

Gödel's Incompleteness Theorems

Lawrence C. Paulson

December 17, 2016

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

1	Syntax of Terms and Formulas using Nominal Logic	6
1.1	Terms and Formulas	6
1.1.1	Hf is a pure permutation type	6
1.1.2	The datatypes	6
1.1.3	Substitution	7
1.1.4	Semantics	9
1.1.5	Derived syntax	11
1.1.6	Derived logical connectives	12
1.2	Axioms and Theorems	13
1.2.1	Logical axioms	13
1.2.2	Concrete variables	14
1.2.3	The HF axioms	14
1.2.4	Equality axioms	15
1.2.5	The proof system	16
1.2.6	Derived rules of inference	17
1.2.7	The Deduction Theorem	20
1.2.8	Cut rules	21
1.3	Miscellaneous logical rules	22
1.3.1	Quantifier reasoning	26
1.3.2	Congruence rules	27
1.4	Equality reasoning	28
1.4.1	The congruence property for <i>op EQ</i> , and other basic properties of equality	28
1.4.2	The congruence property for <i>op IN</i>	30
1.4.3	The congruence properties for <i>Eats</i> and <i>HPair</i>	31
1.4.4	Substitution for Equalities	32
1.4.5	Congruence Rules for Predicates	33
1.5	Zero and Falsity	34
1.5.1	The Formula <i>Fls</i>	35
1.5.2	More properties of <i>Zero</i>	36
1.5.3	Basic properties of <i>Eats</i>	37
1.6	Bounded Quantification involving <i>Eats</i>	39
1.7	Induction	41

2	De Bruijn Syntax, Quotations, Codes, V-Codes	42
2.1	de Bruijn Indices (locally-nameless version)	42
2.2	Characterising the Well-Formed de Bruijn Formulas	46
2.2.1	Well-Formed Terms	46
2.2.2	Well-Formed Formulas	47
2.3	Well formed terms and formulas (de Bruijn representation)	48
2.4	Quotations	51
2.4.1	Quotations of de Bruijn terms	51
2.4.2	Quotations of de Bruijn formulas	52
2.5	Definitions Involving Coding	55
2.6	Quotations are Injective	57
2.6.1	Terms	57
2.6.2	Formulas	57
2.6.3	The set Γ of Definition 1.1, constant terms used for coding	58
2.7	V-Coding for terms and formulas, for the Second Theorem	59
3	Basic Predicates	61
3.1	The Subset Relation	61
3.2	Extensionality	64
3.3	The Disjointness Relation	65
3.4	The Foundation Theorem	68
3.5	The Ordinal Property	70
3.6	Induction on Ordinals	74
3.7	Linearity of Ordinals	75
3.8	The predicate <i>OrdNotEqP</i>	77
3.9	Predecessor of an Ordinal	78
3.10	Case Analysis and Zero/SUCC Induction	79
3.11	The predicate <i>HFun-Sigma</i>	81
3.12	The predicate <i>HDomain-Incl</i>	83
3.13	<i>HPair</i> is Provably Injective	85
3.14	<i>SUCC</i> is Provably Injective	86
3.15	The predicate <i>LstSeqP</i>	88
4	Sigma-Formulas and Theorem 2.5	90
4.1	Ground Terms and Formulas	90
4.2	Sigma Formulas	91
4.2.1	Strict Sigma Formulas	91
4.2.2	Closure properties for Sigma-formulas	92
4.3	Lemma 2.2: Atomic formulas are Sigma-formulas	92
4.4	Universal Quantification Bounded by an Arbitrary Term	97
4.5	Lemma 2.3: Sequence-related concepts are Sigma-formulas	98
4.6	A Key Result: Theorem 2.5	99
4.6.1	Sigma-Eats Formulas	99

4.6.2	Preparation	99
4.6.3	The base cases: ground atomic formulas	101
5	Predicates for Terms, Formulas and Substitution	103
5.1	Predicates for atomic terms	103
5.1.1	Free Variables	103
5.1.2	De Bruijn Indexes	104
5.1.3	Various syntactic lemmas	105
5.2	The predicate <i>SeqCTermP</i> , for Terms and Constants	105
5.3	The predicates <i>TermP</i> and <i>ConstP</i>	108
5.3.1	Definition	108
5.3.2	Correctness: It Corresponds to Quotations of Real Terms	108
5.3.3	Correctness properties for constants	109
5.4	Abstraction over terms	110
5.4.1	Defining the syntax: quantified body	110
5.4.2	Defining the syntax: main predicate	112
5.4.3	Correctness: It Coincides with Abstraction over real terms	113
5.5	Substitution over terms	114
5.5.1	Defining the syntax	114
5.6	Abstraction over formulas	116
5.6.1	The predicate <i>AbstAtomicP</i>	116
5.6.2	The predicate <i>AbsMakeForm</i>	117
5.6.3	Defining the syntax: the main <i>AbstForm</i> predicate	119
5.6.4	Correctness: It Coincides with Abstraction over real Formulas	120
5.7	Substitution over formulas	121
5.7.1	The predicate <i>SubstAtomicP</i>	121
5.7.2	The predicate <i>SubstMakeForm</i>	122
5.7.3	Defining the syntax: the main <i>SubstForm</i> predicate	124
5.7.4	Correctness of substitution over formulas	126
5.8	The predicate <i>AtomicP</i>	127
5.9	The predicate <i>MakeForm</i>	127
5.10	The predicate <i>SeqFormP</i>	128
5.11	The predicate <i>FormP</i>	129
5.11.1	Definition	129
5.11.2	Correctness: It Corresponds to Quotations of Real Formulas	130
5.11.3	The predicate <i>VarNonOccFormP</i> (Derived from <i>Sub- stFormP</i>)	133
5.11.4	Correctness for Real Terms and Formulas	133

6	Formalizing Provability	135
6.1	Section 4 Predicates (Leading up to Pf)	135
6.1.1	The predicate <i>SentP</i> , for the Sentential (Boolean) Ax- ioms	135
6.1.2	The predicate <i>Equality-axP</i> , for the Equality Axioms .	136
6.1.3	The predicate <i>HF-axP</i> , for the HF Axioms	136
6.1.4	The specialisation axioms	137
6.1.5	The induction axioms	138
6.1.6	The predicate <i>AxiomP</i> , for any Axioms	142
6.1.7	The predicate <i>ModPonP</i> , for the inference rule Modus Ponens	143
6.1.8	The predicate <i>ExistsP</i> , for the existential rule	143
6.1.9	The predicate <i>SubstP</i> , for the substitution rule	145
6.1.10	The predicate <i>PrfP</i>	146
6.1.11	The predicate <i>PfP</i>	148
6.2	Proposition 4.4	148
6.2.1	Left-to-Right Proof	148
6.2.2	Right-to-Left Proof	150
7	Uniqueness Results: Syntactic Relations are Functions	153
7.0.1	<i>SeqStTermP</i>	153
7.0.2	<i>SubstAtomicP</i>	158
7.0.3	<i>SeqSubstFormP</i>	159
7.0.4	<i>SubstFormP</i>	164
8	Section 6 Material and Gdel's First Incompleteness Theo- rem	165
8.1	The Function W and Lemma 6.1	165
8.1.1	Predicate form, defined on sequences	165
8.1.2	Predicate form of W	166
8.1.3	Proving that these relations are functions	168
8.1.4	The equivalent function	170
8.2	The Function HF and Lemma 6.2	170
8.2.1	Defining the syntax: quantified body	171
8.2.2	Defining the syntax: main predicate	172
8.2.3	Proving that these relations are functions	173
8.2.4	Finally The Function HF Itself	176
8.3	The Function K and Lemma 6.3	177
8.4	The Diagonal Lemma and Gdel's Theorem	178
9	Syntactic Preliminaries for the Second Incompleteness Theo- rem	180
9.1	NotInDom	181
9.2	Restriction of a Sequence to a Domain	182

9.3	Applications to LstSeqP	184
9.4	Ordinal Addition	185
9.4.1	Predicate form, defined on sequences	185
9.4.2	Proving that these relations are functions	187
9.5	A Shifted Sequence	192
9.6	Union of Two Sets	194
9.7	Append on Sequences	196
9.8	LstSeqP and SeqAppendP	198
9.9	Substitution and Abstraction on Terms	199
9.9.1	Atomic cases	199
9.9.2	Non-atomic cases	202
9.9.3	Substitution over a constant	202
9.10	Substitution on Formulas	203
9.10.1	Membership	203
9.10.2	Equality	204
9.10.3	Negation	206
9.10.4	Disjunction	206
9.10.5	Existential	206
9.11	Constant Terms	207
9.12	Proofs	207
10	Pseudo-Coding: Section 7 Material	209
10.1	General Lemmas	209
10.2	Simultaneous Substitution	210
10.3	The Main Theorems of Section 7	214
11	Quotations of the Free Variables	217
11.1	Sequence version of the “Special p-Function, F*”	217
11.1.1	Defining the syntax: quantified body	217
11.1.2	Correctness properties	219
11.2	The “special function” itself	220
11.2.1	Defining the syntax	220
11.2.2	Correctness properties	221
11.3	The Operator <i>quote-all</i>	227
11.3.1	Definition and basic properties	227
11.3.2	Transferring theorems to the level of derivability	228
11.4	Star Property. Equality and Membership: Lemmas 9.3 and 9.4	230
11.5	Star Property. Universal Quantifier: Lemma 9.7	238
11.6	The Derivability Condition, Theorem 9.1	245
12	Gdel’s Second Incompleteness Theorem	249

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 · (s::hf) = s
  instance
    by standard (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 $subst :: name \Rightarrow tm \Rightarrow tm \Rightarrow tm$

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]:

$j \# subst\ i\ x\ t \iff (atom\ i \# t \wedge j \# t) \vee (j \# x \wedge (j \# t \vee j = atom\ i))$
by (induct t rule: tm.induct) (auto simp: fresh-at-base)

lemma *forget-subst-tm* [simp]: $atom\ a \# tm \implies subst\ a\ x\ tm = tm$

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]:

$atom\ j \# tm \implies subst\ j\ u\ (subst\ i\ t\ tm) = subst\ i\ (subst\ j\ u\ t)\ tm$
by (induct tm rule: tm.induct) (auto simp: fresh-Pair)

lemma *subst-tm-commute2* [simp]:

$atom\ j \# t \implies atom\ i \# u \implies i \neq j \implies subst\ j\ u\ (subst\ i\ t\ tm) = subst\ i\ t\ (subst\ j\ u\ tm)$

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 $subst-fm :: fm \Rightarrow name \Rightarrow tm \Rightarrow fm$ ($-'$ ($::=-'$) [1000, 0, 0] 200)

where

$Mem: (Mem\ t\ u)(i::=x) = Mem\ (subst\ i\ x\ t)\ (subst\ i\ x\ u)$
 | $Eq: (Eq\ t\ u)(i::=x) = Eq\ (subst\ i\ x\ t)\ (subst\ i\ x\ u)$
 | $Disj: (Disj\ A\ B)(i::=x) = Disj\ (A(i::=x))\ (B(i::=x))$

```

| Neg: (Neg A)(i::=x) = Neg (A(i::=x))
| Ex:  atom j # (i, x) ==> (Ex j A)(i::=x) = Ex j (A(i::=x))
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 # A ==> 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 # (A(i::=x)) <=> (atom i # A & j # A) ∨ (j # x & (j # A ∨ 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 # A ==> (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' # (A, i, j, t) ==> (Ex i A)(j ::= t) = Ex i' (((i <-> i') . A)(j ::= t))
  by (rule subst [of Ex i' (((i <-> i') . 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 = Thm.term-of ctrm

```

```

    val atoms = Simplifier.premis-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 $e0 :: (name, hf) \text{ finfun} \rightarrow \text{tm} \Rightarrow hf$ — the null environment
where $e0 \equiv \text{finfun-const } 0$

nominal-function $\text{eval-tm} :: (name, hf) \text{ finfun} \Rightarrow \text{tm} \Rightarrow hf$
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

$\text{-EvalTm} :: \text{tm} \Rightarrow (name, hf) \text{ finfun} \Rightarrow hf \quad (\llbracket - \rrbracket - [0,1000]1000)$

translations

$\llbracket tm \rrbracket e \quad == \text{CONST } \text{eval-tm } e \text{ } tm$

nominal-function $\text{eval-fm} :: (name, hf) \text{ finfun} \Rightarrow \text{fm} \Rightarrow \text{bool}$
where

```

  eval-fm e (t IN u) <=> \llbracket t \rrbracket e \in \llbracket u \rrbracket e
| eval-fm e (t EQ u) <=> \llbracket t \rrbracket e = \llbracket u \rrbracket e
| eval-fm e (A OR B) <=> eval-fm e A \vee eval-fm e B
| eval-fm e (Neg A) <=> (\sim eval-fm e A)
| atom k \# e ==> eval-fm e (Ex k A) <=> (\exists 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  $\llbracket t \rrbracket(\text{finfun-update } e \ k \ x) = \llbracket (k' \leftrightarrow k) \cdot t \rrbracket(\text{finfun-update } e \ k' \ x)$ 
using assms
by (induct t rule: tm.induct) (auto simp: permute-flip-at)

```

```

lemma eval-fm-rename:
  assumes atom k'  $\sharp$  A
  shows eval-fm (finfun-update e k x) A = eval-fm (finfun-update e k' x) ((k'  $\leftrightarrow$ 
k)  $\cdot$  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-fm e (Ex k A)  $\longleftrightarrow$  ( $\exists x$ . eval-fm (finfun-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)  $\cdot$  A) = Ex k A
  by (simp add: Abs1-eq-iff flip-def)
  have eval-fm e (Ex k' ((k'  $\leftrightarrow$  k)  $\cdot$  A)) = ( $\exists x$ . eval-fm (finfun-update e k' x) ((k'
 $\leftrightarrow$  k)  $\cdot$  A))
  using k' by simp
  also have ... = ( $\exists x$ . eval-fm (finfun-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  $\implies$   $\llbracket t \rrbracket(\text{finfun-update } e \ i \ x) = \llbracket t \rrbracket e$ 
by (induct t rule: tm.induct) (simp-all add: fresh-at-base)

```

```

lemma forget-eval-fm [simp]:
  atom k  $\sharp$  A  $\implies$  eval-fm (finfun-update e k x) A = eval-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*: $\llbracket \text{subst } i \ t \ u \rrbracket e = \llbracket u \rrbracket (\text{finfun-update } e \ i \ \llbracket t \rrbracket e)$
by (*induct u rule: tm.induct*) (*auto*)

lemma *eval-subst-fm*: $\text{eval-fm } e \ (\text{fm}(i ::= t)) = \text{eval-fm } (\text{finfun-update } e \ i \ \llbracket t \rrbracket e) \ \text{fm}$
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 $\text{HPair } a \ b = \text{Eats } (\text{Eats } \text{Zero } (\text{Eats } (\text{Eats } \text{Zero } \ b) \ a)) \ (\text{Eats } (\text{Eats } \text{Zero } \ a) \ a)$

lemma *HPair-eqvt* [*eqvt*]: $(p \cdot \text{HPair } a \ b) = \text{HPair } (p \cdot a) \ (p \cdot b)$
by (*auto simp: HPair-def*)

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

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

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

lemma *eval-tm-HPair* [*simp*]: $\llbracket \text{HPair } a \ b \rrbracket e = \text{hpair } \llbracket a \rrbracket e \ \llbracket b \rrbracket e$
by (*auto simp: HPair-def hpair-def*)

Ordinals

definition

SUCC :: $tm \Rightarrow tm$ **where**
 $\text{SUCC } x \equiv \text{Eats } x \ x$

fun *ORD-OF* :: $\text{nat} \Rightarrow tm$

where

$\text{ORD-OF } 0 = \text{Zero}$

| $\text{ORD-OF } (\text{Suc } k) = \text{SUCC } (\text{ORD-OF } k)$

lemma *eval-tm-SUCC* [*simp*]: $\llbracket \text{SUCC } t \rrbracket e = \text{succ } \llbracket t \rrbracket e$
by (*simp add: SUCC-def succ-def*)

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

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

lemma *SUCC-subst* [simp]: $\text{subst } i \ t \ (\text{SUCC } k) = \text{SUCC } (\text{subst } i \ t \ k)$
by (*simp add: SUCC-def*)

lemma *eval-tm-ORD-OF* [simp]: $\llbracket \text{ORD-OF } n \rrbracket e = \text{ord-of } n$
by (*induct n auto*)

lemma *ORD-OF-fresh* [simp]: $a \# \text{ORD-OF } n$
by (*induct n (auto simp: SUCC-def)*)

lemma *ORD-OF-eqvt* [eqvt]: $(p \cdot \text{ORD-OF } n) = \text{ORD-OF } (p \cdot n)$
by (*induct n (auto simp: permute-pure SUCC-eqvt)*)

1.1.6 Derived logical connectives

abbreviation *Imp* :: $fm \Rightarrow fm \Rightarrow fm$ (**infixr** *IMP* 125)
where $\text{Imp } A \ B \equiv \text{Disj } (\text{Neg } A) \ B$

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

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

Conjunction

definition *Conj* :: $fm \Rightarrow fm \Rightarrow fm$ (**infixr** *AND* 135)
where $\text{Conj } A \ B \equiv \text{Neg } (\text{Disj } (\text{Neg } A) \ (\text{Neg } B))$

lemma *Conj-eqvt* [eqvt]: $p \cdot (A \ \text{AND} \ B) = (p \cdot A) \ \text{AND} \ (p \cdot B)$
by (*simp add: Conj-def*)

lemma *fresh-Conj* [simp]: $a \# A \ \text{AND} \ B \longleftrightarrow (a \# A \ \wedge \ a \# B)$
by (*auto simp: Conj-def*)

lemma *supp-Conj* [simp]: $\text{supp } (A \ \text{AND} \ B) = \text{supp } A \ \cup \ \text{supp } B$
by (*auto simp: Conj-def*)

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

lemma *Conj-injective-iff* [iff]: $(A \ \text{AND} \ B) = (A' \ \text{AND} \ B') \longleftrightarrow (A = A' \ \wedge \ B = B')$
by (*auto simp: Conj-def*)

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

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

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*]: $a \# A\ IFF\ B \longleftrightarrow (a \# A \wedge a \# B)$

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*]: $eval\text{-}fm\ e\ (Iff\ A\ B) \longleftrightarrow (eval\text{-}fm\ e\ A \longleftrightarrow eval\text{-}fm\ e\ B)$

by (*auto simp: Iff-def*)

1.2 Axioms and Theorems

1.2.1 Logical axioms

inductive-set *boolean-axioms* :: *fm set*

where

| *Ident*: $A\ IMP\ A \in boolean\text{-}axioms$

| *DisjI1*: $A\ IMP\ (A\ OR\ B) \in boolean\text{-}axioms$

| *DisjCont*: $(A\ OR\ A)\ IMP\ A \in boolean\text{-}axioms$

| *DisjAssoc*: $(A\ OR\ (B\ OR\ C))\ IMP\ ((A\ OR\ B)\ OR\ C) \in boolean\text{-}axioms$

| *DisjConj*: $(C\ OR\ A)\ IMP\ (((Neg\ C)\ OR\ B)\ IMP\ (A\ OR\ B)) \in boolean\text{-}axioms$

lemma *boolean-axioms-hold*: $A \in boolean\text{-}axioms \implies eval\text{-}fm\ e\ A$

by (*induct rule: boolean-axioms.induct, auto*)

inductive-set *special-axioms* :: *fm set* **where**

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

lemma *special-axioms-hold*: $A \in special\text{-}axioms \implies eval\text{-}fm\ e\ A$

by (*induct rule: special-axioms.induct, auto*) (*metis eval-subst-fm*)

inductive-set *induction-axioms* :: *fm set* **where**

ind:
 $atom\ j \# (i, A)$
 $\implies A(i ::= Zero)\ IMP\ ((All\ i\ (All\ j\ (A\ IMP\ (A(i ::= Var\ j)\ IMP\ A(i ::= Eats\ (Var\ i)\ (Var\ j))))))\ IMP\ (All\ i\ A))$
 $\in\ induction-axioms$

lemma *twist-forget-eval-fm* [*simp*]:

$atom\ j \# (i, A)$
 $\implies eval-fm\ (finfun-update\ (finfun-update\ (finfun-update\ e\ i\ x)\ j\ y)\ i\ z)\ A =$
 $eval-fm\ (finfun-update\ e\ i\ z)\ A$
by (*metis finfun-update-twice finfun-update-twist forget-eval-fm fresh-Pair*)

lemma *induction-axioms-hold*: $A \in\ induction-axioms \implies eval-fm\ e\ A$

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

$X0 \equiv Abs-name\ (Atom\ (Sort\ "SyntaxN.name"\ [])\ 0)$

abbreviation

$X1 \equiv Abs-name\ (Atom\ (Sort\ "SyntaxN.name"\ [])\ (Suc\ 0))$

— We prefer *Suc 0* because simplification will transform 1 to that form anyway.

abbreviation

$X2 \equiv Abs-name\ (Atom\ (Sort\ "SyntaxN.name"\ [])\ 2)$

abbreviation

$X3 \equiv Abs-name\ (Atom\ (Sort\ "SyntaxN.name"\ [])\ 3)$

abbreviation

$X4 \equiv Abs-name\ (Atom\ (Sort\ "SyntaxN.name"\ [])\ 4)$

1.2.3 The HF axioms

definition *HF1* :: *fm* **where** — the axiom $(z = 0) = (\forall x. x \notin z)$

$HF1 = (Var\ X0\ EQ\ Zero)\ IFF\ (All\ X1\ (Neg\ (Var\ X1\ IN\ Var\ X0)))$

lemma *HF1-holds*: *eval-fm e HF1*

by (*auto simp: HF1-def*)

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

$HF2 \equiv \text{Var } X0 \text{ EQ Eats } (\text{Var } X1) (\text{Var } X2) \text{ IFF}$
 $\text{All } X3 (\text{Var } X3 \text{ IN Var } X0 \text{ IFF Var } X3 \text{ IN Var } X1 \text{ OR Var } X3 \text{ EQ Var } X2)$

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*
 $\text{refl-ax} = \text{Var } X1 \text{ EQ Var } X1$

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

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

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*
 $\text{mem-cong-ax} = ((\text{Var } X1 \text{ EQ Var } X2) \text{ AND } (\text{Var } X3 \text{ EQ Var } X4)) \text{ IMP}$
 $((\text{Var } X1 \text{ IN Var } X3) \text{ IMP } (\text{Var } X2 \text{ IN Var } X4))$

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*
 $\text{eats-cong-ax} = ((\text{Var } X1 \text{ EQ Var } X2) \text{ AND } (\text{Var } X3 \text{ EQ Var } X4)) \text{ IMP}$
 $((\text{Eats } (\text{Var } X1) (\text{Var } X3)) \text{ EQ } (\text{Eats } (\text{Var } X2) (\text{Var } X4)))$

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*
 $\text{equality-axioms} = \{\text{refl-ax}, \text{eq-cong-ax}, \text{mem-cong-ax}, \text{eats-cong-ax}\}$

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* \Rightarrow *fm* \Rightarrow *bool* (**infixl** \vdash 55)

where

Hyp: $A \in H \Longrightarrow H \vdash A$

| *Extra*: $H \vdash \text{extra-axiom}$

| *Bool*: $A \in \text{boolean-axioms} \Longrightarrow H \vdash A$

| *Eq*: $A \in \text{equality-axioms} \Longrightarrow H \vdash A$

| *Spec*: $A \in \text{special-axioms} \Longrightarrow H \vdash A$

| *HF*: $A \in \text{HF-axioms} \Longrightarrow H \vdash A$

| *Ind*: $A \in \text{induction-axioms} \Longrightarrow H \vdash A$

| *MP*: $H \vdash A \text{ IMP } B \Longrightarrow H' \vdash A \Longrightarrow H \cup H' \vdash B$

| *Exists*: $H \vdash A \text{ IMP } B \Longrightarrow \text{atom } i \# B \Longrightarrow \forall C \in H. \text{atom } i \# C \Longrightarrow H \vdash (\text{Ex } i A) \text{ IMP } B$

Soundness theorem!

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

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*: $insert\ A\ (insert\ A\ H) \vdash B \implies insert\ A\ H \vdash B$
 by (metis insert-absorb2)

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

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

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

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

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

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

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

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

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

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

by (*blast intro: thin*)

lemma rotate5:

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

lemma rotate6:

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

lemma rotate7:

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

lemma rotate8:

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

lemma rotate9:

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

lemma rotate10:

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

lemma rotate11:

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

lemma rotate12:

$insert\ A12\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ H)))))))))) \vdash B$
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ H)))))))))) \vdash B$
by (*blast intro: thin*)

lemma rotate13:

$insert\ A13\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ H)))))))))) \vdash B$
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ (insert\ A13\ H)))))))))) \vdash B$
by (*blast intro: thin*)

lemma rotate14:

$insert\ A14\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ (insert\ A13\ H)))))))))) \vdash B$
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ (insert\ A13\ (insert\ A14\ H)))))))))) \vdash B$
by (*blast intro: thin*)

lemma rotate15:

$insert\ A15\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ (insert\ A13\ (insert\ A14\ H)))))))))) \vdash B$
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ (insert\ A13\ (insert\ A14\ (insert\ A15\ H)))))))))) \vdash B$
by (*blast intro: thin*)

lemma MP-same: $H \vdash A \text{ IMP } B \implies H \vdash A \implies H \vdash B$
by (*metis MP Un-absorb*)

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

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

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

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

```

have  $H' \cup H \vdash (\text{Neg } A) \text{ OR } (C \text{ OR } (\text{Neg } A))$ 
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: 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:  $H \vdash B \implies H \vdash A \text{ IMP } B$ 
by (metis Bool Disj-commute MP-same boolean-axioms.DisjI1)

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

1.2.7 The Deduction Theorem

```

lemma deduction-Diff: assumes  $H \vdash B$  shows  $H - \{C\} \vdash C \text{ IMP } B$ 
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  $\#$  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)
      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!]: $insert\ A\ H\ \vdash\ B \implies H\ \vdash\ A\ IMP\ B$
 by (metis Diff-insert-absorb Imp-triv-I deduction-Diff insert-absorb)

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

1.2.8 Cut rules

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

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

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

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

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

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

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

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

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

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

1.3 Miscellaneous logical rules

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *Conj-E*: **assumes** $\text{insert } A (\text{insert } B H) \vdash C$ **shows** $\text{insert } (A \text{ AND } B) H \vdash C$
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** $(\bigwedge B. \text{atom } i \nmid B \Longrightarrow \text{insert } A H \vdash B)$ **shows** $H \vdash \text{Neg } A$
by (*rule Imp-Neg-I [where B = Zero IN Zero]*) (*auto simp: assms*)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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*: $H \vdash A(i::=x) \implies H \vdash Ex\ i\ A$
by (*metis MP-same Spec special-axioms.intros*)

lemma *Ex-E*:
assumes $insert\ A\ H \vdash B\ atom\ i \# B \forall C \in H. atom\ i \# C$
shows $insert\ (Ex\ i\ A)\ H \vdash B$
by (*metis Exists Imp-I anti-deduction assms*)

lemma *Ex-E-with-renaming*:

assumes $insert ((i \leftrightarrow i') \cdot A) H \vdash B$ $atom\ i' \# (A, i, B) \forall C \in H. atom\ i' \# C$
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*: $insert\ A\ H \vdash B \implies \forall C \in H. atom\ i \# C \implies insert (Ex\ i\ A) H \vdash (Ex\ i\ B)$
by (*auto simp add: intro: Ex-I [where x=Var i]*)

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

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

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

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

lemma *All2-E*: $\llbracket atom\ i \# t; H \vdash x\ IN\ t; insert (A(i::=x)) H \vdash B \rrbracket \implies insert (All2\ i\ t\ A) H \vdash B$
apply (*rule All-E [where x=x], auto*)
by (*metis Swap thin1*)

lemma *All2-E'*: $\llbracket H \vdash All2\ i\ t\ A; H \vdash x\ IN\ t; insert (A(i::=x)) H \vdash B; atom\ i \# t \rrbracket \implies H \vdash B$
by (*metis All2-E cut-same*)

1.3.2 Congruence rules

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

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

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

lemma *Conj-cong*: $H \vdash A \text{ IFF } A' \implies H \vdash B \text{ IFF } B' \implies H \vdash A \text{ AND } B \text{ IFF } A' \text{ AND } B'$

by (*metis Conj-def Disj-cong Neg-cong*)

lemma *Imp-cong*: $H \vdash A \text{ IFF } A' \implies H \vdash B \text{ IFF } B' \implies H \vdash (A \text{ IMP } B) \text{ IFF } (A' \text{ IMP } B')$

by (*metis Disj-cong Neg-cong*)

lemma *Iff-cong*: $H \vdash A \text{ IFF } A' \implies H \vdash B \text{ IFF } B' \implies H \vdash (A \text{ IFF } B) \text{ IFF } (A' \text{ IFF } B')$

by (*metis Iff-def Conj-cong Imp-cong*)

lemma *Ex-cong*: $H \vdash A \text{ IFF } A' \implies \forall C \in H. \text{atom } i \# C \implies H \vdash (Ex\ i\ A) \text{ IFF } (Ex\ i\ A')$

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*: $H \vdash A \text{ IFF } A' \implies \forall C \in H. \text{atom } i \# C \implies H \vdash (All\ i\ A) \text{ IFF } (All\ i\ A')$

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

lemma *Subst*: $H \vdash A \implies \forall B \in H. \text{atom } i \# B \implies H \vdash A\ (i::=x)$

