y' \Longrightarrow Quote\ (succ\ y)\ (q\text{-}Succ\ y')$
  by (*auto simp*: *succ-def q-Succ-def*)

lemma *HR-imp-eq-H*: $HR\ x\ z \Longrightarrow z = [\![HF\ x]\!]e$
  apply (*auto simp add*: *SeqHR-def HR-def*)
  apply (*erule BuildSeq2-induct*, *auto simp add*: *q-defs WR-iff-eq-W* [**where** *e=e*])
  done

lemma *HR-Ord-D*: $HR\ x\ y \Longrightarrow Ord\ x \Longrightarrow WR\ x\ y$
  by (*metis HF-Ord HR-imp-eq-H WR-iff-eq-W*)

lemma *WR-Quote*: $WR\ (ord\text{-}of\ i)\ y \Longrightarrow Quote\ (ord\text{-}of\ i)\ y$
  by (*induct i arbitrary*: *y*) (*auto simp*: *Quote-0 WR0-iff WR-succ-iff q-Succ-def* [*symmetric*])

lemma [*simp*]: $\langle\langle 0,0,0\rangle, x, y\rangle = q\text{-}Eats\ x\ y$
  by (*simp add*: *q-Eats-def*)

lemma *HR-imp-Quote*: $coding\text{-}hf\ x \Longrightarrow HR\ x\ y \Longrightarrow Quote\ x\ y$
  apply (*induct x arbitrary*: *y rule*: *coding-hf.induct*, *auto simp*: *WR-Quote HR-Ord-D*)
  apply (*auto dest*!: *HR-imp-eq-H* [**where** *e= e0*])
  by (*metis hpair-def' Quote-0 HR-H Quote-q-Eats*)

interpretation *qp0*: *quote-perm 0* {} *make-F* {} *0*
  proof *unfold-locales* **qed** *auto*

lemma *MonPon-PfP-implies-PfP*:
  $[\![\{\} \vdash \alpha\ IMP\ \beta;\ ground\text{-}fm\ \alpha;\ ground\text{-}fm\ \beta]\!] \Longrightarrow \{PfP\ \lceil\alpha\rceil\} \vdash PfP\ \lceil\beta\rceil$
  using *qp0.quote-all-MonPon-PfP-ssubst*
  by *auto* (*metis Assume PfP-implies-ModPon-PfP-quot proved-iff-proved-PfP thin0*)

lemma *PfP-quot-contra*: $ground\text{-}fm\ \alpha \Longrightarrow \{\} \vdash PfP\ \lceil\alpha\rceil\ IMP\ PfP\ \lceil Neg\ \alpha\rceil\ IMP\ PfP\ \lceil Fls\rceil$
  using *qp0.quote-all-Contra-PfP-ssubst*
  by (*auto simp*: *qp0.quote-all-Contra-PfP-ssubst ground-fm-aux-def*)

Gdel's second incompleteness theorem: If consistent, our theory cannot prove its own consistency.

theorem *Goedel-II*:
  assumes $\neg \{\} \vdash Fls$
    shows $\neg \{\} \vdash Neg\ (PfP\ \lceil Fls\rceil)$
proof −
  from *assms Goedel-I* **obtain** δ
    where *diag*: $\{\} \vdash \delta\ IFF\ Neg\ (PfP\ \lceil\delta\rceil)\ \ \neg \{\} \vdash \delta$
      and *gnd*: $ground\text{-}fm\ \delta$
    by *metis*
  have $\{PfP\ \lceil\delta\rceil\} \vdash PfP\ \lceil PfP\ \lceil\delta\rceil\rceil$
    by (*auto simp*: *Provability ground-fm-aux-def supp-conv-fresh*)
  moreover have $\{PfP\ \lceil\delta\rceil\} \vdash PfP\ \lceil Neg\ (PfP\ \lceil\delta\rceil)\rceil$
    apply (*rule MonPon-PfP-implies-PfP* [*OF - gnd*])
    apply (*metis Conj-E2 Iff-def Iff-sym diag*(*1*))
    apply (*auto simp*: *ground-fm-aux-def supp-conv-fresh*)
    done
  moreover have *ground-fm* (*PfP* $\lceil\delta\rceil$)
    by (*auto simp*: *ground-fm-aux-def supp-conv-fresh*)
  ultimately have $\{PfP\ \lceil\delta\rceil\} \vdash PfP\ \lceil Fls\rceil$ **using** *PfP-quot-contra*
    by (*metis* (*no-types*) *anti-deduction cut2*)
  thus $\neg \{\} \vdash Neg\ (PfP\ \lceil Fls\rceil)$
    by (*metis Iff-MP2-same Neg-mono cut1 diag*)
qed

end

Bibliography

[1] G. S. Boolos. *The Logic of Provability.* Cambridge University Press, 1993.

[2] S. Feferman et al., editors. *Kurt Gödel: Collected Works*, volume I. Oxford University Press, 1986.

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