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*: $\{\} \vdash (t \text{ EQ } t' \text{ AND } u \text{ EQ } u') \text{ IMP } (t \text{ EQ } u \text{ IMP } t' \text{ EQ } u')$

proof –

obtain $v2::\text{name}$ **and** $v3::\text{name}$ **and** $v4::\text{name}$

where $v2: \text{atom } v2 \# (t, X1, X3, X4)$

and $v3: \text{atom } v3 \# (t, t', X1, v2, X4)$

and $v4: \text{atom } v4 \# (t, t', u, X1, v2, v3)$

by (*metis obtain-fresh*)

have $\{\} \vdash (Var\ X1\ EQ\ Var\ X2 \text{ AND } Var\ X3\ EQ\ Var\ X4) \text{ IMP } (Var\ X1\ EQ\ Var\ X3 \text{ IMP } Var\ X2\ EQ\ Var\ X4)$

by (*rule Eq*) (*simp add: eq-cong-ax-def equality-axioms-def*)

hence $\{\} \vdash (Var\ X1\ EQ\ Var\ X2 \text{ AND } Var\ X3\ EQ\ Var\ X4) \text{ IMP } (Var\ X1\ EQ\ Var\ X3 \text{ IMP } Var\ X2\ EQ\ Var\ X4)$

by (*drule-tac i=X1 and x=Var X1 in Subst*) *simp-all*

hence $\{\} \vdash (Var\ X1\ EQ\ Var\ v2 \text{ AND } Var\ X3\ EQ\ Var\ X4) \text{ IMP } (Var\ X1\ EQ\$

$\text{Var } X3 \text{ IMP Var } v2 \text{ EQ Var } X4$)
by (*drule-tac* $i=X2$ **and** $x=\text{Var } v2$ **in** *Subst*) *simp-all*
hence $\{\} \vdash (\text{Var } X1 \text{ EQ Var } v2 \text{ AND Var } v3 \text{ EQ Var } X4) \text{ IMP } (\text{Var } X1 \text{ EQ Var } v3 \text{ IMP Var } v2 \text{ EQ Var } X4)$
using $v2$
by (*drule-tac* $i=X3$ **and** $x=\text{Var } v3$ **in** *Subst*) *simp-all*
hence $\{\} \vdash (\text{Var } X1 \text{ EQ Var } v2 \text{ AND Var } v3 \text{ EQ Var } v4) \text{ IMP } (\text{Var } X1 \text{ EQ Var } v3 \text{ IMP Var } v2 \text{ EQ Var } v4)$
using $v2 \ v3$
by (*drule-tac* $i=X4$ **and** $x=\text{Var } v4$ **in** *Subst*) *simp-all*
hence $\{\} \vdash (t \text{ EQ Var } v2 \text{ AND Var } v3 \text{ EQ Var } v4) \text{ IMP } (t \text{ EQ Var } v3 \text{ IMP Var } v2 \text{ EQ Var } v4)$
using $v2 \ v3 \ v4$
by (*drule-tac* $i=X1$ **and** $x=t$ **in** *Subst*) *simp-all*
hence $\{\} \vdash (t \text{ EQ } t' \text{ AND Var } v3 \text{ EQ Var } v4) \text{ IMP } (t \text{ EQ Var } v3 \text{ IMP } t' \text{ EQ Var } v4)$
using $v2 \ v3 \ v4$
by (*drule-tac* $i=v2$ **and** $x=t'$ **in** *Subst*) *simp-all*
hence $\{\} \vdash (t \text{ EQ } t' \text{ AND } u \text{ EQ Var } v4) \text{ IMP } (t \text{ EQ } u \text{ IMP } t' \text{ EQ Var } v4)$
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]*: $H \vdash t \text{ EQ } t$

proof –

have $\{\} \vdash \text{Var } X1 \text{ EQ Var } X1$
by (*rule Eq*) (*simp add: equality-axioms-def refl-ax-def*)
hence $\{\} \vdash t \text{ EQ } t$
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** $H \vdash t \text{ EQ } u$ **shows** $H \vdash u \text{ EQ } t$

proof –

have $\{\} \vdash (t \text{ EQ } u \text{ AND } t \text{ EQ } t) \text{ IMP } (t \text{ EQ } t \text{ IMP } u \text{ EQ } t)$
by (*rule Eq-cong1*)
moreover have $\{t \text{ EQ } u\} \vdash t \text{ EQ } u \text{ AND } t \text{ EQ } t$
by (*metis Assume Conj-I Refl*)
ultimately have $\{t \text{ EQ } u\} \vdash u \text{ EQ } t$
by (*metis MP-same MP Refl sup-bot-left*)
thus $H \vdash u \text{ EQ } t$ **by** (*metis assms cut1*)

qed

lemma *Sym-L*: $\text{insert } (t \text{ EQ } u) H \vdash A \implies \text{insert } (u \text{ EQ } t) H \vdash A$

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

lemma *Trans*: **assumes** $H \vdash x \text{ EQ } y$ $H \vdash y \text{ EQ } z$ **shows** $H \vdash x \text{ EQ } z$

proof –

have $\bigwedge H. H \vdash (x \text{ EQ } x \text{ AND } y \text{ EQ } z) \text{ IMP } (x \text{ EQ } y \text{ IMP } x \text{ EQ } z)$

by (metis Eq-cong1 bot-least thin)

moreover have $\{x \text{ EQ } y, y \text{ EQ } z\} \vdash x \text{ EQ } x \text{ AND } y \text{ EQ } z$

by (metis Assume Conj-I Refl thin1)

ultimately have $\{x \text{ EQ } y, y \text{ EQ } z\} \vdash x \text{ EQ } z$

by (metis Hyp MP-same insertI1)

thus ?thesis

by (metis assms cut2)

qed

lemma *Eq-cong*:

assumes $H \vdash t \text{ EQ } t'$ $H \vdash u \text{ EQ } u'$ **shows** $H \vdash t \text{ EQ } u \text{ IFF } t' \text{ EQ } u'$

proof –

{ **fix** $t \ t' \ u \ u'$

assume $H \vdash t \text{ EQ } t'$ $H \vdash u \text{ EQ } u'$

moreover have $\{t \text{ EQ } t', u \text{ EQ } u'\} \vdash t \text{ EQ } u \text{ IMP } t' \text{ EQ } u'$ **using** *Eq-cong1*

by (metis Assume Conj-I MP-null insert-commute)

ultimately have $H \vdash t \text{ EQ } u \text{ IMP } t' \text{ EQ } u'$

by (metis cut2)

}

thus ?thesis

by (metis Iff-def Conj-I assms Sym)

qed

lemma *Eq-Trans-E*: $H \vdash x \text{ EQ } u \implies \text{insert } (t \text{ EQ } u) \ H \vdash A \implies \text{insert } (x \text{ EQ } t) \ H \vdash A$

by (metis Assume Sym-L Trans cut-same thin1 thin2)

1.4.2 The congruence property for *op IN*

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

proof –

obtain $v2::\text{name}$ **and** $v3::\text{name}$ **and** $v4::\text{name}$

where $v2: \text{atom } v2 \ \sharp \ (t, X1, X3, X4)$

and $v3: \text{atom } v3 \ \sharp \ (t, t', X1, v2, X4)$

and $v4: \text{atom } v4 \ \sharp \ (t, t', u, X1, v2, v3)$

by (metis obtain-fresh)

have $\{\} \vdash (\text{Var } X1 \text{ EQ } \text{Var } X2 \text{ AND } \text{Var } X3 \text{ EQ } \text{Var } X4) \text{ IMP } (\text{Var } X1 \text{ IN } \text{Var } X3 \text{ IMP } \text{Var } X2 \text{ IN } \text{Var } X4)$

by (metis mem-cong-ax-def equality-axioms-def insert-iff Eq)

hence $\{\} \vdash (\text{Var } X1 \text{ EQ } \text{Var } v2 \text{ AND } \text{Var } X3 \text{ EQ } \text{Var } X4) \text{ IMP } (\text{Var } X1 \text{ IN } \text{Var } X3 \text{ IMP } \text{Var } v2 \text{ IN } \text{Var } X4)$

by (drule-tac $i=X2$ **and** $x=\text{Var } v2$ **in** *Subst*) *simp-all*

hence $\{\} \vdash (\text{Var } X1 \text{ EQ } \text{Var } v2 \text{ AND } \text{Var } v3 \text{ EQ } \text{Var } X4) \text{ IMP } (\text{Var } X1 \text{ IN } \text{Var } v3 \text{ IMP } \text{Var } v2 \text{ IN } \text{Var } X4)$

using $v2$
by (*drule-tac* $i=X3$ **and** $x=Var\ v3$ **in** *Subst*) *simp-all*
hence $\{\} \vdash (Var\ X1\ EQ\ Var\ v2\ AND\ Var\ v3\ EQ\ Var\ v4)\ IMP\ (Var\ X1\ IN\ Var\ v3\ IMP\ Var\ v2\ IN\ Var\ v4)$
using $v2\ v3$
by (*drule-tac* $i=X4$ **and** $x=Var\ v4$ **in** *Subst*) *simp-all*
hence $\{\} \vdash (t\ EQ\ Var\ v2\ AND\ Var\ v3\ EQ\ Var\ v4)\ IMP\ (t\ IN\ Var\ v3\ IMP\ Var\ v2\ IN\ Var\ v4)$
using $v2\ v3\ v4$
by (*drule-tac* $i=X1$ **and** $x=t$ **in** *Subst*) *simp-all*
hence $\{\} \vdash (t\ EQ\ t'\ AND\ Var\ v3\ EQ\ Var\ v4)\ IMP\ (t\ IN\ Var\ v3\ IMP\ t'\ IN\ Var\ v4)$
using $v2\ v3\ v4$
by (*drule-tac* $i=v2$ **and** $x=t'$ **in** *Subst*) *simp-all*
hence $\{\} \vdash (t\ EQ\ t'\ AND\ u\ EQ\ Var\ v4)\ IMP\ (t\ IN\ u\ IMP\ t'\ IN\ Var\ v4)$
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 $H \vdash t\ EQ\ t'\ H \vdash u\ EQ\ u'$ **shows** $H \vdash t\ IN\ u\ IFF\ t'\ IN\ u'$

proof –

{ **fix** $t\ t'\ u\ u'$

have *cong*: $\{t\ EQ\ t',\ u\ EQ\ u'\} \vdash t\ IN\ u\ IMP\ t'\ IN\ u'$

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*: $\{\} \vdash (t\ EQ\ t'\ AND\ u\ EQ\ u')\ IMP\ (Eats\ t\ u\ EQ\ Eats\ t'\ u')$

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 $\{\} \vdash (Var\ X1\ EQ\ Var\ X2\ AND\ Var\ X3\ EQ\ Var\ X4)\ IMP\ (Eats\ (Var\ X1)\ (Var\ X3)\ EQ\ Eats\ (Var\ X2)\ (Var\ X4))$

by (*metis* *eats-cong-ax-def* *equality-axioms-def* *insert-iff* *Eq*)

hence $\{\} \vdash (Var\ X1\ EQ\ Var\ v2\ AND\ Var\ X3\ EQ\ Var\ X4)\ IMP\ (Eats\ (Var\ X1)\ (Var\ X3)\ EQ\ Eats\ (Var\ v2)\ (Var\ X4))$

by (*drule-tac* $i=X2$ **and** $x=Var\ v2$ **in** *Subst*) *simp-all*

hence $\{\} \vdash (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: $\llbracket H \vdash t \text{ EQ } t'; H \vdash u \text{ EQ } u' \rrbracket \implies H \vdash \text{Eats } t \text{ u EQ Eats } t' \text{ u}'$
 by (metis Conj-I anti-deduction Eats-cong1 cut1)

lemma HPair-cong: $\llbracket H \vdash t \text{ EQ } t'; H \vdash u \text{ EQ } u' \rrbracket \implies H \vdash \text{HPair } t \text{ u EQ HPair } t' \text{ u}'$
 by (metis HPair-def Eats-cong Refl)

lemma SUCC-cong: $H \vdash t \text{ EQ } t' \implies H \vdash \text{SUCC } t \text{ EQ SUCC } t'$
 by (metis Eats-cong SUCC-def)

1.4.4 Substitution for Equalities

lemma Eq-subst-tm-Iff: $\{t \text{ EQ } u\} \vdash \text{subst } i \text{ t tm EQ subst } i \text{ u tm}$
 by (induct tm rule: tm.induct) (auto simp: Eats-cong)

lemma Eq-subst-fm-Iff: $\text{insert } (t \text{ EQ } u) \text{ H} \vdash A(i::=t) \text{ IFF } A(i::=u)$

proof –

have $\{t \text{ EQ } u\} \vdash A(i::=t) \text{ IFF } A(i::=u)$

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: $\text{insert } (\text{Var } i \text{ EQ } t) \text{ H} \vdash A(i::=t) \text{ IFF } A$
 by (metis Eq-subst-fm-Iff Iff-sym subst-fm-id)

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

1.4.5 Congruence Rules for Predicates

lemma *P1-cong*:

fixes $tms :: tm \text{ list}$

assumes $\bigwedge i t x. \text{atom } i \# tms \implies (P t)(i::=x) = P (\text{subst } i x t)$ **and** $H \vdash x \text{ EQ } x'$

shows $H \vdash P x \text{ IFF } P x'$

proof –

obtain $i::\text{name}$ **where** $i: \text{atom } i \# tms$

by (*metis obtain-fresh*)

have $\text{insert } (x \text{ EQ } x') H \vdash (P (\text{Var } i))(i::=x) \text{ IFF } (P (\text{Var } i))(i::=x')$

by (*rule Eq-subst-fm-Iff*)

thus *?thesis using assms i*

by (*metis cut-same subst.simps(2)*)

qed

lemma *P2-cong*:

fixes $tms :: tm \text{ list}$

assumes $\text{sub}: \bigwedge i t u x. \text{atom } i \# tms \implies (P t u)(i::=x) = P (\text{subst } i x t) (\text{subst } i x u)$

and $\text{eq}: H \vdash x \text{ EQ } x' \text{ and } H \vdash y \text{ EQ } y'$

shows $H \vdash P x y \text{ IFF } P x' y'$

proof –

have $yy': \{ y \text{ EQ } y' \} \vdash P x' y \text{ IFF } P x' y'$

by (*rule P1-cong [where tms=[y,x']@tms]*) (*auto simp: fresh-Cons sub*)

have $\{ x \text{ EQ } x' \} \vdash P x y \text{ IFF } P x' y$

by (*rule P1-cong [where tms=[y,x']@tms]*) (*auto simp: fresh-Cons sub*)

hence $\{ x \text{ EQ } x', y \text{ EQ } y' \} \vdash P x y \text{ IFF } P x' y'$

by (*metis Assume Iff-trans cut1 rotate2 yy'*)

thus *?thesis*

by (*metis cut2 eq*)

qed

lemma *P3-cong*:

fixes $tms :: tm \text{ list}$

assumes $\text{sub}: \bigwedge i t u v x. \text{atom } i \# tms \implies$

$(P t u v)(i::=x) = P (\text{subst } i x t) (\text{subst } i x u) (\text{subst } i x v)$

and $\text{eq}: H \vdash x \text{ EQ } x' \text{ and } H \vdash y \text{ EQ } y' \text{ and } H \vdash z \text{ EQ } z'$

shows $H \vdash P x y z \text{ IFF } P x' y' z'$

proof –

obtain $i::\text{name}$ **where** $i: \text{atom } i \# (z, z', y, y', x, x')$

by (*metis obtain-fresh*)

have $tl: \{ y \text{ EQ } y', z \text{ EQ } z' \} \vdash P x' y z \text{ IFF } P x' y' z'$

by (*rule P2-cong [where tms=[z,z',y,y',x,x']@tms]*) (*auto simp: fresh-Cons sub*)

have $hd: \{ x EQ x' \} \vdash P x y z IFF P x' y z$
by (rule P1-cong [where $tms=[z,y,x']@tms$]) (auto simp: fresh-Cons sub)
have $\{ x EQ x', y EQ y', z EQ z' \} \vdash P x y z IFF P x' y' z'$
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 $sub: \bigwedge i t1 t2 t3 t4 x. atom\ i \# tms \implies$
 $(P\ t1\ t2\ t3\ t4)(i ::= x) = P\ (subst\ i\ x\ t1)\ (subst\ i\ x\ t2)\ (subst\ i\ x\ t3)$
 $(subst\ i\ x\ t4)$
and $eq: H \vdash x1 EQ x1' H \vdash x2 EQ x2' H \vdash x3 EQ x3' H \vdash x4 EQ x4'$
shows $H \vdash P\ x1\ x2\ x3\ x4 IFF P\ x1'\ x2'\ x3'\ x4'$
proof –
obtain $i::name$ **where** $i: atom\ i \# (x4,x4',x3,x3',x2,x2',x1,x1')$
by (metis obtain-fresh)
have $tl: \{ x2 EQ x2', x3 EQ x3', x4 EQ x4' \} \vdash P\ x1'\ x2\ x3\ x4 IFF P\ x1'\ x2'\ x3'\ x4'$
by (rule P3-cong [where $tms=[x4,x4',x3,x3',x2,x2',x1,x1']@tms$]) (auto simp: fresh-Cons sub)
have $hd: \{ x1 EQ x1' \} \vdash P\ x1\ x2\ x3\ x4 IFF P\ x1'\ x2\ x3\ x4$
by (auto simp: fresh-Cons sub intro!: P1-cong [where $tms=[x4,x3,x2,x1']@tms$])
have $\{ x1 EQ x1', x2 EQ x2', x3 EQ x3', x4 EQ x4' \} \vdash P\ x1\ x2\ x3\ x4 IFF P\ x1'\ x2'\ x3'\ x4'$
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** $H \vdash (t EQ Zero) IFF (All\ i\ (Neg\ ((Var\ i)\ IN\ t)))$
proof –
obtain $i'::name$ **where** $i': atom\ i' \# (t, X0, X1, i)$
by (rule obtain-fresh)
have $\{ \} \vdash ((Var\ X0)\ EQ\ Zero) IFF (All\ X1\ (Neg\ ((Var\ X1)\ IN\ (Var\ X0))))$
by (simp add: HF HF-axioms-def HF1-def)
then have $\{ \} \vdash (((Var\ X0)\ EQ\ Zero) IFF (All\ X1\ (Neg\ ((Var\ X1)\ IN\ (Var\ X0)))))(X0 ::= t)$
by (rule Subst) simp
hence $\{ \} \vdash (t EQ Zero) IFF (All\ i'\ (Neg\ ((Var\ i')\ IN\ t)))$ **using** i'
by simp
also have $\dots = (FRESH\ i'. (t EQ Zero) IFF (All\ i'\ (Neg\ ((Var\ i')\ IN\ t))))$
using i' **by** simp
also have $\dots = (t EQ Zero) IFF (All\ i\ (Neg\ ((Var\ i)\ IN\ t)))$
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* $\#$ *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*]: $a \# Fls$
 by (*simp add: Fls-def*)

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

lemma *Neg-E* [*intro!*]: $H \vdash A \implies$ *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!]: $H \vdash A \implies \text{insert } B \ H \vdash \text{Fls} \implies H \vdash \text{Neg } (A \text{ IMP } B)$
 by (metis *NegNeg-I Neg-Disj-I Neg-I*)

lemma *Neg-Imp-E* [intro!]: $\text{insert } (\text{Neg } B) (\text{insert } A \ H) \vdash C \implies \text{insert } (\text{Neg } (A \text{ IMP } B)) \ H \vdash C$

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!]: $\text{insert } \text{Fls} \ H \vdash A$
 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*: $H \vdash (\text{Neg } \text{Fls})$
 by (metis *Fls-E Neg-I*)

lemma *ExFalso*: $H \vdash \text{Fls} \implies H \vdash A$
 by (metis *Neg-D truth-provable*)

1.5.2 More properties of Zero

lemma *Eq-Zero-D*:

assumes $H \vdash t \text{ EQ Zero } H \vdash u \text{ IN } t$ shows $H \vdash A$

proof –

obtain $i::\text{name}$ where i : atom $i \# t$

by (rule *obtain-fresh*)

with *assms* have an : $H \vdash (\text{All } i (\text{Neg } ((\text{Var } i) \text{ IN } t)))$

by (metis *Iff-MP-same Mem-Zero-iff*)

have $H \vdash \text{Neg } (u \text{ IN } t)$ 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 # 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 # t* **and** *i2: $\forall B \in H. atom\ i\ \# B$*

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 # (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 # (z,X0,X2,X3)*

and *v2: atom v2 # (t,z,X0,v1,X3)*

and *i': atom i' # (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 $\{\} \vdash ((z \text{ EQ } (\text{Eats } (\text{Var } v1) (\text{Var } v2))) \text{ IFF}$
 $(\text{All } i' (\text{Var } i' \text{ IN } z \text{ IFF } \text{Var } i' \text{ IN } \text{Var } v1 \text{ OR } \text{Var } i' \text{ EQ } \text{Var } v2)))$
using $v1 \ v2 \ i'$ **by** (*simp add: Conj-def Iff-def*)
hence $\{\} \vdash (z \text{ EQ } (\text{Eats } t (\text{Var } v2))) \text{ IFF}$
 $(\text{All } i' (\text{Var } i' \text{ IN } z \text{ IFF } \text{Var } i' \text{ IN } t \text{ OR } \text{Var } i' \text{ EQ } \text{Var } v2))$
using $v1 \ v2 \ i'$ **by** (*drule-tac i=v1 and x=t in Subst simp-all*)
hence $\{\} \vdash (z \text{ EQ } \text{Eats } t \ u) \text{ IFF}$
 $(\text{All } i' (\text{Var } i' \text{ IN } z \text{ IFF } \text{Var } i' \text{ IN } t \text{ OR } \text{Var } i' \text{ EQ } \ u))$
using $v1 \ v2 \ i'$ **by** (*drule-tac i=v2 and x=u in Subst simp-all*)
also have $\dots = (\text{FRESH } i'. (z \text{ EQ } \text{Eats } t \ u) \text{ IFF } (\text{All } i' (\text{Var } i' \text{ IN } z \text{ IFF } \text{Var } i' \text{ IN } t \text{ OR } \text{Var } i' \text{ EQ } \ u)))$
using i' **by** *simp*
also have $\dots = (z \text{ EQ } \text{Eats } t \ u) \text{ IFF } (\text{All } i (\text{Var } i \text{ IN } z \text{ IFF } \text{Var } i \text{ IN } t \text{ OR } \text{Var } i \text{ EQ } \ u))$
using *assms i' by simp*
finally show *?thesis*
by (*rule thin0*)
qed

lemma *Eq-Eats-I:*

$H \vdash \text{All } i (\text{Var } i \text{ IN } z \text{ IFF } \text{Var } i \text{ IN } t \text{ OR } \text{Var } i \text{ EQ } \ u) \implies \text{atom } i \ \sharp (z, t, u) \implies$
 $H \vdash z \text{ EQ } \text{Eats } t \ u$
by (*metis Iff-MP2-same Eq-Eats-iff*)

lemma *Mem-Eats-Iff:*

$H \vdash x \text{ IN } (\text{Eats } t \ u) \text{ IFF } x \text{ IN } t \text{ OR } x \text{ EQ } \ u$

proof –

obtain $i::\text{name}$ **where** $\text{atom } i \ \sharp (\text{Eats } t \ u, t, u)$

by (*rule obtain-fresh*)

thus *?thesis*

using *Iff-MP-same [OF Eq-Eats-iff, THEN All-D]*

by *auto*

qed

lemma *Mem-Eats-I1:* $H \vdash u \text{ IN } t \implies H \vdash u \text{ IN } \text{Eats } t \ z$

by (*metis Disj-I1 Iff-MP2-same Mem-Eats-Iff*)

lemma *Mem-Eats-I2:* $H \vdash u \text{ EQ } z \implies H \vdash u \text{ IN } \text{Eats } t \ z$

by (*metis Disj-I2 Iff-MP2-same Mem-Eats-Iff*)

lemma *Mem-Eats-E:*

assumes $A: \text{insert } (u \text{ IN } t) \ H \vdash C$ **and** $B: \text{insert } (u \text{ EQ } z) \ H \vdash C$

shows $\text{insert } (u \text{ IN } \text{Eats } t \ z) \ H \vdash C$

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*
 $[\text{THEN rotate3}] \text{ Mem-Eats-E } [\text{THEN rotate4}] \text{ Mem-Eats-E } [\text{THEN rotate5}]$
 $\text{ Mem-Eats-E } [\text{THEN rotate6}] \text{ Mem-Eats-E } [\text{THEN rotate7}] \text{ Mem-Eats-E } [\text{THEN rotate8}]$

declare *Mem-Eats-EH* [intro!]

lemma *Mem-SUCC-I1*: $H \vdash u \text{ IN } t \implies H \vdash u \text{ IN } \text{SUCC } t$
by (*metis Mem-Eats-I1 SUCC-def*)

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

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

lemma *Mem-SUCC-E*:
assumes *insert* ($u \text{ IN } t$) $H \vdash C$ *insert* ($u \text{ EQ } t$) $H \vdash C$ **shows** *insert* ($u \text{ IN } \text{SUCC } t$) $H \vdash C$
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* ($\text{Eats } t \text{ u EQ Zero}$) $H \vdash A$
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* ($\text{Zero EQ Eats } t \text{ u}$) $H \vdash A$
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*: $H \vdash t \text{ EQ } t' \implies H \vdash A \text{ IFF } A' \implies \forall C \in H. \text{atom } i \# C \implies$
 $H \vdash (\text{All2 } i \text{ } t \text{ } A) \text{ IFF } (\text{All2 } i \text{ } t' \text{ } A')$
by (*metis All-cong Imp-cong Mem-cong Refl*)

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

lemma *All2-Eats-I-D*:

$atom\ i\ \sharp\ (t,u) \implies \{ All2\ i\ t\ A,\ A(i::=u) \} \vdash (All2\ i\ (Eats\ t\ u)\ A)$
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*:

$\llbracket atom\ i\ \sharp\ (t,u); H \vdash All2\ i\ t\ A; H \vdash A(i::=u) \rrbracket \implies H \vdash (All2\ i\ (Eats\ t\ u)\ A)$
by (*rule cut2* [*OF All2-Eats-I-D*], *auto*)

lemma *All2-Eats-E1*:

$\llbracket atom\ i\ \sharp\ (t,u); \forall C \in H. atom\ i\ \sharp\ C \rrbracket \implies insert\ (All2\ i\ (Eats\ t\ u)\ A)\ H \vdash All2\ i\ t\ A$
by *auto* (*metis Assume Ex-I Imp-E Mem-Eats-I1 Neg-mono subst-fm-id*)

lemma *All2-Eats-E2*:

$\llbracket atom\ i\ \sharp\ (t,u); \forall C \in H. atom\ i\ \sharp\ C \rrbracket \implies insert\ (All2\ i\ (Eats\ t\ u)\ A)\ H \vdash A(i::=u)$
by (*rule All-E* [**where** $x=u$]) (*auto intro: ContraProve Mem-Eats-I2*)

lemma *All2-Eats-E*:

assumes $i: atom\ i\ \sharp\ (t,u)$
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*:

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

lemma *All2-SUCC-E*:

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

lemma *All2-SUCC-E'*:

assumes $H \vdash u\ EQ\ SUCC\ t$
and $atom\ i\ \sharp\ t \forall C \in H. atom\ i\ \sharp\ C$
and $insert\ (All2\ i\ t\ A)\ (insert\ (A(i::=t))\ H) \vdash B$
shows $insert\ (All2\ i\ u\ A)\ H \vdash B$
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.suppl [simp]
declare dbfm.suppl [simp]
```

```
fun lookup :: name list  $\Rightarrow$  nat  $\Rightarrow$  name  $\Rightarrow$  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  $\sharp$  e  $\implies$  name  $\notin$  set e
  by (metis List.finite-set fresh-finite-set-at-base fresh-set)
```

```
lemma lookup-notin: x  $\notin$  set e  $\implies$  lookup e n x = DBVar x
  by (induct e arbitrary: n) auto
```

lemma *lookup-in*:

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

lemma *lookup-fresh*: $x \# \text{lookup } e \ n \ y \longleftrightarrow y \in \text{set } e \vee x \neq \text{atom } y$

by (*induct* *arbitrary*: *n* *rule*: *lookup.induct*) (*auto* *simp*: *pure-fresh* *fresh-at-base*)

lemma *lookup-eqvt*[*eqvt*]: $(p \cdot \text{lookup } xs \ n \ x) = \text{lookup } (p \cdot xs) \ (p \cdot n) \ (p \cdot x)$

by (*induct* *xs* *arbitrary*: *n*) (*simp-all* *add*: *permute-pure*)

lemma *lookup-inject* [*iff*]: $(\text{lookup } e \ n \ x = \text{lookup } e \ n \ y) \longleftrightarrow x = y$

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* \Rightarrow *tm* \Rightarrow *dbtm*

where

trans-tm *e* *Zero* = *DBZero*

| *trans-tm* *e* (*Var* *k*) = *lookup* *e* 0 *k*

| *trans-tm* *e* (*Eats* *t* *u*) = *DBEats* (*trans-tm* *e* *t*) (*trans-tm* *e* *u*)

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*]: $i \# \text{trans-tm } e \ t \longleftrightarrow i \# t \vee i \in \text{atom } e \wedge \text{set } e$

by (*induct* *t* *rule*: *tm.induct*, *auto* *simp*: *lookup-fresh* *fresh-at-base*)

lemma *trans-tm-forget*: $\text{atom } i \# t \implies \text{trans-tm } [i] \ t = \text{trans-tm } [] \ t$

by (*induct* *t* *rule*: *tm.induct*, *auto* *simp*: *fresh-Pair*)

nominal-function (*invariant* $\lambda(xs, -) y. \text{atom } e \wedge \text{set } xs \#* y$)

trans-fm :: *name list* \Rightarrow *fm* \Rightarrow *dbfm*

where

trans-fm *e* (*Mem* *t* *u*) = *DBMem* (*trans-tm* *e* *t*) (*trans-tm* *e* *u*)

| *trans-fm* *e* (*Eq* *t* *u*) = *DBEq* (*trans-tm* *e* *t*) (*trans-tm* *e* *u*)

| *trans-fm* *e* (*Disj* *A* *B*) = *DBDisj* (*trans-fm* *e* *A*) (*trans-fm* *e* *B*)

| *trans-fm* *e* (*Neg* *A*) = *DBNeg* (*trans-fm* *e* *A*)

| $\text{atom } k \# e \implies \text{trans-fm } e \ (Ex \ k \ A) = \text{DBEx} \ (\text{trans-fm } (k\#e) \ A)$

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]:  $i \# \text{trans-fm } e \ A \longleftrightarrow i \# \ A \vee i \in \text{atom } \text{'set } e$ 
  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]:  $\text{trans-fm } e \ (\text{Conj } A \ B) = \text{DBConj } (\text{trans-fm } e \ A) \ (\text{trans-fm } e \ B)$ 
  by (simp add: Conj-def)

lemma trans-tm-inject [iff]:  $(\text{trans-tm } e \ t = \text{trans-tm } e \ u) \longleftrightarrow t = u$ 
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]:  $(\text{trans-fm } e \ A = \text{trans-fm } e \ B) \longleftrightarrow A = B$ 
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-1st]]
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 ↔ c) · A) = trans-fm [c] ((j ↔ c) · B)
  by simp
then show (i ↔ c) · A = (j ↔ c) · B by simp
qed

```

2.2 Characterising the Well-Formed de Bruijn Formulas

2.2.1 Well-Formed Terms

```

inductive wf-dbtm :: dbtm ⇒ bool
  where
    Zero: wf-dbtm DBZero
  | Var: wf-dbtm (DBVar name)
  | Eats: wf-dbtm t1 ⇒ wf-dbtm t2 ⇒ wf-dbtm (DBEats t1 t2)

```

equivariance *wf-dbtm*

```

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

```

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

```

lemma wf-dbtm-imp-is-tm:
  assumes wf-dbtm x
  shows ∃ t::tm. x = trans-tm [] t
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 ↔ (∃ t::tm. x = trans-tm [] t)
  by (metis wf-dbtm-imp-is-tm wf-dbtm-trans-tm)

```

```

nominal-function abst-dbtm :: name ⇒ nat ⇒ dbtm ⇒ dbtm
  where
    abst-dbtm name i DBZero = DBZero

```


$| \text{ abst-dbtm name } i \text{ (DBVar name')} = (\text{if name} = \text{name}' \text{ then DBInd } i \text{ else DBVar name'})$
 $| \text{ abst-dbtm name } i \text{ (DBInd } j) = \text{DBInd } j$
 $| \text{ abst-dbtm name } i \text{ (DBEats } t1 \text{ } t2) = \text{DBEats (abst-dbtm name } i \text{ } t1) \text{ (abst-dbtm name } i \text{ } t2)$
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 \Rightarrow name \Rightarrow dbtm \Rightarrow dbtm*
where

$\text{subst-dbtm } u \text{ } i \text{ DBZero} = \text{DBZero}$
 $| \text{ subst-dbtm } u \text{ } i \text{ (DBVar name)} = (\text{if } i = \text{name} \text{ then } u \text{ else DBVar name})$
 $| \text{ subst-dbtm } u \text{ } i \text{ (DBInd } j) = \text{DBInd } j$
 $| \text{ subst-dbtm } u \text{ } i \text{ (DBEats } t1 \text{ } t2) = \text{DBEats (subst-dbtm } u \text{ } i \text{ } t1) \text{ (subst-dbtm } u \text{ } i \text{ } t2)$
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 \longleftrightarrow 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 \Rightarrow nat \Rightarrow dbfm \Rightarrow dbfm*
where

$\text{abst-dbfm name } i \text{ (DBMem } t1 \text{ } t2) = \text{DBMem (abst-dbtm name } i \text{ } t1) \text{ (abst-dbtm name } i \text{ } t2)$
 $| \text{ abst-dbfm name } i \text{ (DBEq } t1 \text{ } t2) = \text{DBEq (abst-dbtm name } i \text{ } t1) \text{ (abst-dbtm name } i \text{ } t2)$
 $| \text{ abst-dbfm name } i \text{ (DBDisj } A1 \text{ } A2) = \text{DBDisj (abst-dbfm name } i \text{ } A1) \text{ (abst-dbfm name } i \text{ } A2)$
 $| \text{ abst-dbfm name } i \text{ (DBNeg } A) = \text{DBNeg (abst-dbfm name } i \text{ } A)$
 $| \text{ abst-dbfm name } i \text{ (DBEx } A) = \text{DBEx (abst-dbfm name } (i+1) \text{ } A)$

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* \Rightarrow *name* \Rightarrow *dbfm* \Rightarrow *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* \longleftrightarrow *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* \Rightarrow *bool*

where

Mem: *wf-dbtm* *t1* \Longrightarrow *wf-dbtm* *t2* \Longrightarrow *wf-dbfm* (*DBMem* *t1* *t2*)

| *Eq*: *wf-dbtm* *t1* \Longrightarrow *wf-dbtm* *t2* \Longrightarrow *wf-dbfm* (*DBEq* *t1* *t2*)

| *Disj*: *wf-dbfm* *A1* \Longrightarrow *wf-dbfm* *A2* \Longrightarrow *wf-dbfm* (*DBDisj* *A1* *A2*)

| *Neg*: *wf-dbfm* *A* \Longrightarrow *wf-dbfm* (*DBNeg* *A*)

| *Ex*: *wf-dbfm* *A* \Longrightarrow *wf-dbfm* (*DBEx* (*abst-dbfm* *name* 0 *A*))

equivariance *wf-dbfm*

lemma *atom-fresh-abst-dbtm* [*simp*]: *atom* *i* $\#$ *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*

Setting up strong induction: "avoiding" for name. Necessary 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  $\implies$  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  $\exists A::fm. x = trans-fm [] A$ 
using assms
proof (induct rule: wf-dbfm.induct)
  case (Mem t1 t2) thus ?case
    by (metis trans-fm.simps(1) wf-dbfm-imp-is-tm)
next
  case (Eq t1 t2) thus ?case
    by (metis trans-fm.simps(2) wf-dbfm-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-dbfm-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*]: $atom\ name\ \# u \implies abst-dbtm\ name\ j\ u = u$
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*:
 $name \neq name' \implies atom\ name' \# u \implies$
 $subst-dbtm\ u\ name\ (abst-dbtm\ name'\ j\ t) = abst-dbtm\ name'\ j\ (subst-dbtm\ u\ name\ t)$
by (*induct t rule: dbtm.induct*) *auto*

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

lemma *subst-trans-commute* [*simp*]:
 $atom\ i\ \# e \implies subst-dbtm\ (trans-tm\ e\ u)\ i\ (trans-tm\ e\ t) = trans-tm\ e\ (subst\ i\ u\ t)$
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-dbfm.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*:
 $du = trans-tm\ []\ u \implies subst-dbfm\ du\ i\ (trans-fm\ []\ A) = trans-fm\ []\ (A(i::=u))$
by (*metis subst-fm-trans-commute*)

2.4 Quotations

fun *htuple* :: *nat* \Rightarrow *hf* **where**
 htuple 0 = $\langle 0, 0 \rangle$
 | *htuple* (*Suc* *k*) = $\langle 0, \text{htuple } k \rangle$

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

lemma *eval-tm-HTuple* [*simp*]: $\llbracket \text{HTuple } n \rrbracket e = \text{htuple } n$
by (*induct* *n*) *auto*

lemma *fresh-HTuple* [*simp*]: $x \# \text{HTuple } n$
by (*induct* *n*) *auto*

lemma *HTuple-eqvt*[*eqvt*]: $(p \cdot \text{HTuple } n) = \text{HTuple } (p \cdot n)$
by (*induct* *n*, *auto simp: HPair-eqvt permute-pure*)

lemma *htuple-nonzero* [*simp*]: $\text{htuple } k \neq 0$
by (*induct* *k*) *auto*

lemma *htuple-inject* [*iff*]: $\text{htuple } i = \text{htuple } j \longleftrightarrow i=j$
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* \Rightarrow *nat*
where *nat-of-name* *x* = *nat-of* (*atom* *x*)

lemma *nat-of-name-inject* [*simp*]: $\text{nat-of-name } n1 = \text{nat-of-name } n2 \longleftrightarrow n1 = n2$
by (*metis nat-of-name-def atom-components-eq-iff atom-eq-iff sort-of-atom-eq*)

definition *name-of-nat* :: *nat* \Rightarrow *name*
where *name-of-nat* *n* \equiv *Abs-name* (*Atom* (*Sort* "*SyntaxN.name*" []) *n*)

lemma *nat-of-name-Abs-eq* [*simp*]: $\text{nat-of-name } (\text{Abs-name } (\text{Atom } (\text{Sort } \text{"SyntaxN.name"} [])) \text{ } n)) = n$
by (*auto simp: nat-of-name-def atom-name-def Abs-name-inverse*)

lemma *nat-of-name-name-eq* [*simp*]: $\text{nat-of-name } (\text{name-of-nat } n) = n$
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]: $\llbracket \text{quot-dbtm } t \rrbracket e = \llbracket \text{quot-dbtm } u \rrbracket e \longleftrightarrow t = u$

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: $n > 0 \implies \text{htuple } n = \langle 0, \text{htuple } (n - 1) \rangle$
by (metis Suc-diff-1 htuple.simps(2))

lemma HTuple-minus-1: $n > 0 \implies \text{HTuple } n = \text{HPair Zero } (\text{HTuple } (n - 1))$
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]: $\llbracket \text{quot-dbfm } A \rrbracket e = \llbracket \text{quot-dbfm } B \rrbracket e \iff A=B$
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 \Rightarrow tm ([-])

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

instance ..
end

lemma quot-dbtm-fresh [simp]: $s \# (\text{quot-dbtm } t)$
by (induct t rule: dbtm.induct) auto

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

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

lemma *quot-Var*: $[Var\ x] = SUCC\ (ORD-OF\ (nat-of-name\ x))$
by (*simp add: quot-tm-def*)

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

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

lemma *eval-quot-dbtm-ignore*:
 $\llbracket quot-dbtm\ t \rrbracket e = \llbracket quot-dbtm\ t \rrbracket e'$
by (*induct t rule: dbtm.induct*) *auto*

lemma *eval-quot-dbfm-ignore*:
 $\llbracket quot-dbfm\ A \rrbracket e = \llbracket quot-dbfm\ A \rrbracket e'$
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*]: $s \# (quot-dbfm\ A)$
by (*induct A rule: dbfm.induct*) *auto*

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

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

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

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

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

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

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

lemma *eval-quot-fm-ignore*: **fixes** $A:: fm$ **shows** $[[A]]e = [[A]]e'$
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* :: $name \Rightarrow hf$
where $q-Var\ i \equiv succ\ (ord-of\ (nat-of-name\ i))$

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

abbreviation *Q-Eats* :: $tm \Rightarrow tm \Rightarrow tm$
where $Q-Eats\ t\ u \equiv HPair\ (HTuple\ (Suc\ 0))\ (HPair\ t\ u)$

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

abbreviation *Q-Succ* :: $tm \Rightarrow tm$
where $Q-Succ\ t \equiv Q-Eats\ t\ t$

definition *q-Succ* :: $hf \Rightarrow hf$
where $q-Succ\ x \equiv q-Eats\ x\ x$

lemma *quot-Succ*: $[SUCC\ x] = Q-Succ\ [x]$
by (*auto simp: SUCC-def quot-Eats*)

abbreviation *Q-HPair* :: $tm \Rightarrow tm \Rightarrow tm$
where $Q-HPair\ t\ u \equiv$
 $Q-Eats\ (Q-Eats\ Zero\ (Q-Eats\ (Q-Eats\ Zero\ u)\ t))$
 $(Q-Eats\ (Q-Eats\ Zero\ t)\ t)$

definition *q-HPair* :: $hf \Rightarrow hf \Rightarrow hf$
where $q-HPair\ x\ y \equiv$
 $q-Eats\ (q-Eats\ 0\ (q-Eats\ (q-Eats\ 0\ y)\ x))$
 $(q-Eats\ (q-Eats\ 0\ x)\ x)$

abbreviation *Q-Mem* :: $tm \Rightarrow tm \Rightarrow tm$
where $Q-Mem\ t\ u \equiv HPair\ (HTuple\ 0)\ (HPair\ t\ u)$

definition *q-Mem* :: $hf \Rightarrow hf \Rightarrow hf$

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

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

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

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

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

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

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

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 $Q\text{-Ex } t \equiv \text{HPair } (\text{HTuple } 5) \ t$

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

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 $q\text{-defs} = q\text{-Var-def } q\text{-Ind-def } q\text{-Eats-def } q\text{-HPair-def } q\text{-Eq-def } q\text{-Mem-def}$
 $q\text{-Disj-def } q\text{-Neg-def } q\text{-Conj-def } q\text{-Imp-def } q\text{-Ex-def } q\text{-All-def}$

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

lemma *quot-subst-eq*: $\llbracket A(i::=t) \rrbracket = \text{quot-dbfm} (\text{subst-dbfm} (\text{trans-tm } [] t) i (\text{trans-fm } [] A))$
by (*metis quot-fm-def subst-fm-trans-commute*)

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

2.6 Quotations are Injective

2.6.1 Terms

lemma *eval-tm-inject* [*simp*]: **fixes** $t::\text{tm}$ **shows** $\llbracket t \rrbracket e = \llbracket u \rrbracket e \longleftrightarrow t=u$
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** $A::\text{fm}$ **shows** $\llbracket A \rrbracket e = \llbracket B \rrbracket e \longleftrightarrow A=B$
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* \Rightarrow *bool*

where

Ord: $\exists i. x = \text{ORD-OF } i \Longrightarrow \text{coding-tm } x$

| *HPair*: $\text{coding-tm } x \Longrightarrow \text{coding-tm } y \Longrightarrow \text{coding-tm } (\text{HPair } x \ y)$

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* [*A*]

by (metis *quot-dbfm-coding* *quot-fm-def*)

inductive *coding-hf* :: *hf* \Rightarrow *bool*

where

Ord: $\exists i. x = \text{ord-of } i \Longrightarrow \text{coding-hf } x$

| *HPair*: $\text{coding-hf } x \Longrightarrow \text{coding-hf } y \Longrightarrow \text{coding-hf } (\langle x, y \rangle)$

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-tm* *t* \Longrightarrow *coding-hf* $\llbracket t \rrbracket e$

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  $\Rightarrow$  dbtm  $\Rightarrow$  tm
  where
    vquot-dbtm V DBZero = Zero
  | vquot-dbtm V (DBVar name) = (if name  $\in$  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  $\longleftrightarrow$  i  $\#$  tm  $\vee$  i  $\notin$  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  $\Rightarrow$  dbfm  $\Rightarrow$  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  $\longleftrightarrow$  i  $\#$  fm  $\vee$  i  $\notin$  atom ' V
  by (induct fm rule: dbfm.induct) (auto simp: HPair-def HTuple-minus-1)

```

class vquot =

fixes vquot :: 'a \Rightarrow name set \Rightarrow tm ([_]- [0,1000]1000)

instantiation tm :: vquot

begin

definition vquot-tm :: tm \Rightarrow name set \Rightarrow 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  $\cap$  supp t = atom ' W  $\cap$  supp t  $\implies$  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  $\Rightarrow$  name set  $\Rightarrow$  tm
    where vquot-fm A V = vquot-dbfm V (trans-fm [] A)
  instance ..
end

lemma vquot-fm-fresh [simp]: fixes A::fm shows  $i \# \lfloor A \rfloor V \longleftrightarrow i \# A \vee i \notin \text{atom}$ 
' 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  $\cap$  supp A = atom ' W  $\cap$  supp A  $\implies$  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  $\notin$  supp A  $\implies$   $\lfloor A \rfloor (\text{insert } i \text{ } V) = \lfloor A \rfloor V$ 
  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  $\Rightarrow$  tm  $\Rightarrow$  fm (infixr SUBS 150)
  where atom z  $\#$  (t, u)  $\Longrightarrow$  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  $\longleftrightarrow$  a  $\#$  t  $\wedge$  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)  $\longleftrightarrow$  ( $\llbracket t \rrbracket e \leq \llbracket u \rrbracket e$ )
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* $((\text{Var } i \text{ IN } t) H \vdash (\text{Var } i \text{ IN } u \text{ atom } i \# (t,u) \forall B \in H. \text{atom } i \# B$
shows $H \vdash t \text{ SUBS } u$
by $(\text{subst } \text{Subset.simps } [\text{of } i]) (\text{auto simp: assms})$

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

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

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

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

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

lemma *Zero-SubsetE*: $H \vdash A \implies \text{insert } (\text{Zero} \text{ SUBS } X) H \vdash A$
by $(\text{rule } \text{thin1})$

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


```

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

```

```

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

```

```

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

```

```

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

```

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

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

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

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

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

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

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

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

lemma *All2-Subset*: $\llbracket H \vdash \text{All2 } l \ k \ P; H \vdash k' \text{ SUBS } k; \{P\} \vdash P'; \text{atom } l \ \sharp \ (k', k) \rrbracket \implies H \vdash \text{All2 } l \ k' \ P'$
by (*rule cut2 [OF All2-Subset-lemma]*) *auto*

3.2 Extensionality

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

```

proof auto
  show {Zero EQ y} ⊢ 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} ⊢ Zero EQ y
  by (metis AssumeH(2) Subset-Zero-D Sym)
qed
next
  show {} ⊢ 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 {} ⊢ (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 ⊢ y SUBS x ⇒ H ⊢ x SUBS y ⇒ H ⊢ x EQ y
  by (metis Conj-I Extensionality Iff-MP2-same)

```

```

lemma EQ-imp-SUBS: insert (t EQ u) H ⊢ (t SUBS u)

```

```

proof –
  have {t EQ u} ⊢ (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 ⊢ (t SUBS u)
  by (metis EQ-imp-SUBS Sym-L)

```

```

lemma Equality-E: insert (t SUBS u) (insert (u SUBS t) H) ⊢ A ⇒ insert (t
EQ u) H ⊢ 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 $Disjoint :: tm \Rightarrow tm \Rightarrow fm$
where $atom\ z \# (t, u) \Longrightarrow Disjoint\ t\ u = All2\ z\ t\ (Neg\ ((Var\ z)\ IN\ u))$
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*]: $a \# Disjoint\ t\ u \longleftrightarrow a \# t \wedge a \# u$
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 $Disjoint\ cong$: $H \vdash t\ EQ\ t' \Longrightarrow H \vdash u\ EQ\ u' \Longrightarrow H \vdash Disjoint\ t\ u\ IFF\ Disjoint\ t'\ u'$
by (*rule P2-cong*) *auto*

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

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

lemma $Disjoint\ commute$: $\{ Disjoint\ t\ u \} \vdash Disjoint\ u\ t$

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

lemma *Disjoint-commute-D*: $insert\ (Disjoint\ t\ u)\ H \vdash A \implies insert\ (Disjoint\ u\ t)\ H \vdash A$
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 $\{\} \vdash Disjoint\ Zero\ t$
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*: $\{ Disjoint\ (Eats\ x\ y)\ z \} \vdash Disjoint\ x\ z$
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*: $\{ Disjoint\ (Eats\ x\ y)\ z \} \vdash Neg(y\ IN\ z)$
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*:
 $insert\ (Disjoint\ x\ z)\ (insert\ (Neg(y\ IN\ z))\ H) \vdash A \implies insert\ (Disjoint\ (Eats\ x\ y)\ z)\ H \vdash A$

```

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 Reft, 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  $A = (\text{Neg } ((\text{Var } i) \text{ IN } z) \text{ AND } \text{Disjoint } (\text{Var } i) z))$ ])
apply (rule Foundation-lemma [THEN cut1], auto)
done

```

```

lemma Mem-Neg-refl:  $\{\} \vdash \text{Neg } (x \text{ IN } x)$ 
proof –
  obtain  $i::\text{name}$  where  $i: \text{atom } i \# x$ 
    by (metis obtain-fresh)
  have  $\{\} \vdash \text{Disjoint } x (\text{Eats Zero } x)$ 
    apply (rule cut-same [OF Foundation [where  $z = \text{Eats Zero } x$ ]]) using  $i$ 
    apply auto
    apply (rule cut-same [where  $A = \text{Disjoint } x (\text{Eats Zero } x)$ ])
    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!]:  $\text{insert } (x \text{ IN } x) H \vdash A$ 
  by (metis Disj-I1 Mem-Neg-refl anti-deduction thin0)

```

```

lemma Mem-non-refl: assumes  $H \vdash x \text{ IN } x$  shows  $H \vdash A$ 
  by (metis Mem-refl-E assms cut-same)

```

```

lemma Mem-Neg-sym:  $\{ x \text{ IN } y, y \text{ IN } x \} \vdash \text{Fls}$ 
proof –
  obtain  $i::\text{name}$  where  $i: \text{atom } i \# (x,y)$ 
    by (metis obtain-fresh)
  have  $\{\} \vdash \text{Disjoint } x (\text{Eats Zero } y) \text{ OR } \text{Disjoint } y (\text{Eats Zero } x)$ 
    apply (rule cut-same [OF Foundation [where  $i=i$  and  $z = \text{Eats } (\text{Eats Zero } y) x$ ]]) 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:  $\text{insert } (x \text{ IN } y) (\text{insert } (y \text{ IN } x) H) \vdash A$ 
  by (rule cut-thin [OF Mem-Neg-sym]) auto

```

3.5 The Ordinal Property

nominal-function $OrdP :: tm \Rightarrow fm$
where $\llbracket atom\ y \ \sharp\ (x, z); atom\ z \ \sharp\ x \rrbracket \Longrightarrow$
 $OrdP\ x = All2\ y\ x\ ((Var\ y)\ SUBS\ x\ AND\ All2\ z\ (Var\ y)\ ((Var\ z)\ SUBS\ (Var\ y)))$
by $(auto\ simp: eqvt-def\ OrdP-graph-aux-def\ flip-fresh-fresh)\ (metis\ obtain-fresh)$

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

lemma
shows $OrdP\text{-fresh-iff}\ [simp]: a \ \sharp\ OrdP\ x \longleftrightarrow a \ \sharp\ x \quad (\mathbf{is}\ ?thesis1)$
and $eval\text{-fm-}OrdP\ [simp]: eval\text{-fm}\ e\ (OrdP\ x) \longleftrightarrow Ord\ \llbracket x \rrbracket e \quad (\mathbf{is}\ ?thesis2)$
proof $-$
obtain $z::name$ **and** $y::name$ **where** $atom\ z \ \sharp\ x\ atom\ y \ \sharp\ (x, z)$
by $(metis\ obtain-fresh)$
thus $?thesis1\ ?thesis2$
by $(auto\ simp: OrdP.simps\ [of\ y - z])\ Ord-def\ Transset-def)$
qed

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

lemma $OrdP\text{-cong}: H \vdash x\ EQ\ x' \Longrightarrow H \vdash OrdP\ x\ IFF\ OrdP\ x'$
by $(rule\ P1\text{-cong})\ auto$

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

lemma $OrdP\text{-Mem-E}$:


```

assumes atom z  $\#$  (k,l)
          insert (OrdP k) H  $\vdash$  l IN k
          insert (l SUBS k) (insert (All2 z l (Var z SUBS l)) H)  $\vdash$  A
shows insert (OrdP k) H  $\vdash$  A
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  $\vdash$  k IN l and l: H  $\vdash$  OrdP l shows H  $\vdash$  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 }  $\vdash$  SUCC k' SUBS k
by auto (metis Assume thin1 OrdP-Mem-imp-Subset)

```

```

lemma SUCC-Subset-Ord: H  $\vdash$  k' IN k  $\implies$  H  $\vdash$  OrdP k  $\implies$  H  $\vdash$  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 }  $\vdash$  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  $\vdash$  OrdP k  $\implies$  H  $\vdash$  i IN j  $\implies$  H  $\vdash$  j IN k  $\implies$  H  $\vdash$  i IN k
by (blast intro: cut3 [OF OrdP-Trans-lemma])

```

```

lemma Ord-IN-Ord0:
assumes l: H  $\vdash$  l IN k
shows insert (OrdP k) H  $\vdash$  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 }  $\vdash$  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 }  $\vdash$  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*: $H \vdash l \text{ IN } k \implies H \vdash \text{OrdP } k \implies H \vdash \text{OrdP } l$
by (metis *Ord-IN-Ord0 cut-same*)

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

lemma *OrdP-Zero* [*simp*]: $H \vdash \text{OrdP } \text{Zero}$

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

lemma *OrdP-SUCC-I0*: $\{\text{OrdP } k\} \vdash \text{OrdP } (\text{SUCC } k)$

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: $\{\text{Var } y \text{ IN } \text{SUCC } k, \text{OrdP } k\} \vdash \text{Var } y \text{ SUBS } \text{SUCC } k$
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*: $\{\text{Var } y \text{ IN } k, \text{Var } z \text{ IN } \text{Var } y, \text{OrdP } k\} \vdash \text{Var } z \text{ SUBS } \text{Var } y$
apply (rule *OrdP-Mem-E* [of w - Var y, THEN *rotate3*]) **using** *atoms*
apply (*auto intro: All2-E* [THEN *thin1*])
done
have $\{\text{Var } y \text{ EQ } k, \text{Var } z \text{ IN } k, \text{OrdP } k\} \vdash \text{Var } z \text{ SUBS } \text{Var } y$
by (metis *AssumeH(2) AssumeH(3) EQ-imp-SUBS2 OrdP-Mem-imp-Subset*)

Subset-trans)

hence *eq-case*: $\{ \text{Var } y \text{ EQ } k, \text{Var } z \text{ IN } \text{Var } y, \text{OrdP } k \} \vdash \text{Var } z \text{ SUBS } \text{Var } y$
by (*rule cut3*) (*auto intro: EQ-imp-SUBS [THEN cut1] Subset-D*)

have *2*: $\{ \text{Var } z \text{ IN } \text{Var } y, \text{Var } y \text{ IN } \text{SUCC } k, \text{OrdP } k \} \vdash \text{Var } z \text{ SUBS } \text{Var } y$
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*: $H \vdash \text{OrdP } k \implies H \vdash \text{OrdP } (\text{SUCC } k)$
by (*metis OrdP-SUCC-I0 cut1*)

lemma *Zero-In-OrdP*: $\{ \text{OrdP } x \} \vdash x \text{ EQ } \text{Zero} \text{ OR } \text{Zero} \text{ IN } x$

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 Ref]])*
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*: $\text{insert } (\text{OrdP } (\text{HPair } x y)) H \vdash A$

proof –

have $\{ \text{OrdP } (\text{HPair } x y) \} \vdash A$
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*: $\text{insert } (\text{Zero EQ HPair } x y) H \vdash A$
by (*metis Eats-EQ-Zero-E2 HPair-def*)

lemmas $Zero\text{-}Eq\text{-}HPairEH = Zero\text{-}Eq\text{-}HPairE\ Zero\text{-}Eq\text{-}HPairE$ [THEN rotate2]
 $Zero\text{-}Eq\text{-}HPairE$ [THEN rotate3] $Zero\text{-}Eq\text{-}HPairE$ [THEN rotate4] $Zero\text{-}Eq\text{-}HPairE$
[THEN rotate5]
 $Zero\text{-}Eq\text{-}HPairE$ [THEN rotate6] $Zero\text{-}Eq\text{-}HPairE$ [THEN rotate7]
 $Zero\text{-}Eq\text{-}HPairE$ [THEN rotate8] $Zero\text{-}Eq\text{-}HPairE$ [THEN rotate9] $Zero\text{-}Eq\text{-}HPairE$
[THEN rotate10]
declare $Zero\text{-}Eq\text{-}HPairEH$ [intro!]

lemma $HPair\text{-}Eq\text{-}ZeroE$: $insert (HPair\ x\ y\ EQ\ Zero)\ H \vdash A$
by ($metis\ Sym\text{-}L\ Zero\text{-}Eq\text{-}HPairE$)

lemmas $HPair\text{-}Eq\text{-}ZeroEH = HPair\text{-}Eq\text{-}ZeroE\ HPair\text{-}Eq\text{-}ZeroE$ [THEN rotate2]
 $HPair\text{-}Eq\text{-}ZeroE$ [THEN rotate3] $HPair\text{-}Eq\text{-}ZeroE$ [THEN rotate4] $HPair\text{-}Eq\text{-}ZeroE$
[THEN rotate5]
 $HPair\text{-}Eq\text{-}ZeroE$ [THEN rotate6] $HPair\text{-}Eq\text{-}ZeroE$ [THEN rotate7]
 $HPair\text{-}Eq\text{-}ZeroE$ [THEN rotate8] $HPair\text{-}Eq\text{-}ZeroE$ [THEN rotate9] $HPair\text{-}Eq\text{-}ZeroE$
[THEN rotate10]
declare $HPair\text{-}Eq\text{-}ZeroEH$ [intro!]

3.6 Induction on Ordinals

lemma $OrdInd\text{-}lemma$:

assumes j : $atom\ (j::name) \# (i,A)$
shows $\{ OrdP\ (Var\ i) \} \vdash (All\ i\ (OrdP\ (Var\ i)\ IMP\ ((All2\ j\ (Var\ i)\ (A(i::=$
 $Var\ j))))\ IMP\ A)))\ IMP\ A$

proof –

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

```

```

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 Reft,
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 Reft,
THEN Iff-MP-same])
apply (blast intro: ContraProve [THEN rotate4] OrdP-Trans)
done

```

qed

qed

lemma *OrdP-linear-imp*: $\{\} \vdash OrdP\ x\ IMP\ OrdP\ y\ IMP\ x\ IN\ y\ OR\ x\ EQ\ y\ OR\ y\ IN\ x$

proof —

```

obtain  $i::name$  and  $j::name$ 
where  $atoms: atom\ i\ \#\ (x,y)\ atom\ j\ \#\ (x,y,i)$ 
by (metis obtain-fresh)
have  $\{\ OrdP\ (Var\ i) \} \vdash (OrdP\ (Var\ j)\ IMP\ (Var\ i\ IN\ Var\ j\ OR\ Var\ i\ EQ\ Var\ j\ OR\ Var\ j\ IN\ Var\ i))(j::=y)$ 
using  $atoms$  by (metis All-D OrdP-linear-lemma fresh-Pair)
hence  $\{\} \vdash OrdP\ (Var\ i)\ IMP\ OrdP\ y\ IMP\ (Var\ i\ IN\ y\ OR\ Var\ i\ EQ\ y\ OR\ y\ IN\ Var\ i)$ 
using  $atoms$  by auto
hence  $\{\} \vdash (OrdP\ (Var\ i)\ IMP\ OrdP\ y\ IMP\ (Var\ i\ IN\ y\ OR\ Var\ i\ EQ\ y\ OR\ y\ IN\ Var\ i))(i::=x)$ 
by (metis Subst empty-iff)
thus ?thesis
using  $atoms$  by auto

```

qed

lemma *OrdP-linear*:

assumes $H \vdash OrdP\ x\ H \vdash OrdP\ y$

$insert\ (x\ IN\ y)\ H \vdash A\ insert\ (x\ EQ\ y)\ H \vdash A\ insert\ (y\ IN\ x)\ H \vdash A$

shows $H \vdash A$

proof –
have $\{ \text{OrdP } x, \text{OrdP } y \} \vdash x \text{ IN } y \text{ OR } x \text{ EQ } y \text{ OR } y \text{ IN } x$
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*: $\{ \text{OrdP } k \} \vdash \text{Zero IN SUCC } k$
by (*rule OrdP-linear [OF OrdP-Zero OrdP-SUCC-I]*) (*force simp: SUCC-def*)⁺

3.8 The predicate *OrdNotEqP*

nominal-function *OrdNotEqP* :: $tm \Rightarrow tm \Rightarrow fm$ (**infixr** *NEQ* 150)
where $\text{OrdNotEqP } x \ y = \text{OrdP } x \ \text{AND } \text{OrdP } y \ \text{AND } (x \ \text{IN } y \ \text{OR } y \ \text{IN } x)$
by (*auto simp: eqvt-def OrdNotEqP-graph-aux-def*)

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

lemma *OrdNotEqP-fresh-iff* [*simp*]: $a \# \text{OrdNotEqP } x \ y \longleftrightarrow a \# x \wedge a \# y$
by *auto*

lemma *eval-fm-OrdNotEqP* [*simp*]: $\text{eval-fm } e \ (\text{OrdNotEqP } x \ y) \longleftrightarrow \text{Ord } \llbracket x \rrbracket e \wedge \text{Ord } \llbracket y \rrbracket e \wedge \llbracket x \rrbracket e \neq \llbracket y \rrbracket e$
by (*auto simp: hmem-not-refl*) (*metis Ord-linear*)

lemma *OrdNotEqP-subst* [*simp*]: $(\text{OrdNotEqP } x \ y)(i::=t) = \text{OrdNotEqP } (\text{subst } i \ t \ x) \ (\text{subst } i \ t \ y)$
by *simp*

lemma *OrdNotEqP-cong*: $H \vdash x \ \text{EQ } x' \Longrightarrow H \vdash y \ \text{EQ } y' \Longrightarrow H \vdash \text{OrdNotEqP } x \ y \ \text{IFF } \text{OrdNotEqP } x' \ y'$
by (*rule P2-cong*) *auto*

lemma *OrdNotEqP-self-contr*: $\{ x \ \text{NEQ } x \} \vdash \text{Fls}$
by *auto*

lemma *OrdNotEqP-OrdP-E*: $\text{insert } (\text{OrdP } x) \ (\text{insert } (\text{OrdP } y) \ H) \vdash A \Longrightarrow \text{insert } (x \ \text{NEQ } y) \ H \vdash A$
by (*auto intro: thin1 rotate2*)

lemma *OrdNotEqP-I*: $\text{insert } (x \ \text{EQ } y) \ H \vdash \text{Fls} \Longrightarrow H \vdash \text{OrdP } x \Longrightarrow H \vdash \text{OrdP } y \Longrightarrow H \vdash x \ \text{NEQ } y$
by (*rule OrdP-linear [of - x y]*) (*auto intro: ExFalse thin1 Disj-I1 Disj-I2*)

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

lemma *OrdNotEqP-imp-Neg-Eq*: $\{ x \ \text{NEQ } y \} \vdash \text{Neg } (x \ \text{EQ } y)$
by (*blast intro: OrdNotEqP-cong [THEN Iff-MP2-same] OrdNotEqP-self-contr*)

[of x, THEN cut1])

lemma *OrdNotEqP-E*: $H \vdash x \text{ EQ } y \implies \text{insert } (x \text{ NEQ } y) H \vdash A$
 by (metis *ContraProve OrdNotEqP-imp-Neg-Eq rcut1*)

3.9 Predecessor of an Ordinal

lemma *OrdP-set-max-lemma*:

assumes j : atom ($j::\text{name}$) $\#$ i and k : atom ($k::\text{name}$) $\#$ (i, j)

shows $\{\} \vdash (\text{Neg } (\text{Var } i \text{ EQ } \text{Zero}) \text{ AND } (\text{All2 } j (\text{Var } i) (\text{OrdP } (\text{Var } j)))) \text{ IMP}$
 $(\text{Ex } j (\text{Var } j \text{ IN } \text{Var } i \text{ AND } (\text{All2 } k (\text{Var } i) (\text{Var } k \text{ SUBS } \text{Var } j))))$

proof –

obtain $l::\text{name}$ **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 $x = \text{Var } l$], 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 $A = \text{Var } j \text{ IN } \text{Var } l$])

apply (rule *Ex-I* [where $x = \text{Var } l$], auto *intro*: *Mem-Eats-I2*)

apply (rule *Ex-I* [where $x = \text{Var } l$], auto *intro*: *Mem-Eats-I2 ContraProve*)

apply (rule *Ex-I* [where $x = \text{Var } k$], auto)

apply (metis *Assume Subset-trans OrdP-Mem-imp-Subset thin1*)

apply (rule *Ex-I* [where $x = \text{Var } l$], 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 $x = \text{Var } j$], auto *intro*: *Mem-Eats-I1*)

apply (rule *All2-E* [where $x = \text{Var } k$, THEN *rotate3*], auto)

apply (rule *Ex-I* [where $x = \text{Var } k$], *simp*)

apply (metis *Assume NegNeg-I Neg-Disj-I rotate3*)

apply (rule *cut-same* [where $A = \text{OrdP } (\text{Var } j)$])

apply (rule *All2-E* [where $x = \text{Var } j$, THEN *rotate3*], auto)

apply (rule *cut-same* [where $A = \text{Var } l \text{ EQ } \text{Var } j \text{ OR } \text{Var } l \text{ IN } \text{Var } j$])

apply (rule *OrdP-linear* [of - $\text{Var } l \text{ Var } j$], 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 $\{ \text{OrdP } x, \text{Neg } (x \text{ EQ Zero}) \} \vdash \text{Ex } j \text{ (Var } j \text{ IN } x \text{ AND (All2 } k \text{ } x \text{ (Var } k \text{ SUBS Var } j)))$

proof –

obtain $i::\text{name}$ **where** i : atom $i \# (x,j,k)$

by (*metis obtain-fresh*)

have $\{ \} \vdash ((\text{Neg } (\text{Var } i \text{ EQ Zero}) \text{ AND (All2 } j \text{ (Var } i) (\text{OrdP } (\text{Var } j)))) \text{ IMP}$
 $(\text{Ex } j \text{ (Var } j \text{ IN Var } i \text{ AND (All2 } k \text{ (Var } i) (\text{Var } k \text{ SUBS Var } j)))))(i::=x)$

apply (*rule Subst [OF OrdP-set-max-lemma]*)

using i k **apply** *auto*

done

hence $\{ \text{Neg } (x \text{ EQ Zero}) \text{ AND (All2 } j \text{ } x \text{ (OrdP } (\text{Var } j))) \}$
 $\vdash \text{Ex } j \text{ (Var } j \text{ IN } x \text{ AND (All2 } k \text{ } x \text{ (Var } k \text{ SUBS Var } j)))$

using i j k **by** *simp (metis anti-deduction)*

hence $\{ \text{All2 } j \text{ } x \text{ (OrdP } (\text{Var } j)), \text{Neg } (x \text{ EQ Zero}) \}$

$\vdash \text{Ex } j \text{ (Var } j \text{ IN } x \text{ AND (All2 } k \text{ } x \text{ (Var } k \text{ SUBS Var } j)))$

by (*rule cut1 (metis Assume Conj-I thin1)*)

moreover have $\{ \text{OrdP } x \} \vdash \text{All2 } j \text{ } x \text{ (OrdP } (\text{Var } j))$ **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 $\{ \text{OrdP } x, \text{Neg } (x \text{ EQ Zero}) \} \vdash \text{Ex } p \text{ (OrdP } (\text{Var } p) \text{ AND } x \text{ EQ SUCC}$
 $(\text{Var } p))$

proof –

obtain $j::\text{name}$ and $k::\text{name}$ **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*:
 \llbracket *insert (x EQ Zero) H ⊢ A;*
 \llbracket *insert (x EQ SUCC (Var k)) (insert (OrdP (Var k)) H) ⊢ A;*
 \llbracket *atom k # (x,A); $\forall C \in H. \text{atom } k \# C$* \rrbracket
 \implies *insert (OrdP x) H ⊢ A*
by (*rule cut-same [OF OrdP-cases-disj [of k]] (auto simp: insert-commute intro: thin1)*)

lemma *OrdInd2-lemma*:
 $\{ \text{OrdP (Var } i), A(i ::= \text{Zero}), (\text{All } i (\text{OrdP (Var } i) \text{ IMP } A \text{ IMP } (A(i ::= \text{SUCC (Var } i)))))) \} \vdash A$

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 Reft, 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 $H \vdash \text{All } i \text{ (OrdP (Var } i \text{) IMP } A \text{ IMP } (A(i::= \text{SUCC (Var } i))))$
 shows $\text{insert (OrdP (Var } i)) H \vdash A$
 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

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

definition *hfun-sigma-ord* where

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

nominal-function *HFun-Sigma* :: $tm \Rightarrow fm$

where $\llbracket \text{atom } z \# (r, z', x, y, x', y'); \text{atom } z' \# (r, x, y, x', y');$
 $\text{atom } x \# (r, y, x', y'); \text{atom } y \# (r, x', y'); \text{atom } x' \# (r, y'); \text{atom } y' \# (r) \rrbracket$

\Longrightarrow

HFun-Sigma $r =$

$\text{All2 } z \ r \ (\text{All2 } z' \ r \ (\text{Ex } x \ (\text{Ex } y \ (\text{Ex } x' \ (\text{Ex } y' \ (\text{Var } z \ \text{EQ } \text{HPair (Var } x) \ (\text{Var } y) \ \text{AND } \text{Var } z' \ \text{EQ } \text{HPair (Var } x') \ (\text{Var } y')$
 $\text{AND } \text{OrdP (Var } x) \ \text{AND } \text{OrdP (Var } x') \ \text{AND}$
 $((\text{Var } x \ \text{EQ } \text{Var } x') \ \text{IMP } (\text{Var } y \ \text{EQ } \text{Var } y'))))))))$

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]: $a \# \text{HFun-Sigma } r \longleftrightarrow a \# r$ (is *?thesis1*)

and *eval-fm-HFun-Sigma* [simp]:

$\text{eval-fm } e \ (\text{HFun-Sigma } r) \longleftrightarrow \text{hfun-sigma-ord } \llbracket r \rrbracket e$ (is *?thesis2*)

proof –

obtain $x::\text{name}$ and $y::\text{name}$ and $z::\text{name}$ and $x'::\text{name}$ and $y'::\text{name}$ and $z'::\text{name}$

where $\text{atom } z \# (r, z', x, y, x', y')$ $\text{atom } z' \# (r, x, y, x', y')$
 $\text{atom } x \# (r, y, x', y')$ $\text{atom } y \# (r, x', y')$
 $\text{atom } x' \# (r, y')$ $\text{atom } y' \# (r)$

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 $atoms: atom\ z \# (z',x',y',x,y,a,r)$ $atom\ z' \# (x',y',x,y,a,r)$
 $atom\ x' \# (y',x,y,a,r)$ $atom\ y' \# (x,y,a,r)$
by (*metis obtain-fresh*)
hence $\{HFun-Sigma\ r, a\ IN\ r\} \vdash ?concl$ **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* :: $tm \Rightarrow tm \Rightarrow fm$
where $[[atom\ x \# (r,d,y,z); atom\ y \# (r,d,z); atom\ z \# (r,d)]] \implies$
 $HDomain-Incl\ r\ d = All2\ x\ d\ (Ex\ y\ (Ex\ z\ (Var\ z\ IN\ r\ AND\ Var\ z\ EQ\ HPair$
 $(Var\ x)\ (Var\ y))))$
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*]:
 $a \# HDomain-Incl\ r\ d \longleftrightarrow a \# r \wedge a \# d$ (**is** *?thesis1*)
and *eval-fm-HDomain-Incl* [*simp*]:
 $eval-fm\ e\ (HDomain-Incl\ r\ d) \longleftrightarrow [[d]]e \leq hdomain\ [[r]]e$ (**is** *?thesis2*)

proof –
obtain $x::name$ **and** $y::name$ **and** $z::name$
where $atom\ x \# (r,d,y,z)$ $atom\ y \# (r,d,z)$ $atom\ z \# (r,d)$
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*: $\{ HDomain-Incl\ r\ k,\ k'\ SUBS\ k \} \vdash HDomain-Incl\ r\ k'$

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*: $H \vdash HDomain-Incl\ r\ k \implies H \vdash k'\ SUBS\ k \implies H \vdash HDomain-Incl\ r\ k'$

by (*metis HDomain-Incl-Subset-lemma cut2*)

lemma *HDomain-Incl-Mem-Ord*: $H \vdash HDomain-Incl\ r\ k \implies H \vdash k'\ IN\ k \implies H \vdash OrdP\ k \implies H \vdash HDomain-Incl\ r\ k'$

by (*metis HDomain-Incl-Subset OrdP-Mem-imp-Subset*)

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

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 $\{ \} \vdash HDomain-Incl\ r\ Zero$
by (*auto simp: HDomain-Incl.simps [of x - - y z]*)
thus *?thesis*
by (*metis thin0*)
qed

lemma *HDomain-Incl-Eats*: $\{ HDomain-Incl\ r\ d \} \vdash HDomain-Incl\ (Eats\ r\ (HPair\ d\ d'))\ (SUCC\ d)$

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 ⊢ HDomain-Incl r d ⇒ H ⊢ 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) ⊢ A
           insert (a EQ d) (insert (b EQ c) H) ⊢ A
  shows   insert ((Eats (Eats Zero b) a) EQ (Eats (Eats Zero d) c)) H ⊢ A
  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} ⊢ 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} ⊢ b EQ d
  by (metis Assume thin1 Sym Trans)

```

```

lemma HSND: {HPair a b EQ HPair c d} ⊢ 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) ⊢ A
  shows   insert (HPair a b EQ HPair c d) H ⊢ 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 ⊢ HFun-Sigma r

```

and $b: H \vdash \text{HPair } a \ b \ \text{IN } r$
and $b': H \vdash \text{HPair } a \ b' \ \text{IN } r$
shows $H \vdash b \ \text{EQ } b'$
proof –
obtain $x::\text{name}$ **and** $y::\text{name}$ **and** $z::\text{name}$ **and** $x'::\text{name}$ **and** $y'::\text{name}$ **and** $z'::\text{name}$
where $\text{atoms}: \text{atom } z \ \# \ (r, a, b, b', z', x, y, x', y')$ $\text{atom } z' \ \# \ (r, a, b, b', x, y, x', y')$
 $\text{atom } x \ \# \ (r, a, b, b', y, x', y')$ $\text{atom } y \ \# \ (r, a, b, b', x', y')$
 $\text{atom } x' \ \# \ (r, a, b, b', y')$ $\text{atom } y' \ \# \ (r, a, b, b')$
by (*metis obtain-fresh*)
hence $d1: H \vdash \text{All2 } z \ r \ (\text{All2 } z' \ r \ (\text{Ex } x \ (\text{Ex } y \ (\text{Ex } x' \ (\text{Ex } y' \ (\text{Var } z \ \text{EQ } \text{HPair } (\text{Var } x) \ (\text{Var } y) \ \text{AND } \text{Var } z' \ \text{EQ } \text{HPair } (\text{Var } x') \ (\text{Var } y') \ \text{AND } \text{OrdP } (\text{Var } x) \ \text{AND } \text{OrdP } (\text{Var } x') \ \text{AND } ((\text{Var } x \ \text{EQ } \text{Var } x') \ \text{IMP } (\text{Var } y \ \text{EQ } \text{Var } y'))))))))$
using $r \ \text{HFun-Sigma.simps} \ [of \ z \ r \ z' \ x \ y \ x' \ y']$
by *simp*
have $d2: H \vdash \text{All2 } z' \ r \ (\text{Ex } x \ (\text{Ex } y \ (\text{Ex } x' \ (\text{Ex } y' \ (\text{HPair } a \ b \ \text{EQ } \text{HPair } (\text{Var } x) \ (\text{Var } y) \ \text{AND } \text{Var } z' \ \text{EQ } \text{HPair } (\text{Var } x') \ (\text{Var } y') \ \text{AND } \text{OrdP } (\text{Var } x) \ \text{AND } \text{OrdP } (\text{Var } x') \ \text{AND } ((\text{Var } x \ \text{EQ } \text{Var } x') \ \text{IMP } (\text{Var } y \ \text{EQ } \text{Var } y'))))))))$
using $\text{All-D} \ [\text{where } x = \text{HPair } a \ b, \ \text{OF } d1] \ \text{atoms}$
by *simp* (*metis MP-same b*)
have $d4: H \vdash \text{Ex } x \ (\text{Ex } y \ (\text{Ex } x' \ (\text{Ex } y' \ (\text{HPair } a \ b \ \text{EQ } \text{HPair } (\text{Var } x) \ (\text{Var } y) \ \text{AND } \text{HPair } a \ b' \ \text{EQ } \text{HPair } (\text{Var } x') \ (\text{Var } y') \ \text{AND } \text{OrdP } (\text{Var } x) \ \text{AND } \text{OrdP } (\text{Var } x') \ \text{AND } ((\text{Var } x \ \text{EQ } \text{Var } x') \ \text{IMP } (\text{Var } y \ \text{EQ } \text{Var } y'))))))))$
using $\text{All-D} \ [\text{where } x = \text{HPair } a \ b', \ \text{OF } d2] \ \text{atoms}$
by *simp* (*metis MP-same b'*)
have $d': \{ \text{Ex } x \ (\text{Ex } y \ (\text{Ex } x' \ (\text{Ex } y' \ (\text{HPair } a \ b \ \text{EQ } \text{HPair } (\text{Var } x) \ (\text{Var } y) \ \text{AND } \text{HPair } a \ b' \ \text{EQ } \text{HPair } (\text{Var } x') \ (\text{Var } y') \ \text{AND } \text{OrdP } (\text{Var } x) \ \text{AND } \text{OrdP } (\text{Var } x') \ \text{AND } ((\text{Var } x \ \text{EQ } \text{Var } x') \ \text{IMP } (\text{Var } y \ \text{EQ } \text{Var } y')))))))) \} \vdash b \ \text{EQ } b'$
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*: $\{ \text{SUCC } x \ \text{SUBS } \text{SUCC } y \} \vdash x \ \text{SUBS } y$
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 ⊢ x SUBS y*
by (*metis Assume SUCC-SUBS-lemma cut1*)

lemma *SUCC-inject: insert (SUCC x EQ SUCC y) H ⊢ 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 ⊢ A ⇒ insert (SUCC x EQ SUCC y) H ⊢ 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} ⊢ 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 ⊢ OrdP x ⇒ H ⊢ y IN x ⇒ H ⊢ 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} ⊢ 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 ⊢ OrdP x ⇒ H ⊢ SUCC y IN SUCC x ⇒ H ⊢ y IN x*
by (*rule cut2 [OF OrdP-IN-SUCC-D-lemma]*)

lemma *OrdP-IN-SUCC-Iff*: $H \vdash \text{OrdP } y \implies H \vdash \text{SUCC } x \text{ IN } \text{SUCC } y \text{ IFF } x \text{ IN } y$
by (*metis Assume Iff-I OrdP-IN-SUCC OrdP-IN-SUCC-D thin1*)

3.15 The predicate *LstSeqP*

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

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

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

lemma *LstSeq-iff*: $\text{LstSeq } s \ k \ y \iff \text{succ } k \leq \text{hdomain } s \wedge \langle k, y \rangle \in s \wedge \text{hfun-sigma-ord } s$
by (*auto simp: OrdDom-def LstSeq-def Seq-iff hfun-sigma-ord-iff*)

nominal-function *LstSeqP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$

where

$\text{LstSeqP } s \ k \ y = \text{OrdP } k \ \text{AND } \text{HDomain-Incl } s \ (\text{SUCC } k) \ \text{AND } \text{HFun-Sigma } s \ \text{AND } \text{HPair } k \ y \ \text{IN } s$

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

nominal-termination (*eqvt*)

by *lexicographic-order*

lemma

shows *LstSeqP-fresh-iff* [*simp*]:

$a \# \text{LstSeqP } s \ k \ y \iff a \# s \wedge a \# k \wedge a \# y$ (**is** *?thesis1*)

and *eval-fm-LstSeqP* [*simp*]:

$\text{eval-fm } e \ (\text{LstSeqP } s \ k \ y) \iff \text{LstSeq } \llbracket s \rrbracket e \ \llbracket k \rrbracket e \ \llbracket y \rrbracket e$ (**is** *?thesis2*)

proof –

show *?thesis1* *?thesis2*

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

qed

lemma *LstSeqP-subst* [*simp*]:

$(\text{LstSeqP } s \ k \ y)(i::t) = \text{LstSeqP } (\text{subst } i \ t \ s) \ (\text{subst } i \ t \ k) \ (\text{subst } i \ t \ y)$

by (*auto simp: fresh-Pair fresh-at-base*)

lemma *LstSeqP-E*:

assumes *insert* (*HDomain-Incl* $s \ (\text{SUCC } k)$)

(insert (*OrdP* k) (*insert* (*HFun-Sigma* s)

(insert (*HPair* $k \ y \ \text{IN } s$) 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 $H \vdash s \text{ EQ } s' \ H \vdash k \text{ EQ } k' \ H \vdash y \text{ EQ } y'$
shows $H \vdash \text{LstSeqP } s \ k \ y \text{ IFF } \text{LstSeqP } s' \ k' \ y'$
by (*rule P3-cong [OF - assms], auto*)

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

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

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

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

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

lemma *LstSeqP-EQ*: $\llbracket H \vdash \text{LstSeqP } s \ k \ y; H \vdash \text{HPair } k \ y' \text{ IN } s \rrbracket \implies H \vdash y \text{ EQ } y'$
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  $\Rightarrow$  atom set  $\Rightarrow$  bool
  where ground-aux t S  $\equiv$  (supp t  $\subseteq$  S)
```

```
abbreviation ground :: tm  $\Rightarrow$  bool
  where ground t  $\equiv$  ground-aux t {}
```

```
definition ground-fm-aux :: fm  $\Rightarrow$  atom set  $\Rightarrow$  bool
  where ground-fm-aux A S  $\equiv$  (supp A  $\subseteq$  S)
```

```
abbreviation ground-fm :: fm  $\Rightarrow$  bool
  where ground-fm A  $\equiv$  ground-fm-aux A {}
```

```
lemma ground-aux-simps[simp]:
  ground-aux Zero S = True
  ground-aux (Var k) S = (if atom k  $\in$  S then True else False)
  ground-aux (Eats t u) S = (ground-aux t S  $\wedge$  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  $\wedge$  ground-aux u S)
  ground-fm-aux (t EQ u) S = (ground-aux t S  $\wedge$  ground-aux u S)
  ground-fm-aux (A OR B) S = (ground-fm-aux A S  $\wedge$  ground-fm-aux B S)
  ground-fm-aux (A AND B) S = (ground-fm-aux A S  $\wedge$  ground-fm-aux B S)
```

$ground\text{-}fm\text{-}aux (A \text{ IFF } B) S = (ground\text{-}fm\text{-}aux A S \wedge ground\text{-}fm\text{-}aux B S)$
 $ground\text{-}fm\text{-}aux (Neg A) S = (ground\text{-}fm\text{-}aux A S)$
 $ground\text{-}fm\text{-}aux (Ex x A) S = (ground\text{-}fm\text{-}aux A (S \cup \{atom\ x\}))$
by (*auto simp: ground-fm-aux-def ground-aux-def supp-conv-fresh*)

lemma *ground-fresh[simp]:*
 $ground\ t \implies atom\ i \not\# t$
 $ground\text{-}fm\ A \implies atom\ i \not\# A$
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* \Rightarrow *bool* **where**
 $MemI: ss\text{-}fm (Var\ i\ IN\ Var\ j)$
 $| DisjI: ss\text{-}fm\ A \implies ss\text{-}fm\ B \implies ss\text{-}fm (A\ OR\ B)$
 $| ConjI: ss\text{-}fm\ A \implies ss\text{-}fm\ B \implies ss\text{-}fm (A\ AND\ B)$
 $| ExI: ss\text{-}fm\ A \implies ss\text{-}fm (Ex\ i\ A)$
 $| All2I: ss\text{-}fm\ A \implies atom\ j \not\# (i, A) \implies ss\text{-}fm (All2\ i\ (Var\ j)\ A)$

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* \Rightarrow *bool*
where $Sigma\text{-}fm\ A \iff (\exists B. ss\text{-}fm\ B \wedge supp\ B \subseteq supp\ A \wedge \{\} \vdash A \text{ IFF } B)$

lemma *Sigma-fm-Iff*: $\{\} \vdash B \text{ IFF } A; supp\ A \subseteq supp\ B; Sigma\text{-}fm\ A \implies Sigma\text{-}fm\ B$
by (*metis Sigma-fm-def Iff-trans order-trans*)

lemma *ss-fm-imp-Sigma-fm* [*intro*]: $ss\text{-}fm\ A \implies Sigma\text{-}fm\ A$
by (*metis Iff-refl Sigma-fm-def order-refl*)

lemma *Sigma-fm-Fls* [*iff*]: $Sigma\text{-}fm\ Fls$
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 {} ⊢ A IFF SA supp SA ⊆ supp A*
 and *ss-fm SB {} ⊢ B IFF SB supp SB ⊆ 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.support(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 # (i,A)*
 shows *Sigma-fm (All2 i (Var j) A)*

proof –

obtain *SA* where *SA: ss-fm SA {} ⊢ A IFF SA supp SA ⊆ 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 # (z,x,y)*
 shows $\{\} \vdash z \text{ EQ Eats } x \ y \text{ IFF } (All2 \ i \ z \ (Var \ i \ IN \ x \ OR \ Var \ i \ EQ \ y)) \ AND \ x \ SUBS \ z \ AND \ y \ IN \ z$

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 $\bigwedge j::name. \text{Sigma-fm } (Var\ j\ IN\ t)$

and $\bigwedge k::name. \text{Sigma-fm } (Var\ k\ EQ\ u)$

shows $\text{Sigma-fm } (Var\ i\ SUBS\ Eats\ t\ u)$

proof –

obtain $k::name$ **where** $k: \text{atom } k \# (t, u, Var\ i)$

by (*metis obtain-fresh*)

hence $\{\} \vdash Var\ i\ SUBS\ Eats\ t\ u\ IFF\ All2\ k\ (Var\ i)\ (Var\ k\ IN\ t\ OR\ Var\ k\ EQ\ u)$

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 $\bigwedge j::name. \text{Sigma-fm } (Var\ j\ EQ\ t)$

and $\bigwedge k::name. \text{Sigma-fm } (Var\ k\ EQ\ u)$

shows $\text{Sigma-fm } (Var\ i\ EQ\ Eats\ t\ u)$

proof –

obtain $j::name$ **and** $k::name$ **and** $l::name$

where $\text{atoms: } \text{atom } j \# (t, u, i)\ \text{atom } k \# (t, u, i, j)\ \text{atom } l \# (t, u, i, j, k)$

by (*metis obtain-fresh*)

hence $\{\} \vdash Var\ i\ EQ\ Eats\ t\ u\ IFF$

$Ex\ j\ (Ex\ k\ (Var\ i\ EQ\ Eats\ (Var\ j)\ (Var\ k)\ AND\ Var\ j\ EQ\ t\ AND\ Var\ k\ EQ\ u))$

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  $\bigwedge j::name. \text{Sigma-fm } (Var\ j\ EQ\ t)$ 
and  $\bigwedge k::name. \text{Sigma-fm } (Var\ k\ EQ\ u)$ 
shows  $\text{Sigma-fm } (Eats\ t\ u\ IN\ Var\ i)$ 
proof -
obtain  $j::name$  where  $j: atom\ j\ \#(t, u, Var\ i)$ 
by (metis obtain-fresh)
hence  $\{\} \vdash Eats\ t\ u\ IN\ Var\ i\ IFF$ 
 $Ex\ j\ (Var\ j\ IN\ Var\ i\ AND\ Var\ j\ EQ\ Eats\ t\ u)$ 
apply (auto simp: fresh-Pair intro: Ex-I [where  $x=Eats\ t\ u$ ])
apply (metis Assume Mem-cong [OF - Reft, 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*:

$size\ t + size\ u < n \implies \text{Sigma-fm } (t\ SUBS\ u) \wedge \text{Sigma-fm } (t\ IN\ u)$

proof (induction n arbitrary: $t\ u$ rule: less-induct)

case (less $n\ t\ u$)

show ?case

proof

show $\text{Sigma-fm } (t\ SUBS\ u)$

proof (cases t rule: tm.exhaust)

case Zero thus ?thesis

by (auto intro: theorem-sf)

next

case (Var i) thus ?thesis using less.prem

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.prem

by (auto intro!: Sigma-fm-Iff [OF Eats-Subset-Iff] simp: supp-conv-fresh)
(metis add commute)

qed

next

show $\text{Sigma-fm } (t\ IN\ u)$

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 $\langle u = Var\ i \rangle$

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.prem
    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.prem
  by (force intro: Sigma-fm-Iff [OF Mem-Eats-Iff] Sigma-fm-Iff [OF Extensionality - -]
      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

$\{\} \vdash \text{HFun-Sigma } r \text{ IFF}$

$\text{All2 } z \ r \ (\text{All2 } z' \ r \ (\text{Ex } x \ (\text{Ex } y \ (\text{Ex } x' \ (\text{Ex } y' \ (\text{Var } z \ \text{EQ } \text{HPair} \ (\text{Var } x) \ (\text{Var } y) \ \text{AND } \text{Var } z' \ \text{EQ } \text{HPair} \ (\text{Var } x') \ (\text{Var } y')$

$y')$

$\text{AND } \text{OrdP} \ (\text{Var } x) \ \text{AND } \text{OrdP} \ (\text{Var } x') \ \text{AND} \\ ((\text{Var } x \ \text{NEQ } \text{Var } x') \ \text{OR} \ (\text{Var } y \ \text{EQ } \text{Var } y'))))))))$

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 # (t,z',x,y,x',y') atom z' # (t,x,y,x',y')
           atom x # (t,y,x',y') atom y # (t,x',y')
           atom x' # (t,y') atom y' # (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 # 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'*:

$eval\text{-}fm\ e\ (Ex\ k\ A) \longleftrightarrow (\exists t.\ eval\text{-}fm\ e\ (A(k::=t)))$

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*:

$atom\ i\ \sharp\ (t,u) \Longrightarrow$

$eval\text{-}fm\ e\ (All2\ i\ (Eats\ t\ u)\ A) \longleftrightarrow eval\text{-}fm\ e\ (A(i::=u)) \wedge eval\text{-}fm\ e\ (All2\ i\ t\ A)$

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*:

$ground\ t \Longrightarrow eval\text{-}fm\ e\ (All2\ i\ t\ A) \longleftrightarrow (\forall u \in elts\ t.\ eval\text{-}fm\ e\ (A(i::=u)))$

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*:

$\llbracket \text{size } t + \text{size } u < n; \text{ground } t; \text{ground } u \rrbracket$
 $\implies (\llbracket t \rrbracket e \leq \llbracket u \rrbracket e \longrightarrow \{\} \vdash t \text{ SUBS } u) \wedge (\llbracket t \rrbracket e \in \llbracket u \rrbracket e \longrightarrow \{\} \vdash t \text{ IN } u)$
proof (*induction n arbitrary: t u rule: less-induct*)
case (*less n t u*)
show *?case*
proof
show $\llbracket t \rrbracket e \leq \llbracket u \rrbracket e \longrightarrow \{\} \vdash t \text{ SUBS } u$ **using** *less*
by (*cases t rule: tm.exhaust*) *auto*
next
 { **fix** *y t u*
have $\llbracket y < n; \text{size } t + \text{size } u < y; \text{ground } t; \text{ground } u; \llbracket t \rrbracket e = \llbracket u \rrbracket e \rrbracket$
 $\implies \{\} \vdash t \text{ EQ } u$
by (*metis Equality-I less.IH add commute order-refl*)
 }
thus $\llbracket t \rrbracket e \in \llbracket u \rrbracket e \longrightarrow \{\} \vdash t \text{ IN } u$ **using** *less.prem*s
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*: $\llbracket t \rrbracket e \leq \llbracket u \rrbracket e \implies \{\} \vdash t \text{ SUBS } u$
and *ground-prove-IN*: $\llbracket t \rrbracket e \in \llbracket u \rrbracket e \implies \{\} \vdash t \text{ IN } u$
and *ground-prove-EQ*: $\llbracket t \rrbracket e = \llbracket u \rrbracket e \implies \{\} \vdash t \text{ EQ } u$
by (*metis Equality-I assms ground-prove [OF lessI] order-refl*)**+**

lemma *ground-subst*:

ground-aux tm (insert (atom i) S) \implies ground t \implies 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) \implies ground t \implies 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*: $u \in \text{elts } t \implies \text{ground-aux } t \implies \text{ground-aux } u \text{ } S$

by (*induct t rule: tm.induct*) *auto*

lemma *ground-se-fm-induction*:

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

```

case (less n  $\alpha$ )
show ?case using  $\langle$ se-fm  $\alpha$  $\rangle$ 
proof (cases rule: se-fm.cases)
  case (MemI t u) thus  $\{\}$   $\vdash$   $\alpha$  using less
    by (auto intro: ground-prove-IN)
next
  case (DisjI A B) thus  $\{\}$   $\vdash$   $\alpha$  using less
    by (auto intro: Disj-I1 Disj-I2)
next
  case (ConjI A B) thus  $\{\}$   $\vdash$   $\alpha$  using less
    by auto
next
  case (ExI A i)
  thus  $\{\}$   $\vdash$   $\alpha$  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  $(\forall u \in \text{elts } t. \text{eval-fm } e (A(i::=u)))$ 
    by (metis All2I(1) t eval-fm-All2-Iff-elts less(5))
  thus  $\{\}$   $\vdash$   $\alpha$  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:  $ss\text{-fm } A \implies se\text{-fm } A$ 
  by (erule ss-fm.induct) auto

lemma se-fm-imp-thm:  $\llbracket se\text{-fm } A; \text{ground-fm } A; \text{eval-fm } e \ A \rrbracket \implies \{\} \vdash A$ 
  by (metis ground-se-fm-induction lessI)

  Theorem 2.5

theorem Sigma-fm-imp-thm:  $\llbracket \text{Sigma-fm } A; \text{ground-fm } A; \text{eval-fm } e \ 0 \ A \rrbracket \implies \{\} \vdash A$ 
  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  $\Rightarrow$  bool where is-Var x  $\equiv$  Ord x  $\wedge$  0  $\in$  x
```

```
definition VarP :: tm  $\Rightarrow$  fm where VarP x  $\equiv$  OrdP x AND Zero IN x
```

```
lemma VarP-eqvt [eqvt]: (p  $\cdot$  VarP x) = VarP (p  $\cdot$  x)
by (simp add: VarP-def)
```

```
lemma VarP-fresh-iff [simp]: a  $\#$  VarP x  $\longleftrightarrow$  a  $\#$  x
by (simp add: VarP-def)
```

```
lemma eval-fm-VarP [simp]: eval-fm e (VarP x)  $\longleftrightarrow$  is-Var  $\llbracket$ x $\rrbracket$ 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*: $H \vdash x \text{ EQ } x' \implies H \vdash \text{VarP } x \text{ IFF } \text{VarP } x'$
by (*rule P1-cong*) *auto*

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

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

lemma *is-Var-q-Var* [*iff*]: $\text{is-Var } (q\text{-Var } i)$
by (*simp add: q-Var-def*)

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

lemma *decode-Var-q-Var* [*simp*]: $\text{decode-Var } (q\text{-Var } i) = i$
by (*simp add: decode-Var-def q-Var-def*)

lemma *is-Var-imp-decode-Var*: $\text{is-Var } x \implies x = \llbracket \text{Var } (\text{decode-Var } x) \rrbracket \ e$
by (*simp add: is-Var-def quot-Var decode-Var-def*) (*metis empty-iff succ-pred*)

lemma *is-Var-iff*: $\text{is-Var } v \longleftrightarrow v = \text{succ } (\text{ord-of } (\text{nat-of-name } (\text{decode-Var } v)))$
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*]: $\text{is-Var } v \implies \text{is-Var } v' \implies \text{decode-Var } v = \text{decode-Var } v' \longleftrightarrow v = v'$
by (*metis is-Var-iff*)

5.1.2 De Bruijn Indexes

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

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

nominal-function *IndP* :: $tm \Rightarrow fm$
where *atom* $m \ \# \ x \implies$
 $\text{IndP } x = \text{Ex } m \ (\text{OrdP } (\text{Var } m) \ \text{AND } x \ \text{EQ } \text{HPair } (\text{HTuple } 6) \ (\text{Var } m))$
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*]: $a \ \# \ \text{IndP } x \longleftrightarrow a \ \# \ x$ (*is ?thesis1*)
and *eval-fm-IndP* [*simp*]: $\text{eval-fm } e \ (\text{IndP } x) \longleftrightarrow \text{is-Ind } \llbracket x \rrbracket \ e$ (*is ?thesis2*)
and *IndP-sf* [*iff*]: $\text{Sigma-fm } (\text{IndP } x)$ (*is ?thsf*)

and *OrdP-IndP-Q-Ind*: $\{OrdP\ x\} \vdash IndP\ (Q-Ind\ x)$ (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*: $H \vdash OrdP\ x \implies H \vdash IndP\ (Q-Ind\ x)$
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*: $H \vdash x\ EQ\ x' \implies H \vdash IndP\ x\ IFF\ IndP\ x'$
by (*rule P1-cong auto*)

definition *decode-Ind* :: $hf \Rightarrow nat$
where *decode-Ind* $x \equiv nat-of-ord\ (hsnd\ x)$

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

5.1.3 Various syntactic lemmas

lemma *eval-Var-q*: $\llbracket [Var\ i] \rrbracket e = q-Var\ i$
by (*simp add: quot-tm-def q-Var-def*)

lemma *is-Var-eval-Var* [*simp*]: $is-Var\ \llbracket [Var\ i] \rrbracket e$
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* :: $bool \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$
where *SeqCTerm* $vf\ s\ k\ t \equiv BuildSeq\ (\lambda u. u=0 \vee vf \wedge is-Var\ u)\ (\lambda u\ v\ w. u = q-Eats\ v\ w)\ s\ k\ t$

nominal-function *SeqCTermP* :: $bool \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $\llbracket atom\ l\ \# (s,k,sl,m,n,sm,sn); atom\ sl\ \# (s,m,n,sm,sn);$
 $atom\ m\ \# (s,n,sm,sn); atom\ n\ \# (s,sm,sn);$
 $atom\ sm\ \# (s,sn); atom\ sn\ \# (s) \rrbracket \implies$
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*]:

$a \# SeqCTermP\ vf\ s\ k\ t \longleftrightarrow a \# s \wedge a \# k \wedge a \# t$ (**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 # (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)*

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 # (j,w,s,k,sl,m,n,sm,sn)* *atom sl # (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* \Rightarrow *hf* \Rightarrow *bool*
 where *CTerm* *vf t* \equiv (\exists *s k*. *SeqCTerm* *vf s k t*)

nominal-function *CTermP* :: *bool* \Rightarrow *tm* \Rightarrow *fm*
 where $\llbracket \text{atom } k \# (s,t); \text{atom } s \# t \rrbracket \Longrightarrow$
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*]: $a \# \text{CTermP } vf \ t \longleftrightarrow a \# t$ (is *?thesis1*)
and *eval-fm-CTermP* [*simp*]: $\text{eval-fm } e \ (\text{CTermP } vf \ t) \longleftrightarrow \text{CTerm } vf \ \llbracket t \rrbracket e$ (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* \Rightarrow *bool*
 where *Term* \equiv *CTerm* *True*

abbreviation *TermP* :: *tm* \Rightarrow *fm*
 where *TermP* \equiv *CTermP* *True*

abbreviation *Const* :: *hf* \Rightarrow *bool*
 where *Const* \equiv *CTerm* *False*

abbreviation *ConstP* :: *tm* \Rightarrow *fm*
 where *ConstP* \equiv *CTermP* *False*

5.3.2 Correctness: It Corresponds to Quotations of Real Terms

lemma *wf-Term-quot-dbtm* [*simp*]: *wf-dbtm u* \Longrightarrow *Term* $\llbracket \text{quot-dbtm } u \rrbracket e$

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** $Term \llbracket t \rrbracket e$
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 $\exists t::dbtm. wf-dbtm t \wedge x = \llbracket quot-dbtm t \rrbracket e$
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 $wf-dbtm tm1 y = \llbracket quot-dbtm tm1 \rrbracket e$
 $wf-dbtm tm2 z = \llbracket quot-dbtm tm2 \rrbracket e$
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** $wf-dbtm t x = \llbracket quot-dbtm t \rrbracket e$
by (*metis assms SeqCTerm-imp-wf-dbtm CTerm-def*)

corollary *Term-imp-is-tm*: **assumes** $Term x$ **obtains** $t::tm$ **where** $x = \llbracket t \rrbracket e$
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** $x = \llbracket quot-dbtm t \rrbracket e$ $y = \llbracket quot-dbtm u \rrbracket e$
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*: $Const x \implies Term x$
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} ⊢ 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 (λy y'. (is-Ind y ∨ Ord y) ∧ y' = (if y=v then u else y))

(λu 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

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'))))))))))))

apply (*simp-all add: eqvt-def SeqStTermP-graph-aux-def flip-fresh-fresh*)

by *auto (metis obtain-fresh)*

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

lemma

shows *SeqStTermP-fresh-iff* [*simp*]:

$a \# \text{SeqStTermP } v \ i \ t \ u \ s \ k \longleftrightarrow a \# v \wedge a \# i \wedge a \# t \wedge a \# u \wedge a \# s \wedge a \# k$

(**is** *?thesis1*)

and *eval-fm-SeqStTermP* [*simp*]:

$\text{eval-fm } e \ (\text{SeqStTermP } v \ i \ t \ u \ s \ k) \longleftrightarrow \text{SeqStTerm } \llbracket v \rrbracket e \ \llbracket i \rrbracket e \ \llbracket t \rrbracket e \ \llbracket u \rrbracket e \ \llbracket s \rrbracket e \ \llbracket k \rrbracket e$ (**is** *?thesis2*)

and *SeqStTermP-sf* [*iff*]:

$\text{Sigma-fm } (\text{SeqStTermP } v \ i \ t \ u \ s \ k)$ (**is** *?thsf*)

and *SeqStTermP-imp-OrdP*:

$\{ \text{SeqStTermP } v \ i \ t \ u \ s \ k \} \vdash \text{OrdP } k$ (**is** *?thord*)

and *SeqStTermP-imp-VarP*:

$\{ \text{SeqStTermP } v \ i \ t \ u \ s \ k \} \vdash \text{VarP } v$ (**is** *?thvar*)

and *SeqStTermP-imp-LstSeqP*:

$\{ \text{SeqStTermP } v \ i \ t \ u \ s \ k \} \vdash \text{LstSeqP } s \ k \ (\text{HPair } t \ u)$ (**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*:

$\text{atom } l \# (s, k, v, i, sl, sl', m, n, sm, sm', sn, sn')$

$\text{atom } sl \# (s, v, i, sl', m, n, sm, sm', sn, sn') \text{ atom } sl' \# (s, v, i, m, n, sm, sm', sn, sn')$

$\text{atom } m \# (s, n, sm, sm', sn, sn') \text{ atom } n \# (s, sm, sm', sn, sn')$

$\text{atom } sm \# (s, sm', sn, sn') \text{ atom } sm' \# (s, sn, sn')$

$\text{atom } sn \# (s, sn') \text{ atom } sn' \# (s)$

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 $\llbracket s \rrbracket e$, OF LstSeq-imp-Seq-succ]

Ord-trans [of - - succ $\llbracket k \rrbracket e$]

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*]:

$(\text{SeqStTermP } v \ i \ t \ u \ s \ k)(j::=w) =$

$\text{SeqStTermP } (\text{subst } j \ w \ v) \ (\text{subst } j \ w \ i) \ (\text{subst } j \ w \ t) \ (\text{subst } j \ w \ u) \ (\text{subst } j \ w \ s) \ (\text{subst } j \ w \ k)$

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*:
 $\llbracket H \vdash t\ EQ\ t'; H \vdash u\ EQ\ u'; H \vdash s\ EQ\ s'; H \vdash k\ EQ\ k' \rrbracket$
 $\implies H \vdash SeqStTermP\ v\ i\ t\ u\ s\ k\ IFF\ SeqStTermP\ v\ i\ t'\ u'\ s'\ k'$
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* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $\llbracket atom\ s \# (v,i,t,u,k); atom\ k \# (v,i,t,u) \rrbracket \implies$
 $AbstTermP\ v\ i\ t\ u =$
 $OrdP\ i\ AND\ Ex\ s\ (Ex\ k\ (SeqStTermP\ v\ (Q-Ind\ i)\ t\ u\ (Var\ s)\ (Var\ k)))$
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*]:
 $a \# AbstTermP\ v\ i\ t\ u \iff a \# v \wedge a \# i \wedge a \# t \wedge a \# u$ (**is** *?thesis1*)
and *eval-fm-AbstTermP* [*simp*]:
 $eval-fm\ e\ (AbstTermP\ v\ i\ t\ u) \iff AbstTerm\ \llbracket v \rrbracket e\ \llbracket i \rrbracket e\ \llbracket t \rrbracket e\ \llbracket u \rrbracket e$ (**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  $\implies$   $\neg$  is-Ind v
  by (auto simp: is-Var-def is-Ind-def)

```

```

lemma AbstTerm-imp-abst-dbtm:
  assumes AbstTerm v i x x'
  shows  $\exists t. x = \llbracket \text{quot-dbtm } t \rrbracket e \wedge$ 
         $x' = \llbracket \text{quot-dbtm } (\text{abst-dbtm } (\text{decode-Var } v) (\text{nat-of-ord } i) t) \rrbracket e$ 

```

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 =  $\llbracket \text{quot-dbtm } tm1 \rrbracket e$ 

```

```

            y' =  $\llbracket \text{quot-dbtm } (\text{abst-dbtm } (\text{decode-Var } v) (\text{nat-of-ord } i) tm1) \rrbracket e$ 

```

```

            z =  $\llbracket \text{quot-dbtm } tm2 \rrbracket e$ 

```

```

            z' =  $\llbracket \text{quot-dbtm } (\text{abst-dbtm } (\text{decode-Var } v) (\text{nat-of-ord } i) tm2) \rrbracket e$ 

```

```

      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*:

$AbstTerm (q-Var i) (ord-of n) \llbracket quot-dbtm t \rrbracket e$
 $\llbracket quot-dbtm (abst-dbtm i n t) \rrbracket e$

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* :: $hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$

where $SubstTerm v u x x' \equiv Term u \wedge (\exists s k. SeqStTerm v u x x' s k)$

5.5.1 Defining the syntax

nominal-function *SubstTermP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$

where $\llbracket atom s \# (v,i,t,u,k); atom k \# (v,i,t,u) \rrbracket \Longrightarrow$

$SubstTermP v i t u = TermP i AND Ex s (Ex k (SeqStTermP v i t u (Var s)$
 $(Var k)))$

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*]:

$a \# SubstTermP v i t u \longleftrightarrow a \# v \wedge a \# i \wedge a \# t \wedge a \# u$ (**is** *?thesis1*)

and *eval-fm-SubstTermP* [*simp*]:

$eval-fm e (SubstTermP v i t u) \longleftrightarrow SubstTerm \llbracket v \rrbracket e \llbracket i \rrbracket e \llbracket t \rrbracket e \llbracket u \rrbracket e$ (**is** *?thesis2*)

and *SubstTermP-sf* [*iff*]:

$Sigma-fm (SubstTermP v i t u)$ (**is** *?thsf*)

and *SubstTermP-imp-TermP*:

$\{ SubstTermP v i t u \} \vdash TermP i$ (**is** *?thterm*)

and *SubstTermP-imp-VarP*:

$\{ SubstTermP v i t u \} \vdash VarP v$ (**is** *?thvar*)

proof –

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*:
 $\llbracket H \vdash v \text{EQ } v'; H \vdash i \text{EQ } i'; H \vdash t \text{EQ } t'; H \vdash u \text{EQ } u' \rrbracket$
 $\implies H \vdash \text{SubstTermP } v \ i \ t \ u \ \text{IFF } \text{SubstTermP } v' \ i' \ t' \ u'$
 by (*rule P4-cong*) *auto*

declare *SubstTermP.simps [simp del]*

lemma *SubstTerm-imp-subst-dbtm*:
 assumes *SubstTerm* *v* $\llbracket \text{quot-dbtm } u \rrbracket e \ x \ x'$
 shows $\exists t. x = \llbracket \text{quot-dbtm } t \rrbracket e \wedge$
 $x' = \llbracket \text{quot-dbtm } (\text{subst-dbtm } u \ (\text{decode-Var } v) \ t) \rrbracket e$

proof –

obtain *s k* **where** *v*: *is-Var* *v* **and** *u*: *Term* $\llbracket \text{quot-dbtm } u \rrbracket e$
and *sk*: *SeqStTerm* *v* $\llbracket \text{quot-dbtm } u \rrbracket e \ x \ x' \ s \ k$
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* = $\llbracket \text{quot-dbtm } tm1 \rrbracket e$

$y' = \llbracket \text{quot-dbtm } (\text{subst-dbtm } u \ (\text{decode-Var } v) \ tm1) \rrbracket e$

$z = \llbracket \text{quot-dbtm } tm2 \rrbracket e$

$z' = \llbracket \text{quot-dbtm } (\text{subst-dbtm } u \ (\text{decode-Var } v) \ tm2) \rrbracket e$

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

lemma *SubstTerm-subst-dbtm*:
assumes *Term* $\llbracket \text{quot-dbtm } u \rrbracket e$
shows *SubstTerm* $(q\text{-Var } v) \llbracket \text{quot-dbtm } u \rrbracket e \llbracket \text{quot-dbtm } t \rrbracket e \llbracket \text{quot-dbtm } (\text{subst-dbtm } u v t) \rrbracket e$
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* $v i y y' \equiv$
 $(\exists t u t' u'. \text{AbstTerm } v i t t' \wedge \text{AbstTerm } v i u u' \wedge$
 $((y = q\text{-Eq } t u \wedge y' = q\text{-Eq } t' u') \vee (y = q\text{-Mem } t u \wedge y' = q\text{-Mem } t'$
 $u')))$

nominal-function *AbstAtomicP* $:: tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $\llbracket \text{atom } t \# (v, i, y, y', t', u, u'); \text{atom } t' \# (v, i, y, y', u, u');$
 $\text{atom } u \# (v, i, y, y', u'); \text{atom } u' \# (v, i, y, y') \rrbracket \Longrightarrow$
 $\text{AbstAtomicP } v i y y' =$
 $\text{Ex } t (\text{Ex } u (\text{Ex } t' (\text{Ex } u'$
 $(\text{AbstTermP } v i (\text{Var } t) (\text{Var } t') \text{ AND } \text{AbstTermP } v i (\text{Var } u) (\text{Var } u')$
 AND
 $((y \text{ EQ } Q\text{-Eq } (\text{Var } t) (\text{Var } u) \text{ AND } y' \text{ EQ } Q\text{-Eq } (\text{Var } t') (\text{Var}$
 $u')) \text{ OR}$
 $(y \text{ EQ } Q\text{-Mem } (\text{Var } t) (\text{Var } u) \text{ AND } y' \text{ EQ } Q\text{-Mem } (\text{Var } t')$
 $(\text{Var } u'))))))))$
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*]:
 $a \# \text{AbstAtomicP } v i y y' \longleftrightarrow a \# v \wedge a \# i \wedge a \# y \wedge a \# y'$ (**is**
?thesis1)
and *eval-fm-AbstAtomicP* [*simp*]:
 $\text{eval-fm } e (\text{AbstAtomicP } v i y y') \longleftrightarrow \text{AbstAtomic } \llbracket v \rrbracket e \llbracket i \rrbracket e \llbracket y \rrbracket e \llbracket y' \rrbracket e$ (**is**
?thesis2)
and *AbstAtomicP-sf* [*iff*]: *Sigma-fm* $(\text{AbstAtomicP } v i y y')$ (**is** *?thsf*)
proof –
obtain *t::name* **and** *u::name* **and** *t'::name* **and** *u'::name*

```

where atom t # (v,i,y,y',t',u,u') atom t' # (v,i,y,y',u,u')
        atom u # (v,i,y,y',u') atom u' # (v,i,y,y')
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 # (v,tm,y,y',w,i,t',u,u') atom t' # (v,tm,y,y',w,i,u,u')
        atom u # (v,tm,y,y',w,i,u') atom u' # (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 *AbsMakeForm* :: hf ⇒ hf ⇒ hf ⇒ hf ⇒ hf ⇒ hf ⇒ hf ⇒ hf ⇒ hf ⇒ hf ⇒ bool

where *AbsMakeForm* 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) *AbsMakeForm* s k i x x'

nominal-function *SeqAbstFormP* :: tm ⇒ tm ⇒ tm ⇒ tm ⇒ tm ⇒ tm ⇒ fm

where \llbracket atom l # (s,k,v,sl,sl',m,n,smi,sm,sm',sni,sn,sn');
 atom sl # (s,v,sl,sl',m,n,smi,sm,sm',sni,sn,sn');
 atom sl' # (s,v,sl',m,n,smi,sm,sm',sni,sn,sn');
 atom sl'' # (s,v,m,n,smi,sm,sm',sni,sn,sn');
 atom m # (s,n,smi,sm,sm',sni,sn,sn');
 atom n # (s,smi,sm,sm',sni,sn,sn'); atom smi # (s,sm,sm',sni,sn,sn');
 atom sm # (s,sm',sni,sn,sn'); atom sm' # (s,sni,sn,sn');
 atom sni # (s,sn,sn'); atom sn # (s,sn'); atom sn' # (s) $\rrbracket \implies$
SeqAbstFormP v i x x' s k =
 LstSeqP s k (HPair i (HPair x x')) AND
 All2 l (SUCC k) (Ex sl (Ex sl' (HPair (Var l) (HPair (Var sl) (HPair (Var sl) (Var sl')))) IN s AND
 (AbstAtomicP v (Var sl) (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*]:

$a \# SeqAbstFormP v i x x' s k \longleftrightarrow a \# v \wedge a \# i \wedge a \# x \wedge a \# x' \wedge a \# s \wedge$
 $a \# k$ (**is** *?thesis1*)

and *eval-fm-SeqAbstFormP* [*simp*]:

$eval-fm e (SeqAbstFormP v i x x' s k) \longleftrightarrow SeqAbstForm \llbracket v \rrbracket e \llbracket i \rrbracket e \llbracket x \rrbracket e \llbracket x' \rrbracket e$
 $\llbracket s \rrbracket e \llbracket k \rrbracket e$ (**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 $\# (s, k, v, sli, sl, sl', m, n, smi, sm, sm', sni, sn, sn')$

atom sli $\# (s, v, sl, sl', m, n, smi, sm, sm', sni, sn, sn')$

atom sl $\# (s, v, sl', m, n, smi, sm, sm', sni, sn, sn')$

atom sl' $\# (s, v, m, n, smi, sm, sm', sni, sn, sn')$

atom m $\# (s, n, smi, sm, sm', sni, sn, sn')$ *atom n* $\# (s, smi, sm, sm', sni, sn, sn')$

atom smi $\# (s, sm, sm', sni, sn, sn')$

atom sm $\# (s, sm', sni, sn, sn')$

atom sm' $\# (s, sni, sn, sn')$

atom sni $\# (s, sn, sn')$ *atom sn* $\# (s, sn')$ *atom sn'* $\# s$

by (*metis obtain-fresh*)

thus *?thesis1* *?thsf*

by (*auto intro: LstSeqP-OrdP*)


```

show ?thesis2 using atoms
unfolding SeqAbstForm-def BuildSeq3-def BuildSeq-def Builds-def
  HBall-def HBex-def q-defs AbstMakeForm-def
by (force simp add: LstSeq-imp-Ord Ord-trans [of - - succ  $\llbracket k \rrbracket e$ ]
  Seq-iff-app [of  $\llbracket s \rrbracket e$ , OF LstSeq-imp-Seq-succ]
  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  $\Rightarrow$  hf  $\Rightarrow$  hf  $\Rightarrow$  hf  $\Rightarrow$  bool
  where AbstForm v i x x'  $\equiv$  is-Var v  $\wedge$  Ord i  $\wedge$  ( $\exists$  s k. SeqAbstForm v i x x' s k)

nominal-function AbstFormP :: tm  $\Rightarrow$  tm  $\Rightarrow$  tm  $\Rightarrow$  tm  $\Rightarrow$  fm
  where  $\llbracket$ atom s  $\#$  (v,i,x,x',k);
  atom k  $\#$  (v,i,x,x')  $\rrbracket \Longrightarrow$ 
  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*]:
 $a \# \text{AbstFormP } v \ i \ x \ x' \longleftrightarrow a \# v \wedge a \# i \wedge a \# x \wedge a \# x'$ (**is** *?thesis1*)
and *eval-fm-AbstFormP* [*simp*]:
 $\text{eval-fm } e \ (\text{AbstFormP } v \ i \ x \ x') \longleftrightarrow \text{AbstForm } \llbracket v \rrbracket e \ \llbracket i \rrbracket e \ \llbracket x \rrbracket e \ \llbracket x' \rrbracket e$ (**is** *?thesis2*)
and *AbstFormP-sf* [*iff*]:
 $\text{Sigma-fm } (\text{AbstFormP } v \ i \ x \ x')$ (**is** *?thsf*)
proof –
obtain *s::name* **and** *k::name* **where** $\text{atom } s \# (v, i, x, x', k)$ $\text{atom } k \# (v, i, x, x')$
by (*metis obtain-fresh*)
thus *?thesis1* *?thesis2* *?thsf*
by (*auto simp: AbstForm-def*)
qed

lemma *AbstFormP-subst* [*simp*]:
 $(\text{AbstFormP } v \ i \ x \ x')(j::=t) = \text{AbstFormP } (\text{subst } j \ t \ v) \ (\text{subst } j \ t \ i) \ (\text{subst } j \ t \ x) \ (\text{subst } j \ t \ x')$
proof –
obtain *s::name* **and** *k::name* **where** $\text{atom } s \# (v, i, x, x', t, j, k)$ $\text{atom } k \# (v, i, x, x', t, j)$
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*: $\text{AbstForm } v \ u \ x \ x' \implies \text{Ord } v$
by (*metis AbstForm-def is-Var-def*)

lemma *AbstForm-imp-abst-dbfm*:
assumes $\text{AbstForm } v \ i \ x \ x'$
shows $\exists A. x = \llbracket \text{quot-dbfm } A \rrbracket e \wedge x' = \llbracket \text{quot-dbfm } (\text{abst-dbfm } (\text{decode-Var } v) \ (\text{nat-of-ord } i) \ A) \rrbracket e$
proof –
obtain *s k* **where** $v: \text{is-Var } v$ **and** $i: \text{Ord } i$ **and** $sk: \text{SeqAbstForm } v \ i \ x \ x' \ s \ k$
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-dbfm* [**where** $e=e$])
apply (*rule-tac* [1] $x=\text{DBEq } ta \ tb$ **in** *exI*)
apply (*rule-tac* [2] $x=\text{DBMem } ta \ tb$ **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* :: $hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$

```

  where SubstAtomic v tm y y'  $\equiv$ 
    ( $\exists t u t' u'. \text{SubstTerm } v \text{ tm } t t' \wedge \text{SubstTerm } v \text{ tm } u u' \wedge$ 
      $((y = q\text{-Eq } t u \wedge y' = q\text{-Eq } t' u') \vee (y = q\text{-Mem } t u \wedge y' = q\text{-Mem } t'$ 
      $u'))$ )

```

nominal-function *SubstAtomicP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$

```

  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')]  $\implies$ 
    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*]:

$a \# SubstAtomicP\ v\ tm\ y\ y' \longleftrightarrow a \# v \wedge a \# tm \wedge a \# y \wedge a \# y'$ (**is**
?*thesis1*)

and *eval-fm-SubstAtomicP* [*simp*]:

$eval-fm\ e\ (SubstAtomicP\ v\ tm\ y\ y') \longleftrightarrow SubstAtomic\ [v]e\ [tm]e\ [y]e\ [y']e$
(**is** ?*thesis2*)

and *SubstAtomicP-sf* [*iff*]: *Sigma-fm* (*SubstAtomicP* $v\ tm\ y\ y'$) (**is**
?*thsf*)

proof –

obtain $t::name$ **and** $u::name$ **and** $t'::name$ **and** $u'::name$

where $atom\ t \# (v,tm,y,y',t',u,u')$ $atom\ t' \# (v,tm,y,y',u,u')$
 $atom\ u \# (v,tm,y,y',u')$ $atom\ u' \# (v,tm,y,y')$

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 $atom\ t \# (v,tm,y,y',w,i,t',u,u')$ $atom\ t' \# (v,tm,y,y',w,i,u,u')$
 $atom\ u \# (v,tm,y,y',w,i,u')$ $atom\ u' \# (v,tm,y,y',w,i)$

by (*metis obtain-fresh*)

thus ?*thesis*

by (*simp add: SubstAtomicP.simps [of t - - - t' u u']*)

qed

lemma *SubstAtomicP-cong*:

$\llbracket H \vdash v\ EQ\ v'; H \vdash tm\ EQ\ tm'; H \vdash x\ EQ\ x'; H \vdash y\ EQ\ y' \rrbracket$
 $\implies H \vdash SubstAtomicP\ v\ tm\ x\ y\ IFF\ SubstAtomicP\ v'\ tm'\ x'\ y'$

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* $y\ y'\ u\ u'\ w\ w' \equiv$

$((y = q-Disj\ u\ w \wedge y' = q-Disj\ u'\ w') \vee$
 $(y = q-Neg\ u \wedge y' = q-Neg\ u') \vee$
 $(y = q-Ex\ u \wedge y' = q-Ex\ u'))$

definition *SeqSubstForm* :: *hf* \Rightarrow *hf* \Rightarrow *hf* \Rightarrow *hf* \Rightarrow *hf* \Rightarrow *hf* \Rightarrow *bool*
where *SeqSubstForm* *v u x x' s k* \equiv *BuildSeq2* (*SubstAtomic* *v u*) *SubstMakeForm* *s k x x'*

nominal-function *SeqSubstFormP* :: *tm* \Rightarrow *tm* \Rightarrow *tm* \Rightarrow *tm* \Rightarrow *tm* \Rightarrow *tm* \Rightarrow *fm*

where \llbracket *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* $\rrbracket \Longrightarrow$
SeqSubstFormP *v u x x' s k* =
LstSeqP *s k* (*HPair* *x x'*) *AND*
All2 *l* (*SUCC* *k*) (*Ex* *sl* (*Ex* *sl'* (*HPair* (*Var* *l*) (*HPair* (*Var* *sl*) (*Var* *sl'*)) *IN* *s* *AND*
(*SubstAtomicP* *v u* (*Var* *sl*) (*Var* *sl'*) *OR*
Ex *m* (*Ex* *n* (*Ex* *sm* (*Ex* *sm'* (*Ex* *sn* (*Ex* *sn'* (*Var* *m* *IN* *Var* *l* *AND*
Var *n* *IN* *Var* *l* *AND*
HPair (*Var* *m*) (*HPair* (*Var* *sm*) (*Var* *sm'*)) *IN* *s* *AND*
HPair (*Var* *n*) (*HPair* (*Var* *sn*) (*Var* *sn'*)) *IN* *s* *AND*
(*Var* *sl* *EQ* *Q-Disj* (*Var* *sm*) (*Var* *sn*) *AND*
Var *sl'* *EQ* *Q-Disj* (*Var* *sm'*) (*Var* *sn'*)) *OR*
(*Var* *sl* *EQ* *Q-Neg* (*Var* *sm*) *AND* *Var* *sl'* *EQ* *Q-Neg* (*Var* *sm'*))
OR
(*Var* *sl* *EQ* *Q-Ex* (*Var* *sm*) *AND* *Var* *sl'* *EQ* *Q-Ex* (*Var* *sm'*))))))))))

apply (*simp-all* *add: eqvt-def SeqSubstFormP-graph-aux-def flip-fresh-fresh*)
by *auto* (*metis* *obtain-fresh*)

nominal-termination (*eqvt*)

by *lexicographic-order*

lemma

shows *SeqSubstFormP-fresh-iff* [*simp*]:

$a \# \text{SeqSubstFormP } v u x x' s k \longleftrightarrow a \# v \wedge a \# u \wedge a \# x \wedge a \# x' \wedge a \# s$
 $\wedge a \# k$ (**is** *?thesis1*)

and *eval-fm-SeqSubstFormP* [*simp*]:

$\text{eval-fm } e (\text{SeqSubstFormP } v u x x' s k) \longleftrightarrow$
 $\text{SeqSubstForm } \llbracket v \rrbracket e \llbracket u \rrbracket e \llbracket x \rrbracket e \llbracket x' \rrbracket e \llbracket s \rrbracket e \llbracket k \rrbracket e$ (**is** *?thesis2*)

and *SeqSubstFormP-sf* [*iff*]:

$\text{Sigma-fm } (\text{SeqSubstFormP } v u x x' s k)$ (**is** *?thsf*)

and *SeqSubstFormP-imp-OrdP*:

$\{ \text{SeqSubstFormP } v u x x' s k \} \vdash \text{OrdP } k$ (**is** *?thOrd*)

and *SeqSubstFormP-imp-LstSeqP*:

$\{ \text{SeqSubstFormP } v u x x' s k \} \vdash \text{LstSeqP } s k (\text{HPair } x x')$ (**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*
unfolding *SeqSubstForm-def BuildSeq2-def BuildSeq-def Builds-def*
HBall-def HBex-def q-defs SubstMakeForm-def
by (*force simp add: LstSeq-imp-Ord Ord-trans [of - - succ [k]]e*)
Seq-iff-app [of [s]e, OF LstSeq-imp-Seq-succ]
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:*
 $[H \vdash t\ EQ\ t'; H \vdash u\ EQ\ u'; H \vdash s\ EQ\ s'; H \vdash k\ EQ\ k']$
 $\implies H \vdash SeqSubstFormP\ v\ i\ t\ u\ s\ k\ IFF\ SeqSubstFormP\ v\ i\ t'\ u'\ s'\ k'$
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 \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool*
where $SubstForm\ v\ u\ x\ x' \equiv is-Var\ v \wedge Term\ u \wedge (\exists s\ k. SeqSubstForm\ v\ u\ x\ x'\ s\ k)$

nominal-function *SubstFormP* :: *tm* \Rightarrow *tm* \Rightarrow *tm* \Rightarrow *tm* \Rightarrow *fm*
where $\llbracket \text{atom } s \# (v, i, x, x', k); \text{atom } k \# (v, i, x, x') \rrbracket \Longrightarrow$
 $\text{SubstFormP } v \ i \ x \ x' =$
 $\text{VarP } v \ \text{AND } \text{TermP } i \ \text{AND } \text{Ex } s \ (\text{Ex } k \ (\text{SeqSubstFormP } v \ i \ x \ x' \ (\text{Var } s) \ (\text{Var } k)))$
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*]:

$a \# \text{SubstFormP } v \ i \ x \ x' \longleftrightarrow a \# v \wedge a \# i \wedge a \# x \wedge a \# x'$ (**is** *?thesis1*)

and *eval-fm-SubstFormP* [*simp*]:

$\text{eval-fm } e \ (\text{SubstFormP } v \ i \ x \ x') \longleftrightarrow \text{SubstForm } \llbracket v \rrbracket e \ \llbracket i \rrbracket e \ \llbracket x \rrbracket e \ \llbracket x' \rrbracket e$ (**is** *?thesis2*)

and *SubstFormP-sf* [*iff*]:

$\text{Sigma-fm } (\text{SubstFormP } v \ i \ x \ x')$ (**is** *?thsf*)

proof –

obtain *s::name* **and** *k::name*

where $\text{atom } s \# (v, i, x, x', k) \ \text{atom } k \# (v, i, x, x')$

by (*metis obtain-fresh*)

thus *?thesis1* *?thesis2* *?thsf*

by (*auto simp: SubstForm-def*)

qed

lemma *SubstFormP-subst* [*simp*]:

$(\text{SubstFormP } v \ i \ x \ x')(j::=t) = \text{SubstFormP } (\text{subst } j \ t \ v) \ (\text{subst } j \ t \ i) \ (\text{subst } j \ t \ x) \ (\text{subst } j \ t \ x')$

proof –

obtain *s::name* **and** *k::name* **where** $\text{atom } s \# (v, i, x, x', t, j, k) \ \text{atom } k \# (v, i, x, x', t, j)$

by (*metis obtain-fresh*)

thus *?thesis*

by (*auto simp: SubstFormP.simps [of s - - - k]*)

qed

lemma *SubstFormP-cong*:

$\llbracket H \vdash v \ EQ \ v'; H \vdash i \ EQ \ i'; H \vdash t \ EQ \ t'; H \vdash u \ EQ \ u' \rrbracket$

$\Longrightarrow H \vdash \text{SubstFormP } v \ i \ t \ u \ \text{IFF } \text{SubstFormP } v' \ i' \ t' \ u'$

by (*rule P4-cong*) *auto*

lemma *ground-SubstFormP* [*simp*]: $\text{ground-fm } (\text{SubstFormP } v \ y \ x \ x') \longleftrightarrow \text{ground } v \wedge \text{ground } y \wedge \text{ground } x \wedge \text{ground } x'$

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 $\llbracket \text{quot-dbtm } u \rrbracket e$ x x'

shows $\exists A. x = \llbracket \text{quot-dbfm } A \rrbracket e \wedge$

$x' = \llbracket \text{quot-dbfm } (\text{subst-dbfm } u (\text{decode-Var } v) A) \rrbracket e$

proof –

obtain s k **where** v : *is-Var* v **and** u : *Term* $\llbracket \text{quot-dbtm } u \rrbracket e$

and sk : *SeqSubstForm* v $\llbracket \text{quot-dbtm } u \rrbracket e$ 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 $y = \llbracket \text{quot-dbfm } A \rrbracket e$ $y' = \llbracket \text{quot-dbfm } (\text{subst-dbfm } u (\text{decode-Var } v) A) \rrbracket e$

$z = \llbracket \text{quot-dbfm } B \rrbracket e$ $z' = \llbracket \text{quot-dbfm } (\text{subst-dbfm } u (\text{decode-Var } v) B) \rrbracket e$

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** $u = \llbracket \text{quot-dbtm } t \rrbracket e$

$x = \llbracket \text{quot-dbfm } A \rrbracket e$

$x' = \llbracket \text{quot-dbfm } (\text{subst-dbfm } t (\text{decode-Var } v) A) \rrbracket e$

proof –

obtain t **where** $u = \llbracket \text{quot-dbtm } t \rrbracket e$

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 $\text{SubstForm } (q\text{-Var } i) \llbracket \text{quot-dbtm } u \rrbracket e \llbracket \text{quot-dbfm } A \rrbracket e$
 $\llbracket \text{quot-dbfm } (\text{subst-dbfm } u \ i \ A) \rrbracket e$
apply (*induction* A *rule*: *dbfm.induct*)
apply (*force simp*: u *SubstForm-def* *SeqSubstForm-def* *SubstAtomic-def* *SubstMakeForm-def*
 $\text{SubstTerm-subst-dbtm}$ $q\text{-defs}$ *simp del*: $q\text{-Var-def}$
intro: *BuildSeq2-exI* *BuildSeq2-combine*)
done

corollary *SubstForm-subst-dbfm-eq*:
 $\llbracket v = q\text{-Var } i; \text{Term } ux; ux = \llbracket \text{quot-dbtm } u \rrbracket e; A' = \text{subst-dbfm } u \ i \ A \rrbracket$
 $\implies \text{SubstForm } v \ ux \llbracket \text{quot-dbfm } A \rrbracket e \llbracket \text{quot-dbfm } A' \rrbracket e$
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* :: $hf \Rightarrow bool$
where $\text{Atomic } y \equiv \exists t \ u. \text{Term } t \wedge \text{Term } u \wedge (y = q\text{-Eq } t \ u \vee y = q\text{-Mem } t \ u)$

nominal-function *AtomicP* :: $tm \Rightarrow fm$
where $\llbracket \text{atom } t \ \sharp (u,y); \text{atom } u \ \sharp y \rrbracket \implies$
 $\text{AtomicP } y = \text{Ex } t \ (\text{Ex } u \ (\text{TermP } (\text{Var } t) \ \text{AND } \text{TermP } (\text{Var } u) \ \text{AND}$
 $(y \ \text{EQ } \text{Q-Eq } (\text{Var } t) \ (\text{Var } u) \ \text{OR}$
 $y \ \text{EQ } \text{Q-Mem } (\text{Var } t) \ (\text{Var } u))))$
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*]: $a \ \sharp \ \text{AtomicP } y \longleftrightarrow a \ \sharp \ y$ (**is** *?thesis1*)
and *eval-fm-AtomicP* [*simp*]: $\text{eval-fm } e \ (\text{AtomicP } y) \longleftrightarrow \text{Atomic} \llbracket y \rrbracket e$ (**is**
?thesis2)
and *AtomicP-sf* [*iff*]: $\text{Sigma-fm } (\text{AtomicP } y)$ (**is** *?thsf*)
proof –
obtain $t::\text{name}$ **and** $u::\text{name}$ **where** $\text{atom } t \ \sharp (u,y)$ $\text{atom } u \ \sharp y$
by (*metis* *obtain-fresh*)
thus *?thesis1* *?thesis2* *?thsf*
by (*auto simp*: *Atomic-def* $q\text{-defs}$)
qed

5.9 The predicate *MakeForm*

definition *MakeForm* :: $hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$

where $MakeForm\ y\ u\ w \equiv$
 $y = q-Disj\ u\ w \vee y = q-Neg\ u \vee$
 $(\exists v\ u'.\ AbstForm\ v\ 0\ u\ u' \wedge y = q-Ex\ u')$

nominal-function $MakeFormP :: tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $\llbracket atom\ v\ \# (y,u,w,au); atom\ au\ \# (y,u,w) \rrbracket \implies$
 $MakeFormP\ y\ u\ w =$
 $y\ EQ\ Q-Disj\ u\ w\ OR\ y\ EQ\ Q-Neg\ u\ OR$
 $Ex\ v\ (Ex\ au\ (AbstFormP\ (Var\ v)\ Zero\ u\ (Var\ au)\ AND\ y\ EQ\ Q-Ex\ (Var$
 $au)))$
by $(auto\ simp: eqvt-def\ MakeFormP-graph-aux-def\ flip-fresh-fresh)\ (metis\ obtain-fresh)$

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

lemma

shows $MakeFormP-fresh-iff\ [simp]:$
 $a\ \#\ MakeFormP\ y\ u\ w \longleftrightarrow a\ \# y \wedge a\ \# u \wedge a\ \# w\ (\mathbf{is}\ ?thesis1)$
and $eval-fm-MakeFormP\ [simp]:$
 $eval-fm\ e\ (MakeFormP\ y\ u\ w) \longleftrightarrow MakeForm\ \llbracket y \rrbracket e\ \llbracket u \rrbracket e\ \llbracket w \rrbracket e\ (\mathbf{is}\ ?thesis2)$
and $MakeFormP-sf\ [iff]:$
 $Sigma-fm\ (MakeFormP\ y\ u\ w)\ (\mathbf{is}\ ?thsf)$

proof –

obtain $v::name$ **and** $au::name$ **where** $atom\ v\ \# (y,u,w,au)\ atom\ au\ \# (y,u,w)$
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 $\llbracket atom\ l\ \# (s,k,t,sl,m,n,sm,sn); atom\ sl\ \# (s,k,t,m,n,sm,sn);$
 $atom\ m\ \# (s,k,t,n,sm,sn); atom\ n\ \# (s,k,t,sm,sn);$
 $atom\ sm\ \# (s,k,t,sn); atom\ sn\ \# (s,k,t) \rrbracket \implies$
 $SeqFormP\ s\ k\ t =$
 $LstSeqP\ s\ k\ t\ AND$
 $All2\ n\ (SUCC\ k)\ (Ex\ sn\ (HPair\ (Var\ n)\ (Var\ sn)\ IN\ s\ AND\ (AtomicP\ (Var$
 $sn)\ OR$
 $Ex\ m\ (Ex\ l\ (Ex\ sm\ (Ex\ sl\ (Var\ m\ IN\ Var\ n\ AND\ Var\ l\ IN\ Var\ n$
 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]:
 $a \# \text{SeqFormP } s \ k \ t \longleftrightarrow a \# s \wedge a \# k \wedge a \# t$ (is ?thesis1)
and eval-fm-SeqFormP [simp]:
 $\text{eval-fm } e \ (\text{SeqFormP } s \ k \ t) \longleftrightarrow \text{SeqForm } \llbracket s \rrbracket e \ \llbracket k \rrbracket e \ \llbracket t \rrbracket e$ (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 # (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 ?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 $\llbracket s \rrbracket e$, OF LstSeq-imp-Seq-succ]
Ord-trans [of - - succ $\llbracket k \rrbracket e$])

qed

lemma SeqFormP-subst [simp]:
 $(\text{SeqFormP } s \ k \ t)(j::=w) = \text{SeqFormP } (\text{subst } j \ w \ s) \ (\text{subst } j \ w \ k) \ (\text{subst } j \ w \ t)$

proof –
obtain l::name and sl::name and m::name and n::name and sm::name and sn::name
where atom l # (j,w,s,t,k,sl,m,n,sm,sn) atom sl # (j,w,s,k,t,m,n,sm,sn)
atom m # (j,w,s,k,t,n,sm,sn) atom n # (j,w,s,k,t,sm,sn)
atom sm # (j,w,s,k,t,sn) atom sn # (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 \Rightarrow bool
where Form y $\equiv (\exists s \ k. \text{SeqForm } s \ k \ y)$

nominal-function FormP :: tm \Rightarrow fm
where $\llbracket \text{atom } k \ \# \ (s,y); \text{atom } s \ \# \ y \rrbracket \Longrightarrow$

$FormP\ y = Ex\ k\ (Ex\ s\ (SeqFormP\ (Var\ s)\ (Var\ k)\ y))$
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*]: $a \# FormP\ y \longleftrightarrow a \# y$ (**is** *?thesis1*)
and *eval-fm-FormP* [*simp*]: $eval\text{-}fm\ e\ (FormP\ y) \longleftrightarrow Form\ \llbracket y \rrbracket e$ (**is** *?thesis2*)
and *FormP-sf* [*iff*]: $Sigma\text{-}fm\ (FormP\ y)$ (**is** *?thsf*)
proof –
obtain *k::name* **and** *s::name* **where** $k: atom\ k \# (s,y)$ $atom\ s \# y$
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** $atom\ k \# (s,j,w,y)$ $atom\ s \# (j,w,y)$
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*:
 $AbstForm\ (q\text{-}Var\ i)\ 0\ \llbracket \llbracket A \rrbracket \rrbracket e\ \llbracket quot\text{-}dbfm\ (trans\text{-}fm\ [i]\ A) \rrbracket e$
by (*metis abst-trans-fm ord-of.simps(1) quot-fm-def AbstForm-abst-dbfm*)

corollary *AbstForm-trans-fm-eq*:
 $\llbracket x = \llbracket \llbracket A \rrbracket \rrbracket e; x' = \llbracket quot\text{-}dbfm\ (trans\text{-}fm\ [i]\ A) \rrbracket e \rrbracket \Longrightarrow AbstForm\ (q\text{-}Var\ i)\ 0\ x$
 x'
by (*metis AbstForm-trans-fm*)

lemma *wf-Form-quot-dbfm* [*simp*]:
assumes *wf-dbfm A* **shows** $Form\ \llbracket quot\text{-}dbfm\ A \rrbracket e$
using *assms*
proof (*induct rule: wf-dbfm.induct*)
case (*Mem tm1 tm2*)
hence $Atomic\ \llbracket quot\text{-}dbfm\ (DBMem\ tm1\ tm2) \rrbracket e$
by (*auto simp: Atomic-def quot-Mem q-Mem-def dest: wf-Term-quot-dbfm*)
thus *?case*
by (*auto simp: Form-def SeqForm-def BuildSeq-exI*)
next
case (*Eq tm1 tm2*)
hence $Atomic\ \llbracket quot\text{-}dbfm\ (DBEq\ tm1\ tm2) \rrbracket e$

```

    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  $\bigwedge y. \text{MakeForm } [[\text{quot-dbfm } (DBNeg A)]]e [[\text{quot-dbfm } A]]e y$ 
    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  $\bigwedge A y. \text{MakeForm } [[\text{quot-dbfm } (DBEx (abst-dbfm i 0 A) )]]e [[\text{quot-dbfm } A]]e$ 
 $y$ 
    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 $A :: \text{fm}$ shows $\text{Form } [[A]]e$*
 by (metis quot-fm-def wf-Form-quot-dbfm wf-dbfm-trans-fm)

lemma *Atomic-Form-is-wf-dbfm: Atomic $x \implies \exists A. \text{wf-dbfm } A \wedge x = [[\text{quot-dbfm } A]]e$*

```

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  $\exists A. \text{wf-dbfm } A \wedge q\text{-Eq } t u = [[\text{quot-dbfm } A]]e$ 
    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  $\exists A. \text{wf-dbfm } A \wedge q\text{-Mem } t u = [[\text{quot-dbfm } A]]e$ 
    by (auto simp: quot-Mem q-Mem-def)
qed

```

```

lemma SeqForm-imp-wf-dbfm:
  assumes SeqForm s k x
  shows  $\exists A. \text{wf-dbfm } A \wedge x = \llbracket \text{quot-dbfm } A \rrbracket e$ 
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 =  $\llbracket \text{quot-dbfm } A \rrbracket e$ 
    wf-dbfm B z =  $\llbracket \text{quot-dbfm } B \rrbracket e$ 
    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 =  $\llbracket \text{quot-dbfm } A \rrbracket e$ 
  by (metis assms SeqForm-imp-wf-dbfm Form-def)

```

```

lemma Form-imp-is-fm: assumes Form x obtains A::fm where x =  $\llbracket A \rrbracket e$ 
  by (metis assms Form-imp-wf-dbfm quot-fm-def wf-dbfm-imp-is-fm)

```

```

lemma SubstForm-imp-subst-fm:
  assumes SubstForm v  $\llbracket u \rrbracket e$  x x' Form x
  obtains A::fm where x =  $\llbracket A \rrbracket e$  x' =  $\llbracket A(\text{decode-Var } v::=u) \rrbracket e$ 
  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'  $\longleftrightarrow$ 
    ( $\exists t::tm. y = \llbracket t \rrbracket e \wedge (\exists A::fm. x = \llbracket A \rrbracket e \wedge x' = \llbracket A(\text{decode-Var } v::=t) \rrbracket e)$ )
  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*: $\text{SubstForm } (q\text{-Var } i) \llbracket [t] \rrbracket e \llbracket [A] \rrbracket e \ x' \longleftrightarrow x' = \llbracket [A(i::=t)] \rrbracket e$

by (*subst SubstForm-unique [where e=e]*) *auto*

lemma *SubstForm-quot*: $\text{SubstForm } \llbracket [\text{Var } i] \rrbracket e \llbracket [t] \rrbracket e \llbracket [A] \rrbracket e \llbracket [A(i::=t)] \rrbracket e$

by (*metis SubstForm-quot-unique eval-Var-q*)

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

definition *VarNonOccForm* :: $hf \Rightarrow hf \Rightarrow bool$

where $\text{VarNonOccForm } v \ x \equiv \text{Form } x \wedge \text{SubstForm } v \ 0 \ x \ x$

nominal-function *VarNonOccFormP* :: $tm \Rightarrow tm \Rightarrow fm$

where $\text{VarNonOccFormP } v \ x = \text{FormP } x \ \text{AND} \ \text{SubstFormP } v \ \text{Zero } x \ x$

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

nominal-termination (*eqvt*)

by *lexicographic-order*

lemma

shows *VarNonOccFormP-fresh-iff* [*simp*]: $a \ \# \ \text{VarNonOccFormP } v \ y \longleftrightarrow a \ \# \ v \wedge a \ \# \ y$ (**is** *?thesis1*)

and *eval-fm-VarNonOccFormP* [*simp*]:

$\text{eval-fm } e \ (\text{VarNonOccFormP } v \ y) \longleftrightarrow \text{VarNonOccForm } \llbracket [v] \rrbracket e \ \llbracket [y] \rrbracket e$ (**is** *?thesis2*)

and *VarNonOccFormP-sf* [*iff*]: $\text{Sigma-fm } (\text{VarNonOccFormP } v \ y)$ (**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 $\text{VarNonOccForm } v \ x$

shows $\exists A. \text{wf-dbfm } A \wedge x = \llbracket [\text{quot-dbfm } A] \rrbracket e \wedge \text{atom } (\text{decode-Var } v) \ \# \ A$

proof –

obtain A' **where** $A': \text{wf-dbfm } A' \ x = \llbracket [\text{quot-dbfm } A'] \rrbracket e \ \text{SubstForm } v \ \llbracket [\text{quot-dbfm } \text{DBZero}] \rrbracket e \ x \ x$

using *assms [unfolded VarNonOccForm-def]*

by *auto (metis Form-imp-wf-dbfm)*

then obtain A **where** $x = \llbracket [\text{quot-dbfm } A] \rrbracket e$

$x = \llbracket [\text{quot-dbfm } (\text{subst-dbfm } \text{DBZero } (\text{decode-Var } v) \ A)] \rrbracket e$

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** $x = \llbracket A \rrbracket e$ *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*:
 $wf-dbfm A \implies atom\ i\ \# A \implies VarNonOccForm\ (q-Var\ i)\ \llbracket quot-dbfm\ A \rrbracket e$
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** $atom\ i\ \# A \implies VarNonOccForm\ (q-Var\ i)\ \llbracket A \rrbracket e$
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 Sentential (Boolean) Axioms

definition *Sent-axioms* :: $hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$ where

```
Sent-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  $\equiv \{x. \exists y\ z\ w. \text{Form}\ y \wedge \text{Form}\ z \wedge \text{Form}\ w \wedge \text{Sent-axioms}\ x\ y\ z\ w\}$ 
```

nominal-function *SentP* :: $tm \Rightarrow fm$

```
where  $\llbracket atom\ y\ \#\ (z,w,x); atom\ z\ \#\ (w,x); atom\ w\ \#\ x \rrbracket \Longrightarrow$ 
   $\text{SentP}\ x = \text{Ex}\ y\ (\text{Ex}\ z\ (\text{Ex}\ w\ (\text{FormP}\ (\text{Var}\ y)\ \text{AND}\ \text{FormP}\ (\text{Var}\ z)\ \text{AND}\ \text{FormP}\ (\text{Var}\ w)\ \text{AND}$ 
     $(\ (x\ \text{EQ}\ \text{Q-Imp}\ (\text{Var}\ y)\ (\text{Var}\ y))\ \text{OR}$ 
     $(x\ \text{EQ}\ \text{Q-Imp}\ (\text{Var}\ y)\ (\text{Q-Disj}\ (\text{Var}\ y)\ (\text{Var}\ z))\ \text{OR}$ 
     $(x\ \text{EQ}\ \text{Q-Imp}\ (\text{Q-Disj}\ (\text{Var}\ y)\ (\text{Var}\ y))\ (\text{Var}\ y))\ \text{OR}$ 
     $(x\ \text{EQ}\ \text{Q-Imp}\ (\text{Q-Disj}\ (\text{Var}\ y)\ (\text{Q-Disj}\ (\text{Var}\ z)\ (\text{Var}\ w)))$ 
     $(\text{Q-Disj}\ (\text{Q-Disj}\ (\text{Var}\ y)\ (\text{Var}\ z))\ (\text{Var}\ w)))\ \text{OR}$ 
     $(x\ \text{EQ}\ \text{Q-Imp}\ (\text{Q-Disj}\ (\text{Var}\ y)\ (\text{Var}\ z))$ 
     $(\text{Q-Imp}\ (\text{Q-Disj}\ (\text{Q-Neg}\ (\text{Var}\ y))\ (\text{Var}\ w))\ (\text{Q-Disj}\ (\text{Var}$ 
     $z)\ (\text{Var}\ w))))))$ 
  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*]: $a \# \text{SentP } x \longleftrightarrow a \# x$ (is *?thesis1*)
and *eval-fm-SentP* [*simp*]: $\text{eval-fm } e (\text{SentP } x) \longleftrightarrow \llbracket x \rrbracket e \in \text{Sent}$ (is *?thesis2*)
and *SentP-sf* [*iff*]: $\text{Sigma-fm } (\text{SentP } x)$ (is *?thsf*)

proof –

obtain *y::name* **and** *z::name* **and** *w::name* **where** *atom y* $\# (z, w, x)$ *atom z* $\# (w, x)$ *atom w* $\# x$
 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**

$\text{Equality-ax} \equiv \{ \llbracket \text{refl-ax} \rrbracket e0, \llbracket \text{eq-cong-ax} \rrbracket e0, \llbracket \text{mem-cong-ax} \rrbracket e0, \llbracket \text{eats-cong-ax} \rrbracket e0 \}$

function *Equality-axP* :: *tm* \Rightarrow *fm*

where *Equality-axP* *x* =
 $x \text{EQ} \llbracket \text{refl-ax} \rrbracket \text{OR } x \text{EQ} \llbracket \text{eq-cong-ax} \rrbracket \text{OR } x \text{EQ} \llbracket \text{mem-cong-ax} \rrbracket \text{OR } x \text{EQ} \llbracket \text{eats-cong-ax} \rrbracket$
 by *auto*

termination

by *lexicographic-order*

lemma *eval-fm-Equality-axP* [*simp*]: $\text{eval-fm } e (\text{Equality-axP } x) \longleftrightarrow \llbracket x \rrbracket e \in \text{Equality-ax}$
 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**

$\text{HF-ax} \equiv \{ \llbracket \text{HF1} \rrbracket e0, \llbracket \text{HF2} \rrbracket e0 \}$

function *HF-axP* :: *tm* \Rightarrow *fm*

where *HF-axP* *x* = $x \text{EQ} \llbracket \text{HF1} \rrbracket \text{OR } x \text{EQ} \llbracket \text{HF2} \rrbracket$
 by *auto*

termination

by *lexicographic-order*

lemma *eval-fm-HF-axP* [*simp*]: $\text{eval-fm } e (\text{HF-axP } x) \longleftrightarrow \llbracket x \rrbracket e \in \text{HF-ax}$
 by (*auto simp: HF-ax-def intro: eval-quot-fm-ignore*)

lemma *HF-axP-sf* [*iff*]: $\text{Sigma-fm } (\text{HF-axP } t)$

by *auto*

6.1.4 The specialisation axioms

inductive-set *Special-ax* :: *hf set* **where**

I: $\llbracket \text{AbstForm } v \ 0 \ x \ ax; \text{SubstForm } v \ y \ x \ sx; \text{Form } x; \text{is-Var } v; \text{Term } y \rrbracket$
 $\implies q\text{-Imp } sx \ (q\text{-Ex } ax) \in \text{Special-ax}$

Defining the syntax

nominal-function *Special-axP* :: *tm* \Rightarrow *fm* **where**

$\llbracket \text{atom } v \ \# \ (p, sx, y, ax, x); \text{atom } x \ \# \ (p, sx, y, ax);$
 $\text{atom } ax \ \# \ (p, sx, y); \text{atom } y \ \# \ (p, sx); \text{atom } sx \ \# \ p \rrbracket \implies$
 $\text{Special-axP } p = \text{Ex } v \ (\text{Ex } x \ (\text{Ex } ax \ (\text{Ex } y \ (\text{Ex } sx$
 $(\text{FormP } (\text{Var } x) \ \text{AND } \text{VarP } (\text{Var } v) \ \text{AND } \text{TermP } (\text{Var } y) \ \text{AND}$
 $\text{AbstFormP } (\text{Var } v) \ \text{Zero } (\text{Var } x) \ (\text{Var } ax) \ \text{AND}$
 $\text{SubstFormP } (\text{Var } v) \ (\text{Var } y) \ (\text{Var } x) \ (\text{Var } sx) \ \text{AND}$
 $p \ \text{EQ } Q\text{-Imp } (\text{Var } sx) \ (Q\text{-Ex } (\text{Var } ax))))))$

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*]: $a \ \# \ \text{Special-axP } p \longleftrightarrow a \ \# \ p$ (**is** *?thesis1*)
and *eval-fm-Special-axP* [*simp*]: $\text{eval-fm } e \ (\text{Special-axP } p) \longleftrightarrow \llbracket p \rrbracket e \in \text{Special-ax}$
(is *?thesis2*)
and *Special-axP-sf* [*iff*]: $\text{Sigma-fm } (\text{Special-axP } p)$ (**is** *?thesis3*)

proof –

obtain *v::name* **and** *x::name* **and** *ax::name* **and** *y::name* **and** *sx::name*
where $\text{atom } v \ \# \ (p, sx, y, ax, x) \ \text{atom } x \ \# \ (p, sx, y, ax)$
 $\text{atom } ax \ \# \ (p, sx, y) \ \text{atom } y \ \# \ (p, sx) \ \text{atom } sx \ \# \ p$
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 $x \in \text{Special-ax}$ **shows** $\exists A. x = \llbracket [A] \rrbracket e \wedge A \in \text{special-axioms}$

using *assms*

proof (*induction rule: Special-ax.induct*)

case (*I v x ax y sx*)

obtain *fm::fm* **and** *u::tm* **where** $fm: x = \llbracket [fm] \rrbracket e$ **and** $u: y = \llbracket [u] \rrbracket e$
using *I* **by** (*auto elim!: Form-imp-is-fm Term-imp-is-tm*)

obtain *B* **where** $x: x = \llbracket \text{quot-dbfm } B \rrbracket e$

and *ax*: $ax = \llbracket \text{quot-dbfm } (\text{abst-dbfm } (\text{decode-Var } v) \ 0 \ B) \rrbracket e$

using *I* *AbstForm-imp-abst-dbfm* **by** *force*

```

obtain  $B'$  where  $x'$ :  $x = \llbracket \text{quot-dbfm } B' \rrbracket e$ 
      and  $sx$ :  $sx = \llbracket \text{quot-dbfm (subst-dbfm (trans-tm [] u) (decode-Var v))} \rrbracket e$ 
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  $fm(\text{decode-Var } v ::= u) \text{ IMP SyntaxN.Ex } (\text{decode-Var } v) \text{ fm} \in \text{special-axioms}$ 
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:  $A \in \text{special-axioms} \implies \llbracket A \rrbracket e \in \text{Special-ax}$ 
proof (induct rule: special-axioms.induct)
case ( $I A i t$ )
have  $\llbracket A(i ::= t) \text{ IMP SyntaxN.Ex } i A \rrbracket e =$ 
       $q\text{-Imp } \llbracket \text{quot-dbfm (subst-dbfm (trans-tm [] t) i (trans-fm [] A))} \rrbracket e$ 
       $(q\text{-Ex } \llbracket \text{quot-dbfm (trans-fm [i] A)} \rrbracket e)$ 
by (simp add: quot-fm-def q-defs)
also have  $\dots \in \text{Special-ax}$ 
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:  $\text{Special-ax} = (\bigcup A \in \text{special-axioms}. \{ \llbracket A \rrbracket e \})$ 
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*

```

 $I$ :  $\llbracket \text{SubstForm } v \ 0 \ x \ x0$ ;
       $\llbracket \text{SubstForm } v \ w \ x \ xw$ ;
       $\llbracket \text{SubstForm } v \ (q\text{-Eats } v \ w) \ x \ xevw$ ;
       $\llbracket \text{AbstForm } w \ 0 \ (q\text{-Imp } x \ (q\text{-Imp } xw \ xevw)) \ \text{allw}$ ;
       $\llbracket \text{AbstForm } v \ 0 \ (q\text{-All } \text{allw}) \ \text{allvw}$ ;
       $\llbracket \text{AbstForm } v \ 0 \ x \ ax$ ;
       $v \neq w$ ;  $\llbracket \text{VarNonOccForm } w \ x \rrbracket$ 
 $\implies q\text{-Imp } x0 \ (q\text{-Imp } (q\text{-All } \text{allvw}) \ (q\text{-All } ax)) \in \text{Induction-ax}$ 

```

Defining the syntax

nominal-function *Induction-axP* :: *tm* \Rightarrow *fm* **where**
 \llbracket *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* $\rrbracket \Longrightarrow$
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* \longleftrightarrow *a* $\#$ *p* (**is** *?thesis1*)
and *eval-fm-Induction-axP* [*simp*]:
eval-fm e (*Induction-axP p*) \longleftrightarrow \llbracket *p* \rrbracket *e* \in *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 \llbracket *p* \rrbracket *e* \in *Induction-ax* **using** *atoms*
by (*auto intro!: Induction-ax.I [unfolded q-defs]*)
next
assume \llbracket *p* \rrbracket *e* \in *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 ∈ Induction-ax shows ∃ A. x = ⟦⟦A⟦⟦e ∧ A ∈ 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 dw [simp]: decode-Var v ≠ decode-Var w atom (decode-Var w) ‡
    [decode-Var v]
    by (auto simp: AbstForm-def fresh-Cons)
  obtain A::fm where A: x = ⟦⟦A⟦⟦e and wfresh: atom (decode-Var w) ‡ A
    using I VarNonOccForm-imp-fresh by blast
  then obtain A' A'' where A': q-Imp (⟦⟦A⟦⟦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 ≡ A(decode-Var v ::= Var (decode-Var w))
  def Ae ≡ A(decode-Var v ::= Eats (Var (decode-Var v)) (Var (decode-Var w)))
  have x0: x0 = ⟦⟦A(decode-Var v ::= Zero)⟦⟦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 = ⟦⟦Aw⟦⟦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 (⟦⟦Eats (Var (decode-Var v)) (Var (decode-Var w))⟦⟦e) x xevw
    using I by (simp add: quot-simps q-defs) (metis is-Var-iff v w)
  hence xevw: xevw = ⟦⟦Ae⟦⟦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  $\equiv$ 
  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  $\in$  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 allw ax x0 abst-trans-fm2 abst-trans-fm)
qed

```

lemma induction-axioms-into-Induction-ax:

```

  A  $\in$  induction-axioms  $\implies$   $\llbracket A \rrbracket e \in$  Induction-ax
proof (induct rule: induction-axioms.induct)
case (ind j i A)
hence eq:  $\llbracket A(i ::= Zero) IMP All i (All j (A IMP A(i ::= Var j) IMP A(i ::= Eats$ 
  (Var i) (Var j)))) IMP All i A \rrbracket e =
  q-Imp  $\llbracket quot-dbfm (subst-dbfm (trans-tm [] Zero) i (trans-fm [] A)) \rrbracket e$ 
  (q-Imp (q-All (q-All
    (q-Imp  $\llbracket quot-dbfm (trans-fm [j, i] A) \rrbracket e$ 
      (q-Imp
         $\llbracket quot-dbfm (trans-fm [j, i] (A(i ::= Var j))) \rrbracket e$ 
         $\llbracket quot-dbfm (trans-fm [j, i] (A(i ::= Eats (Var i) (Var j)))) \rrbracket e$ 
      )))
    (q-All  $\llbracket quot-dbfm (trans-fm [i] A) \rrbracket e$ 
      )))
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  $\llbracket A \rrbracket e$ 
   $\llbracket quot-dbfm (subst-dbfm (trans-tm [] Zero) i (trans-fm [] A)) \rrbracket e$ 
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)  $\llbracket A \rrbracket e$   $\llbracket quot-dbfm (subst-dbfm (DBVar j)$ 
  i (trans-fm [] A)) \rrbracket 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))  $\llbracket A \rrbracket e$ 
   $\llbracket quot-dbfm (subst-dbfm (DBEats (DBVar i) (DBVar j)) i (trans-fm []$ 
  A)) \rrbracket 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  $\llbracket A \rrbracket e$ 
      (q-Imp  $\llbracket \text{quot-dbfm} (\text{subst-dbfm} (\text{DBVar } j) i (\text{trans-fm } \llbracket A \rrbracket)) \rrbracket e$ 
         $\llbracket \text{quot-dbfm} (\text{subst-dbfm} (\text{DBEats} (\text{DBVar } i) (\text{DBVar } j)) i (\text{trans-fm}
\llbracket A \rrbracket)) \rrbracket e$ 
           $\llbracket \text{quot-dbfm} (\text{trans-fm } [j] (A \text{ IMP } (A(i::= \text{Var } j) \text{ IMP } A(i::= \text{Eats} (\text{Var }
i) (\text{Var } j)))) \rrbracket e$ 
            by (rule AbstForm-trans-fm-eq [where  $A = (A \text{ IMP } A(i::= \text{Var } j) \text{ IMP }
A(i::= \text{Eats} (\text{Var } i) (\text{Var } j))$ ]))
              (auto simp: quot-simps q-defs quot-fm-def subst-fm-trans-commute-eq)
    next
      show AbstForm (q-Var i) 0
        (q-All  $\llbracket \text{quot-dbfm} (\text{trans-fm } [j] (A \text{ IMP } A(i::= \text{Var } j) \text{ IMP } A(i::= \text{Eats} (\text{Var } i)
(\text{Var } j)))) \rrbracket e$ 
          (q-All
            (q-Imp  $\llbracket \text{quot-dbfm} (\text{trans-fm } [j, i] A) \rrbracket e$ 
              (q-Imp  $\llbracket \text{quot-dbfm} (\text{trans-fm } [j, i] (A(i::= \text{Var } j))) \rrbracket e$ 
                 $\llbracket \text{quot-dbfm} (\text{trans-fm } [j, i] (A(i::= \text{Eats} (\text{Var } i) (\text{Var } j)))) \rrbracket e$ ))
            apply (rule AbstForm-trans-fm-eq
              [where  $A = \text{All } j (A \text{ IMP } (A(i::= \text{Var } j) \text{ IMP } A(i::= \text{Eats} (\text{Var } i) (\text{Var }
j))))$ ]))
            apply (auto simp: q-defs quot-fm-def)
          done
    next
      show AbstForm (q-Var i) 0 ( $\llbracket A \rrbracket e$ )  $\llbracket \text{quot-dbfm} (\text{trans-fm } [i] A) \rrbracket e$ 
        by (metis AbstForm-trans-fm)
    next
      show q-Var i  $\neq$  q-Var j using ind
        by (simp add: q-Var-def)
    next
      show VarNonOccForm (q-Var j) ( $\llbracket A \rrbracket e$ )
        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*:

$\text{Induction-ax} = (\bigcup A \in \text{induction-axioms. } \{\llbracket A \rrbracket e\})$

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**

$\text{Extra-ax} \equiv \{\llbracket \text{extra-axiom} \rrbracket e0\}$

definition *Axiom* :: hf set **where**

$\text{Axiom} \equiv \text{Extra-ax} \cup \text{Sent} \cup \text{Equality-ax} \cup \text{HF-ax} \cup \text{Special-ax} \cup \text{Induction-ax}$

definition *AxiomP* :: tm \Rightarrow fm

where $AxiomP\ x \equiv x\ EQ\ [extra-axiom]\ OR\ SentP\ x\ OR\ Equality-axP\ x\ OR$
 $HF-axP\ x\ OR\ Special-axP\ x\ OR\ Induction-axP\ x$

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

lemma $AxiomP-fresh-iff\ [simp]: a \# AxiomP\ x \longleftrightarrow a \# x$
by $(auto\ simp: AxiomP-def)$

lemma $eval-fm-AxiomP\ [simp]: eval-fm\ e\ (AxiomP\ x) \longleftrightarrow \llbracket x \rrbracket e \in Axiom$
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 $ModPon :: hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$ **where**
 $ModPon\ x\ y\ z \equiv (y = q-Imp\ x\ z)$

definition $ModPonP :: tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $ModPonP\ x\ y\ z = (y\ EQ\ Q-Imp\ x\ z)$

lemma $ModPonP-eqvt\ [eqvt]: (p \cdot ModPonP\ x\ y\ z) = ModPonP\ (p \cdot x)\ (p \cdot y)\ (p \cdot z)$
by $(simp\ add: ModPonP-def)$

lemma $ModPonP-fresh-iff\ [simp]: a \# ModPonP\ x\ y\ z \longleftrightarrow a \# x \wedge a \# y \wedge a \# z$
by $(auto\ simp: ModPonP-def)$

lemma $eval-fm-ModPonP\ [simp]: eval-fm\ e\ (ModPonP\ x\ y\ z) \longleftrightarrow ModPon\ \llbracket x \rrbracket e$
 $\llbracket y \rrbracket e\ \llbracket z \rrbracket e$
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 $Exists :: hf \Rightarrow hf \Rightarrow bool$ **where**
 $Exists\ p\ q \equiv (\exists x\ x'\ y\ v.\ Form\ x \wedge VarNonOccForm\ v\ y \wedge AbstForm\ v\ 0\ x\ x' \wedge$
 $p = q-Imp\ x\ y \wedge q = q-Imp\ (q-Ex\ x')\ y)$

nominal-function *ExistsP* :: *tm* \Rightarrow *tm* \Rightarrow *fm* **where**
 $\llbracket \text{atom } x \# (p, q, v, y, x'); \text{atom } x' \# (p, q, v, y);$
 $\text{atom } y \# (p, q, v); \text{atom } v \# (p, q) \rrbracket \Longrightarrow$
 $\text{ExistsP } p \ q = \text{Ex } x \ (\text{Ex } x' \ (\text{Ex } y \ (\text{Ex } v \ (\text{FormP } (\text{Var } x) \ \text{AND}$
 $\text{VarNonOccFormP } (\text{Var } v) \ (\text{Var } y) \ \text{AND}$
 $\text{AbstFormP } (\text{Var } v) \ \text{Zero } (\text{Var } x) \ (\text{Var } x') \ \text{AND}$
 $p \ \text{EQ } Q\text{-Imp } (\text{Var } x) \ (\text{Var } y) \ \text{AND}$
 $q \ \text{EQ } Q\text{-Imp } (Q\text{-Ex } (\text{Var } x')) \ (\text{Var } y))))))$
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*]: $a \# \text{ExistsP } p \ q \longleftrightarrow a \# p \wedge a \# q$ (**is** *?thesis1*)
and *eval-fm-ExistsP* [*simp*]: $\text{eval-fm } e \ (\text{ExistsP } p \ q) \longleftrightarrow \text{Exists } \llbracket p \rrbracket e \ \llbracket q \rrbracket e$ (**is** *?thesis2*)
and *ExistsP-sf* [*iff*]: $\text{Sigma-fm } (\text{ExistsP } p \ q)$ (**is** *?thesis3*)

proof –

obtain *x::name* **and** *x'::name* **and** *y::name* **and** *v::name*
where $\text{atom } x \# (p, q, v, y, x')$ $\text{atom } x' \# (p, q, v, y)$ $\text{atom } y \# (p, q, v)$ $\text{atom } v \# (p, q)$
by (*metis obtain-fresh*)
thus *?thesis1* *?thesis2* *?thesis3*
by (*auto simp: Exists-def q-defs*)

qed

lemma *ExistsP-subst* [*simp*]: $(\text{ExistsP } p \ q)(j::=w) = \text{ExistsP } (\text{subst } j \ w \ p) \ (\text{subst } j \ w \ q)$

proof –

obtain *x::name* **and** *x'::name* **and** *y::name* **and** *v::name*
where $\text{atom } x \# (j, w, p, q, v, y, x')$ $\text{atom } x' \# (j, w, p, q, v, y)$
 $\text{atom } y \# (j, w, p, q, v)$ $\text{atom } v \# (j, w, p, q)$
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 $\exists A \ B \ i. p = \llbracket [A \ \text{IMP } B] \rrbracket e \wedge q = \llbracket [(Ex \ i \ A) \ \text{IMP } B] \rrbracket e \wedge \text{atom } i \# B$

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-Imp x y
    and q: q = q-Imp (q-Ex ax) y
    using assms by (auto simp: Exists-def)
  then obtain B::fm where B: y =  $\llbracket B \rrbracket e$  and vfresh: atom (decode-Var v)  $\#$  B
    by (metis VarNonOccForm-imp-fresh)
  obtain A::fm where A: x =  $\llbracket A \rrbracket e$ 
    by (metis Form-imp-is-fm x)
  with AbstForm-imp-abst-dbfm [OF abst, of e]
  have ax: ax =  $\llbracket \text{quot-dbfm} (\text{abst-dbfm} (\text{decode-Var } v) 0 (\text{trans-fm } [] A)) \rrbracket e$ 
    p =  $\llbracket A \text{ IMP } B \rrbracket e$  using p A B
    by (auto simp: quot-simps quot-fm-def q-defs)
  have q =  $\llbracket (Ex (\text{decode-Var } v) A) \text{ IMP } B \rrbracket e$  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 $i \# B \implies \text{Exists} (\llbracket A \text{ IMP } B \rrbracket e) \llbracket (Ex i A) \text{ IMP } B \rrbracket e$
 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 $p \ q \longleftrightarrow (\exists A \ B \ i. p = \llbracket A \text{ IMP } B \rrbracket e \wedge q = \llbracket (Ex i A) \text{ IMP } B \rrbracket e \wedge \text{atom } i \# B)$
 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 $H \vdash A \implies H \vdash A (i ::= x)$

definition *Subst* :: $hf \Rightarrow hf \Rightarrow \text{bool}$ **where**

Subst $p \ q \equiv (\exists v \ u. \text{SubstForm } v \ u \ p \ q)$

nominal-function *SubstP* :: $tm \Rightarrow tm \Rightarrow fm$ **where**

$\llbracket \text{atom } u \# (p, q, v); \text{atom } v \# (p, q) \rrbracket \implies$

SubstP $p \ q = Ex \ v (Ex \ u (\text{SubstFormP} (\text{Var } v) (\text{Var } u) p \ q))$

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]: $a \# \text{SubstP } p \ q \longleftrightarrow a \# p \wedge a \# q$ (is ?thesis1)
and *eval-fm-SubstP* [simp]: $\text{eval-fm } e \ (\text{SubstP } p \ q) \longleftrightarrow \text{Subst } \llbracket p \rrbracket e \ \llbracket q \rrbracket e$ (is ?thesis2)
and *SubstP-sf* [iff]: $\text{Sigma-fm } (\text{SubstP } p \ q)$ (is ?thesis3)
proof –
obtain *u::name* **and** *v::name* **where** $\text{atom } u \# (p, q, v)$ $\text{atom } v \# (p, q)$
by (*metis obtain-fresh*)
thus ?thesis1 ?thesis2 ?thesis3
by (*auto simp: Subst-def q-defs*)
qed

lemma *SubstP-subst* [simp]: $(\text{SubstP } p \ q)(j::=w) = \text{SubstP } (\text{subst } j \ w \ p) \ (\text{subst } j \ w \ q)$
proof –
obtain *u::name* **and** *v::name* **where** $\text{atom } u \# (j, w, p, q, v)$ $\text{atom } v \# (j, w, p, q)$
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 $\exists i \ t. p = \llbracket [A] \rrbracket e \wedge q = \llbracket [A(i::=t)] \rrbracket e$
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* $\llbracket [t] \rrbracket e \ p \ q$
by (*metis SubstForm-def Term-imp-is-tm*)
with *SubstForm-imp-subst-fm [OF substt] assms*
obtain *A* **where** $p = \llbracket [A] \rrbracket e$ $q = \llbracket [A(\text{decode-Var } v::=t)] \rrbracket e$
by *auto*
thus ?thesis
by *blast*
qed

6.1.10 The predicate *PrfP*

definition *Prf* :: $hf \Rightarrow hf \Rightarrow hf \Rightarrow \text{bool}$
where $\text{Prf } s \ k \ y \equiv \text{BuildSeq } (\lambda x. x \in \text{Axiom}) \ (\lambda u \ v \ w. \text{ModPon } v \ w \ u \vee \text{Exists } v \ u \vee \text{Subst } v \ u) \ s \ k \ y$

nominal-function *PrfP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $\llbracket \text{atom } l \# (s, sl, m, n, sm, sn); \text{atom } sl \# (s, m, n, sm, sn);$
 $\text{atom } m \# (s, n, sm, sn); \text{atom } n \# (s, k, sm, sn);$
 $\text{atom } sm \# (s, sn); \text{atom } sn \# (s) \rrbracket \Longrightarrow$
 $\text{PrfP } s \ k \ t =$
 $\text{LstSeqP } s \ k \ t \ \text{AND}$

$All2\ n\ (SUCC\ k)\ (Ex\ sn\ (HPair\ (Var\ n)\ (Var\ sn)\ IN\ s\ AND\ (AxiomP\ (Var\ sn)\ OR\ Ex\ m\ (Ex\ l\ (Ex\ sm\ (Ex\ sl\ (Var\ m\ IN\ Var\ n\ AND\ Var\ l\ IN\ Var\ n\ AND\ HPair\ (Var\ m)\ (Var\ sm)\ IN\ s\ AND\ HPair\ (Var\ l)\ (Var\ sl)\ IN\ s\ AND\ (ModPonP\ (Var\ sm)\ (Var\ sl)\ (Var\ sn)\ OR\ ExistsP\ (Var\ sm)\ (Var\ sn)\ OR\ SubstP\ (Var\ sm)\ (Var\ sn)))))))))$
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*]: $a\ \#\ PrfP\ s\ k\ t\ \longleftrightarrow\ a\ \#\ s\ \wedge\ a\ \#\ k\ \wedge\ a\ \#\ t$ (**is** *?thesis1*)
and *eval-fm-PrfP* [*simp*]: $eval\text{-}fm\ e\ (PrfP\ s\ k\ t)\ \longleftrightarrow\ Prf\ [s]e\ [k]e\ [t]e$ (**is** *?thesis2*)
and *PrfP-imp-OrdP* [*simp*]: $\{PrfP\ s\ k\ t\} \vdash OrdP\ k$ (**is** *?thord*)
and *PrfP-imp-LstSeqP* [*simp*]: $\{PrfP\ s\ k\ t\} \vdash LstSeqP\ s\ k\ t$ (**is** *?thlstseq*)
and *PrfP-sf* [*iff*]: $Sigma\text{-}fm\ (PrfP\ s\ k\ t)$ (**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 # (s,sl,m,n,sm,sn) atom sl # (s,m,n,sm,sn)*
atom m # (s,n,sm,sn) atom n # (s,k,sm,sn)
atom sm # (s,sn) atom sn # (s)

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 # (t,u,v,j,w,sl,m,n,sm,sn) atom sl # (t,u,v,j,w,m,n,sm,sn)*
atom m # (t,u,v,j,w,n,sm,sn) atom n # (t,u,v,j,w,sm,sn)
atom sm # (t,u,v,j,w,sn) atom sn # (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* \Rightarrow *bool*
where *Pf* *y* \equiv (\exists *s k*. *Prf* *s k y*)

nominal-function *PfP* :: *tm* \Rightarrow *fm*
where $\llbracket \text{atom } k \# (s,y); \text{atom } s \# y \rrbracket \Longrightarrow$
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* \longleftrightarrow *a* $\#$ *y* (is *?thesis1*)
and *eval-fm-PfP* [*simp*]: *eval-fm* *e* (*PfP* *y*) \longleftrightarrow *Pf* $\llbracket y \rrbracket e$ (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* $\llbracket \llbracket \text{extra-axiom} \rrbracket e \rrbracket$

proof –

have $\llbracket \llbracket \text{extra-axiom} \rrbracket e \rrbracket \in \text{Extra-ax}$
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  $\alpha \in \text{boolean-axioms}$  shows  $Pf \llbracket \alpha \rrbracket e$ 
proof –
  have  $\llbracket \alpha \rrbracket e \in \text{Sent}$  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  $\alpha \in \text{equality-axioms}$  shows  $Pf \llbracket \alpha \rrbracket e$ 
proof –
  have  $\llbracket \alpha \rrbracket e \in \text{Equality-ax}$  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  $\alpha \in \text{HF-axioms}$  shows  $Pf \llbracket \alpha \rrbracket e$ 
proof –
  have  $\llbracket \alpha \rrbracket e \in \text{HF-ax}$  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  $\alpha \in \text{special-axioms}$  shows  $Pf \llbracket \alpha \rrbracket e$ 
proof –
  have  $\llbracket \alpha \rrbracket e \in \text{Special-ax}$ 
    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  $\alpha \in \text{induction-axioms}$  shows  $Pf \llbracket \alpha \rrbracket e$ 
proof –
  have  $\llbracket \alpha \rrbracket e \in \text{Induction-ax}$ 
    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:  $\llbracket Pf \llbracket Q\text{-Imp } x \ y \rrbracket e; Pf \llbracket x \rrbracket e \rrbracket \implies Pf \llbracket y \rrbracket e$ 
  by (auto simp: Pf-def Prf-def ModPon-def q-defs intro: BuildSeq-combine)

```

lemma *quot-ModPon-imp-Pf*: $\llbracket Pf \llbracket [\alpha \text{ IMP } \beta] \rrbracket e; Pf \llbracket [\alpha] \rrbracket e \rrbracket \Longrightarrow Pf \llbracket [\beta] \rrbracket e$
by (*simp add: ModPon-imp-Pf quot-fm-def quot-simps q-defs*)

lemma *quot-Exists-imp-Pf*: $\llbracket Pf \llbracket [\alpha \text{ IMP } \beta] \rrbracket e; atom\ i\ \#\ \beta \rrbracket \Longrightarrow Pf \llbracket [Ex\ i\ \alpha \text{ IMP } \beta] \rrbracket e$
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** $Pf \llbracket [\alpha] \rrbracket e$
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*: $\{\} \vdash \alpha \Longrightarrow \{\} \vdash PfP\ [\alpha]$
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** $\exists A. x = \llbracket [A] \rrbracket e \wedge \{\} \vdash A$
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 $A: y = \llbracket [A] \rrbracket e$ **and** $B: z = \llbracket [B] \rrbracket e$ **and** $C: w = \llbracket [C] \rrbracket e$
by $(metis\ Form\text{-}imp\text{-}is\text{-}fm)$
have $\exists A. q\text{-}Imp\ y\ y = \llbracket [A] \rrbracket e \wedge \{\} \vdash A$
by $(force\ simp\ add: A\ quot\text{-}Disj\ quot\text{-}Neg\ q\text{-}defs\ hfthm.\ Bool\ boolean\text{-}axioms.\ intros)$
moreover have $\exists A. q\text{-}Imp\ y\ (q\text{-}Disj\ y\ z) = \llbracket [A] \rrbracket e \wedge \{\} \vdash A$
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 $\exists A. q\text{-}Imp\ (q\text{-}Disj\ y\ y)\ y = \llbracket [A] \rrbracket e \wedge \{\} \vdash A$
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 $\exists A. q\text{-}Imp\ (q\text{-}Disj\ y\ (q\text{-}Disj\ z\ w))\ (q\text{-}Disj\ (q\text{-}Disj\ y\ z)\ w) =$
 $\llbracket [A] \rrbracket e \wedge \{\} \vdash A$
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 $\exists A. q\text{-}Imp\ (q\text{-}Disj\ y\ z)\ (q\text{-}Imp\ (q\text{-}Disj\ (q\text{-}Neg\ y)\ w)\ (q\text{-}Disj\ z$
 $w)) = \llbracket [A] \rrbracket e \wedge \{\} \vdash A$
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** $x = \llbracket [A] \rrbracket e \wedge \{\} \vdash A$
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** $x = \llbracket [A] \rrbracket e \wedge \{\} \vdash A$
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** $x = \llbracket [A] \rrbracket e \wedge \{\} \vdash A$
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** $x = \llbracket [A] \rrbracket e \wedge \{\} \vdash A$
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** $x = \llbracket [A] \rrbracket e \wedge \{\} \vdash A$
by $(metis\ Induction\text{-}ax\text{-}imp\text{-}induction\text{-}axioms\ assms\ hfthm.Ind)$

lemma *Exists-imp-hfthm*: $\llbracket \text{Exists } \llbracket A \rrbracket e y; \{\} \vdash A \rrbracket \implies \exists B. y = \llbracket B \rrbracket e \wedge \{\} \vdash B$

by (*drule* *Exists-imp-exists* [**where** $e=e$]) (*auto intro: anti-deduction*)

lemma *Subst-imp-hfthm*: $\llbracket \text{Subst } \llbracket A \rrbracket e y; \{\} \vdash A \rrbracket \implies \exists B. y = \llbracket B \rrbracket e \wedge \{\} \vdash B$

by (*drule* *Subst-imp-subst* [**where** $e=e$], *auto intro: Subst*)

lemma *eval-Neg-imp-Neg*: $\llbracket \llbracket \alpha \rrbracket e = q\text{-Neg } x \rrbracket \implies \exists A. \alpha = \text{Neg } A \wedge \llbracket A \rrbracket e = x$

by (*cases* α *rule: fm.exhaust*) (*auto simp: quot-simps q-defs htuple-minus-1*)

lemma *eval-Disj-imp-Disj*: $\llbracket \llbracket \alpha \rrbracket e = q\text{-Disj } x y \rrbracket \implies \exists A B. \alpha = A \text{ OR } B \wedge \llbracket A \rrbracket e = x \wedge \llbracket B \rrbracket e = y$

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** $\exists A. x = \llbracket A \rrbracket e \wedge \{\} \vdash A$

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** $y = \llbracket A \rrbracket e \{\} \vdash A$ $z = \llbracket B \rrbracket e \{\} \vdash B$

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*: $Pf \llbracket \llbracket \alpha \rrbracket e \rrbracket \implies \{\} \vdash \alpha$

by (*metis Pf-def Prf-imp-proved eval-fm-inject*)

Proposition 4.4!

theorem *proved-iff-proved-PfP*: $\{\} \vdash \alpha \iff \{\} \vdash PfP \llbracket \alpha \rrbracket$

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
```

7.0.1 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 { $IndP\ x\ OR\ x\ NEQ\ v,\ x\ EQ\ HPair\ (HPair\ Zero\ (HPair\ Zero\ Zero))\ y$ }
 $\vdash\ A$
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* }
 $\vdash\ ((t\ EQ\ v\ AND\ u\ EQ\ i)\ OR$
 $((IndP\ t\ OR\ t\ NEQ\ v)\ AND\ u\ EQ\ t))\ OR$
 $Ex\ m\ (Ex\ n\ (Ex\ sm\ (Ex\ sm'\ (Ex\ sn\ (Ex\ sn'\ (Var\ m\ IN\ k\ AND\ Var\ n$
 $IN\ k\ AND$

$SeqStTermP\ v\ i\ (Var\ sm)\ (Var\ sm')\ s\ (Var\ m)\ AND$
 $SeqStTermP\ v\ i\ (Var\ sn)\ (Var\ sn')\ s\ (Var\ n)\ AND$
 $t\ EQ\ Q-Eats\ (Var\ sm)\ (Var\ sn)\ AND$
 $u\ EQ\ Q-Eats\ (Var\ sm')\ (Var\ sn'))))))))$

proof –

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  $x = \text{Var } sm \wedge$ , simp])
apply (rule Ex-I [where  $x = \text{Var } sn$ ], simp)
apply (rule Ex-I [where  $x = \text{Var } sn \wedge$ , 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*: $atom\ i \# (s, s', v, a, t, u, u')$ $atom\ j \# (s, s', v, a, t, i, t, u, u')$
 $atom\ j' \# (s, s', v, a, t, i, j, t, u, u')$
 $atom\ k \# (s, s', v, a, t, u, u', kk', i, j, j')$ $atom\ k' \# (s, s', v, a, t, u, u', k, i, j, j')$
 $atom\ l \# (s, s', v, a, t, i, j, j', k, k')$
 $atom\ m \# (s, s', v, a, i, j, j', k, k', l)$ $atom\ n \# (s, s', v, a, i, j, j', k, k', l, m)$
 $atom\ sm \# (s, s', v, a, i, j, j', k, k', l, m, n)$ $atom\ sn \# (s, s', v, a, i, j, j', k, k', l, m, n, sm)$
 $atom\ sm' \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn)$ $atom\ sn' \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm')$
 $atom\ m2 \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm', sn')$ $atom\ n2 \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2)$
 $atom\ sm2 \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2, n2)$ $atom\ sn2 \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2, n2, sm2)$
 $atom\ sm2' \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2, n2, sm2, sn2)$
 $atom\ sn2' \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2, n2, sm2, sn2, sm2')$

by (*metis obtain-fresh*)

have { *OrdP* (*Var* k), *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  $x = \text{Var } m$ ], blast)
apply (rule All-E' [OF Hyp, where  $x = \text{Var } n$ ], blast, simp)
apply (rule Disj-EH, blast intro: thin1 ContraProve)+
apply (rule All-E [where  $x = \text{Var } sm$ ], simp)
apply (rule All-E [where  $x = \text{Var } sm'$ ], simp)
apply (rule All-E [where  $x = \text{Var } sm2'$ ], simp)
apply (rule All-E [where  $x = \text{Var } m2$ ], simp)
apply (rule All-E [where  $x = \text{Var } sn$ , THEN rotate2], simp)
apply (rule All-E [where  $x = \text{Var } sn'$ ], simp)
apply (rule All-E [where  $x = \text{Var } sn2'$ ], simp)
apply (rule All-E [where  $x = \text{Var } n2$ ], simp)
apply (rule cut-same [where  $A = Q\text{-Eats } (\text{Var } sm) (\text{Var } sn) \text{ EQ } Q\text{-Eats } (\text{Var } sm2) (\text{Var } sn2)$ ])
apply (blast intro: Sym Trans, clarify)
apply (rule cut-same [where  $A = \text{SeqStTermP } v \ a \ (\text{Var } sn) (\text{Var } sn2') \ s' \ (\text{Var } n2)$ ])
apply (blast intro: Hyp SeqStTermP-cong [OF Hyp Refl Refl, THEN Iff-MP2-same])
apply (rule cut-same [where  $A = \text{SeqStTermP } v \ a \ (\text{Var } sm) (\text{Var } sm2') \ s' \ (\text{Var } m2)$ ])
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}
   $\vdash$  (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}
   $\vdash$  (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}
   $\vdash$  (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}
   $\vdash$  (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}  $\vdash$  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}  $\vdash$  ((SeqStTermP v a t u s (Var k) IMP (SeqStTermP v a t u'

```

$s' \text{ kk}' \text{ IMP } u' \text{ EQ } u$))
by (*metis Assume MP-same Imp-I*)
hence $\{ \text{VarP } v \} \vdash ((\text{SeqStTermP } v \text{ a } t \text{ u } s \text{ (Var } k) \text{ IMP } (\text{SeqStTermP } v \text{ a } t \text{ u}' \text{ s}' \text{ kk}' \text{ IMP } u' \text{ EQ } u)))(k::=kk)$
using atoms by (*force intro!: Subst*)
hence $\{ \text{VarP } v \} \vdash \text{SeqStTermP } v \text{ a } t \text{ u } s \text{ kk IMP } (\text{SeqStTermP } v \text{ a } t \text{ u}' \text{ s}' \text{ kk}' \text{ IMP } u' \text{ EQ } u)$
using atoms by simp
hence $\{ \text{SeqStTermP } v \text{ a } t \text{ u } s \text{ kk} \} \vdash \text{SeqStTermP } v \text{ a } t \text{ u } s \text{ kk IMP } (\text{SeqStTermP } v \text{ a } t \text{ u}' \text{ s}' \text{ kk}' \text{ IMP } u' \text{ EQ } u)$
by (*metis SeqStTermP-imp-VarP rcut1*)
thus *?thesis*
by (*metis Assume AssumeH(2) MP-same rcut1*)
qed

theorem *SubstTermP-unique*: $\{ \text{SubstTermP } v \text{ tm } t \text{ u}, \text{SubstTermP } v \text{ tm } t \text{ u}' \} \vdash u' \text{ EQ } u$

proof –

obtain $s::\text{name}$ **and** $s'::\text{name}$ **and** $k::\text{name}$ **and** $k'::\text{name}$
where $\text{atom } s \# (v, \text{tm}, t, u, u', k, k')$ $\text{atom } s' \# (v, \text{tm}, t, u, u', k, k', s)$
 $\text{atom } k \# (v, \text{tm}, t, u, u')$ $\text{atom } k' \# (v, \text{tm}, t, u, u', k)$
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

7.0.2 *SubstAtomicP*

lemma *SubstTermP-eq*:

$\llbracket H \vdash \text{SubstTermP } v \text{ tm } x \text{ z}; \text{insert } (\text{SubstTermP } v \text{ tm } y \text{ z}) H \vdash A \rrbracket \implies \text{insert } (x \text{ EQ } y) H \vdash A$

by (*metis Assume rotate2 Iff-E1 cut-same thin1 SubstTermP-cong [OF Refl Refl - Refl]*)

lemma *SubstAtomicP-unique*: $\{ \text{SubstAtomicP } v \text{ tm } x \text{ y}, \text{SubstAtomicP } v \text{ tm } x \text{ y}' \} \vdash y' \text{ EQ } y$

proof –

obtain $t::\text{name}$ **and** $ts::\text{name}$ **and** $u::\text{name}$ **and** $us::\text{name}$
and $t'::\text{name}$ **and** $ts'::\text{name}$ **and** $u'::\text{name}$ **and** $us'::\text{name}$
where $\text{atom } t \# (v, \text{tm}, x, y, y', ts, u, us)$ $\text{atom } ts \# (v, \text{tm}, x, y, y', u, us)$
 $\text{atom } u \# (v, \text{tm}, x, y, y', us)$ $\text{atom } us \# (v, \text{tm}, x, y, y')$
 $\text{atom } t' \# (v, \text{tm}, x, y, y', t, ts, u, us, ts', u', us')$ $\text{atom } ts' \# (v, \text{tm}, x, y, y', t, ts, u, us, u', us')$
 $\text{atom } u' \# (v, \text{tm}, x, y, y', t, ts, u, us, us')$ $\text{atom } us' \# (v, \text{tm}, x, y, y', t, ts, u, us)$
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

```

7.0.3 SeqSubstFormP

lemma SeqSubstFormP-lemma:

```

assumes atom m # (v,u,x,y,s,k,n,sm,sm',sn,sn') atom n # (v,u,x,y,s,k,sm,sm',sn,sn')
          atom sm # (v,u,x,y,s,k,sm',sn,sn') atom sm' # (v,u,x,y,s,k,sn,sn')
          atom sn # (v,u,x,y,s,k,sn') atom sn' # (v,u,x,y,s,k)
shows { SeqSubstFormP v u x y s k }
        ⊢ SubstAtomicP v u x y OR
          Ex m (Ex n (Ex sm (Ex sm' (Ex sn (Ex sn' (Var m IN k AND Var n IN
k AND
          SeqSubstFormP v u (Var sm) (Var sm') s (Var m) AND
          SeqSubstFormP v u (Var sn) (Var sn') s (Var n) AND
          (((x EQ Q-Disj (Var sm) (Var sn) AND y EQ Q-Disj (Var sm')
(Var sn')) OR
          (x EQ Q-Neg (Var sm) AND y EQ Q-Neg (Var sm')) OR
          (x EQ Q-Ex (Var sm) AND y EQ Q-Ex (Var sm'))))))))))))

```

proof –

```

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*: $\{y \text{ EQ } Q\text{-Neg } z, \text{ SubstAtomicP } v \text{ tm } y \ y'\} \vdash \text{Fls}$
(is ?thesis1)

and *Disj-SubstAtomicP-Fls*: $\{y \text{ EQ } Q\text{-Disj } z \ w, \text{ SubstAtomicP } v \text{ tm } y \ y'\} \vdash \text{Fls}$
(is ?thesis2)

and *Ex-SubstAtomicP-Fls*: $\{y \text{ EQ } Q\text{-Ex } z, \text{ SubstAtomicP } v \text{ tm } y \ y'\} \vdash \text{Fls}$
(is ?thesis3)

proof —

obtain *t::name* **and** *u::name* **and** *t'::name* **and** *u'::name*

where *atom t* $\# (z, w, v, \text{tm}, y, y', t', u, u')$ *atom t'* $\# (z, w, v, \text{tm}, y, y', u, u')$
atom u $\# (z, w, v, \text{tm}, y, y', u')$ *atom u'* $\# (z, w, v, \text{tm}, y, y')$

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*:
 $\llbracket H \vdash \text{SeqSubstFormP } v \text{ tm } x \text{ z } s \text{ k}; \text{ insert } (\text{SeqSubstFormP } v \text{ tm } y \text{ z } s \text{ k}) H \vdash A \rrbracket$
 $\implies \text{insert } (x \text{ EQ } y) H \vdash A$
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*: $\{\text{SeqSubstFormP } v \text{ a } x \text{ y } s \text{ kk}, \text{SeqSubstFormP } v \text{ a } x \text{ y}' \text{ s}' \text{ kk}'\} \vdash y' \text{ EQ } y$
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 $\# (s, s', v, a, x, y, y', k, i, j, j')$ *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*) }
 $\vdash \text{All } i \text{ (All } j \text{ (All } j' \text{ (All } k' \text{ (SeqSubstFormP } v \text{ a (Var } i) \text{ (Var } j) \text{ s (Var } k) \text{ IMP (SeqSubstFormP } v \text{ a (Var } i) \text{ (Var } j') \text{ s}' \text{ (Var } k') \text{ IMP Var } j' \text{ EQ Var } j))))))$
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*(γ) *Disj-I1 Neg-I anti-deduction cut2* [*OF Disj-SubstAtomicP-Fls*])
apply (rule *Conj-EH Disj-EH*)+
apply (metis *Assume AssumeH*(γ) *Disj-I1 Neg-I anti-deduction cut2* [*OF Neg-SubstAtomicP-Fls*])
apply (rule *Conj-EH*)+
apply (metis *Assume AssumeH*(γ) *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*(γ) *Disj-I1 Neg-I anti-deduction cut2* [*OF Disj-SubstAtomicP-Fls*])
apply (rule *Conj-EH Disj-EH*)+
apply (metis *Assume AssumeH*(γ) *Disj-I1 Neg-I anti-deduction cut2* [*OF Neg-SubstAtomicP-Fls*])
apply (rule *Conj-EH*)+
apply (metis *Assume AssumeH*(γ) *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* $x = \text{Var } m$], *blast*)
apply (rule *All-E'* [*OF Hyp, where* $x = \text{Var } n$], *blast, simp*)
apply (rule *Disj-EH, blast intro: thin1 ContraProve*)+
apply (rule *All-E* [*where* $x = \text{Var } sm$], *simp*)
apply (rule *All-E* [*where* $x = \text{Var } sm \uparrow$], *simp*)
apply (rule *All-E* [*where* $x = \text{Var } sm2 \uparrow$], *simp*)
apply (rule *All-E* [*where* $x = \text{Var } m2$], *simp*)
apply (rule *All-E* [*where* $x = \text{Var } sn, THEN rotate2$], *simp*)
apply (rule *All-E* [*where* $x = \text{Var } sn \uparrow$], *simp*)
apply (rule *All-E* [*where* $x = \text{Var } sn2 \uparrow$], *simp*)
apply (rule *All-E* [*where* $x = \text{Var } n2$], *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)* }
 \vdash (*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)* }
 \vdash (*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 $p3: \{OrdP (Var k)\}$
 $\vdash (All k' (SeqSubstFormP v a x y s (Var k)$
 $IMP (SeqSubstFormP v a x (Var j') s' (Var k') IMP Var j' EQ$
 $y))))(j'::=y')$
apply (rule *All-D*)
using atoms $p2$ **by** *simp*
have $p4: \{OrdP (Var k)\}$
 $\vdash (SeqSubstFormP v a x y s (Var k) IMP (SeqSubstFormP v a x y' s' (Var$
 $k') IMP y' EQ y))(k'::=kk')$
apply (rule *All-D*)
using atoms $p3$ **by** *simp*
hence $\{OrdP (Var k)\} \vdash SeqSubstFormP v a x y s (Var k) IMP (SeqSubstFormP$
 $v a x y' s' kk' IMP y' EQ y)$
using atoms **by** *simp*
hence $\{SeqSubstFormP v a x y s (Var k)\}$
 $\vdash SeqSubstFormP v a x y s (Var k) IMP (SeqSubstFormP v a x y' s' kk'$
 $IMP y' EQ y)$
by (*metis SeqSubstFormP-imp-OrdP rcut1*)
hence $\{\} \vdash SeqSubstFormP v a x y s (Var k) IMP (SeqSubstFormP v a x y' s'$
 $kk' IMP y' EQ y)$
by (*metis Assume Disj-Neg-2 Disj-commute anti-deduction Imp-I*)
hence $\{\} \vdash ((SeqSubstFormP v a x y s (Var k) IMP (SeqSubstFormP v a x y' s'$
 $kk' IMP y' EQ y))(k::=kk)$
using atoms **by** (*force intro!: Subst*)
thus *?thesis*
using atoms **by** *simp (metis DisjAssoc2 Disj-commute anti-deduction)*
qed

7.0.4 *SubstFormP*

theorem *SubstFormP-unique*: $\{SubstFormP v tm x y, SubstFormP v tm x y'\} \vdash y' EQ y$

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-Eats (app s l) (app s l))

nominal-function SeqWRP :: tm  $\Rightarrow$  tm  $\Rightarrow$  tm  $\Rightarrow$  fm
  where  $\llbracket$ atom l  $\sharp$  (s,k,sl); atom sl  $\sharp$  (s) $\rrbracket \implies$ 
    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-fm e (SeqWRP s k y)  $\longleftrightarrow$  SeqWR  $\llbracket$ s $\rrbracket$  e
 $\llbracket$ k $\rrbracket$  e  $\llbracket$ y $\rrbracket$  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 $\llbracket s \rrbracket e$, OF LstSeq-imp-Seq-succ]
Ord-trans [of - - succ $\llbracket k \rrbracket e$])

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* :: $hf \Rightarrow hf \Rightarrow bool$
where $WR\ x\ y \equiv (\exists s. SeqWR\ s\ x\ y)$

nominal-function *WRP* :: $tm \Rightarrow tm \Rightarrow fm$
where $\llbracket atom\ s \# (x,y) \rrbracket \Longrightarrow$
 $WRP\ x\ y = Ex\ s\ (SeqWRP\ (Var\ s)\ x\ y)$
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*]: $a \# WRP\ x\ y \longleftrightarrow a \# x \wedge a \# y$ (**is** $?thesis1$)
and *eval-fm-WRP* [*simp*]: $eval\ fm\ e\ (WRP\ x\ y) \longleftrightarrow WR\ \llbracket x \rrbracket e\ \llbracket y \rrbracket e$ (**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** $atom\ s\ \#\ (x,y,t,i)$

by (*metis obtain-fresh*)

thus *?thesis*

by (*auto simp: WRP.simps [of s]*)

qed

lemma *WRP-cong*: $H \vdash t\ EQ\ t' \implies H \vdash u\ EQ\ u' \implies H \vdash WRP\ t\ u\ IFF\ WRP\ t'\ u'$

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*: $Ord\ i \implies WR\ (succ\ i)\ (q\text{-Eats}\ y\ y) = WR\ i\ y$

by (*metis WR-succ-iff q-Eats-iff*)

lemma *WR-ord-of*: $WR\ (ord\text{-of}\ i)\ \llbracket [ORD\text{-OF}\ i] \rrbracket e$

by (*induct i*) (*auto simp: WR0-iff WR-succ-iff quot-Succ q-defs*)

Lemma 6.1

lemma *WR-quot-Var*: $WR \llbracket \llbracket Var\ x \rrbracket \rrbracket e \llbracket \llbracket \llbracket Var\ x \rrbracket \rrbracket \rrbracket e$
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*]: $ground\text{-}fm\ (WRP\ x\ y) \longleftrightarrow ground\ x \wedge ground\ y$
by (*auto simp: ground-aux-def ground-fm-aux-def supp-conv-fresh*)

lemma *prove-WRP*: $\{\} \vdash WRP \llbracket Var\ x \rrbracket \llbracket \llbracket Var\ x \rrbracket \rrbracket$
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 $atom\ l \# (s,k,sl)$ $atom\ sl \# (s)$
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 $y': atom\ y' \# (s,k,y)$
shows $\{SeqWRP\ s\ (SUCC\ k)\ y\} \vdash Ex\ y' (SeqWRP\ s\ k\ (Var\ y')\ AND\ y\ EQ\ Q\text{-}Succ\ (Var\ y'))$
proof –
obtain $l::name$ **and** $sl::name$
where $atoms: atom\ l \# (s,k,y,y',sl)$ $atom\ sl \# (s,k,y,y')$
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 $y': atom\ y' \# (s,k,y)$ **and** $k': H \vdash k'\ EQ\ (SUCC\ k)$
shows *insert* ($SeqWRP\ s\ k'\ y$) $H \vdash Ex\ y' (SeqWRP\ s\ k\ (Var\ y')\ AND\ y\ EQ)$

$Q\text{-Succ } (Var\ y')$
using $SeqWRP\text{-cong } [OF\ Refl\ k'\ Refl]\ cut1\ [OF\ SeqWRP\text{-SUCC}\text{-lemma } [of\ y'\ s\ k\ y]]$
by $(metis\ Assume\ Iff\text{-MP}\text{-left}\ Iff\text{-sym}\ y')$

lemma $SeqWRP\text{-unique}: \{OrdP\ x,\ SeqWRP\ s\ x\ y,\ SeqWRP\ s'\ x\ y'\} \vdash y'\ EQ\ y$
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 $(metis\ obtain\text{-fresh})$

have $\{OrdP\ (Var\ i)\} \vdash All\ j\ (All\ j'\ (SeqWRP\ s\ (Var\ i)\ (Var\ j)\ IMP\ (SeqWRP\ s'\ (Var\ i)\ (Var\ j')\ IMP\ Var\ j'\ EQ\ Var\ j)))$
apply $(rule\ OrdIndH\ [where\ j=k])$
using $i\ j\ j'\ atoms$ **apply** $auto$
apply $(rule\ rotate4)$
apply $(rule\ OrdP\text{-cases}\text{-E}\ [where\ k=pi],\ simp\text{-all})$
– Zero case
apply $(rule\ SeqWRP\text{-Zero}\text{-E}\ [THEN\ rotate3])$
prefer 2 **apply** $blast$
apply $(rule\ SeqWRP\text{-Zero}\text{-E}\ [THEN\ rotate4])$
prefer 2 **apply** $blast$
apply $(blast\ intro: ContraProve\ [THEN\ rotate4]\ Sym\ Trans)$
– SUCC case
apply $(rule\ Ex\text{-I}\ [where\ x = Var\ pi],\ auto)$
apply $(metis\ ContraProve\ EQ\text{-imp}\text{-SUBS2}\ Mem\text{-SUCC}\text{-I2}\ Refl\ Subset\text{-D})$
apply $(rule\ cut\text{-same})$
apply $(rule\ SeqWRP\text{-SUCC}\text{-E}\ [of\ sl'\ s'\ Var\ pi,\ THEN\ rotate4],\ auto)$
apply $(rule\ cut\text{-same})$
apply $(rule\ SeqWRP\text{-SUCC}\text{-E}\ [of\ sl\ s\ Var\ pi,\ THEN\ rotate7],\ auto)$
apply $(rule\ All\text{-E}\ [where\ x = Var\ sl,\ THEN\ rotate5],\ simp)$
apply $(rule\ All\text{-E}\ [where\ x = Var\ sl'],\ simp)$
apply $(rule\ Imp\text{-E},\ blast)+$
apply $(rule\ cut\text{-same}\ [OF\ Q\text{-Succ}\text{-cong}\ [OF\ Assume]])$
apply $(blast\ intro: Trans\ [OF\ Hyp\ Sym]\ HPair\text{-cong})$
done

hence $\{OrdP\ (Var\ i)\} \vdash (All\ j'\ (SeqWRP\ s\ (Var\ i)\ (Var\ j)\ IMP\ (SeqWRP\ s'\ (Var\ i)\ (Var\ j')\ IMP\ Var\ j'\ EQ\ Var\ j))) (j'::=y)$
by $(metis\ All\text{-D})$

hence $\{OrdP\ (Var\ i)\} \vdash (SeqWRP\ s\ (Var\ i)\ y\ IMP\ (SeqWRP\ s'\ (Var\ i)\ (Var\ j')\ IMP\ Var\ j'\ EQ\ y)) (j'::=y')$
using $j\ j'$
by $simp\ (drule\ All\text{-D}\ [where\ x=y'],\ simp)$

hence $\{\} \vdash OrdP\ (Var\ i)\ IMP\ (SeqWRP\ s\ (Var\ i)\ y\ IMP\ (SeqWRP\ s'\ (Var\ i)\ y'\ IMP\ y'\ EQ\ y))$
using $j\ j'$

by *simp* (*metis Imp-I*)
 hence $\{\} \vdash (\text{OrdP } (\text{Var } i) \text{ IMP } (\text{SeqWRP } s \ (\text{Var } i) \ y \ \text{IMP } (\text{SeqWRP } s' \ (\text{Var } i) \ y' \ \text{IMP } y' \ \text{EQ } y))) (i ::= x)$
 by (*metis Subst emptyE*)
 thus *?thesis using i*
 by *simp* (*metis anti-deduction insert-commute*)
qed

theorem *WRP-unique*: $\{\text{OrdP } x, \text{WRP } x \ y, \text{WRP } x \ y'\} \vdash y' \ \text{EQ } y$

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* \Rightarrow *tm*

where *W* \equiv *hmemrec* ($\lambda f \ z. \text{if } z=0 \text{ then Zero else } Q\text{-Eats } (f \ (\text{pred } z)) \ (f \ (\text{pred } z)))$)

lemma *W0* [*simp*]: *W 0 = Zero*

by (*rule trans [OF def-hmemrec [OF W-def]] auto*)

lemma *W-succ* [*simp*]: *Ord i* \Longrightarrow *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* \Longrightarrow *WR x y* \longleftrightarrow *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* \Rightarrow *hf* \Rightarrow *hf* \Rightarrow *hf* \Rightarrow *bool*

where *SeqHR x x' s k* \equiv

$\text{BuildSeq2 } (\lambda y \ y'. \text{Ord } y \wedge \text{WR } y \ y')$
 $(\lambda u \ u' \ v \ v' \ w \ w'. u = \langle v, w \rangle \wedge u' = q\text{-HPair } v' \ w') \ s \ k \ x \ x'$

8.2.1 Defining the syntax: quantified body

nominal-function $SeqHRP :: tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$

where $\llbracket atom\ l \ \# \ (s,k,sl,sl',m,n,sm,sm',sn,sn') ;$
 $atom\ sl \ \# \ (s,sl',m,n,sm,sm',sn,sn') ;$
 $atom\ sl' \ \# \ (s,m,n,sm,sm',sn,sn') ;$
 $atom\ m \ \# \ (s,n,sm,sm',sn,sn') ;$
 $atom\ n \ \# \ (s,sm,sm',sn,sn') ;$
 $atom\ sm \ \# \ (s,sm',sn,sn') ;$
 $atom\ sm' \ \# \ (s,sn,sn') ;$
 $atom\ sn \ \# \ (s,sn') ;$
 $atom\ sn' \ \# \ (s) \rrbracket \implies$

$SeqHRP\ x\ x'\ s\ k =$

$LstSeqP\ s\ k\ (HPair\ x\ x')\ AND$

$All2\ l\ (SUCC\ k)\ (Ex\ sl\ (Ex\ sl'\ (HPair\ (Var\ l)\ (HPair\ (Var\ sl)\ (Var\ sl'))\ IN\ s\ AND$

$((OrdP\ (Var\ sl)\ AND\ WRP\ (Var\ sl)\ (Var\ sl'))\ OR$

$Ex\ m\ (Ex\ n\ (Ex\ sm\ (Ex\ sm'\ (Ex\ sn\ (Ex\ sn'\ (Var\ m\ IN\ Var\ l\ AND\ Var\ n\ IN\ Var\ l\ AND$

$HPair\ (Var\ m)\ (HPair\ (Var\ sm)\ (Var\ sm'))\ IN\ s\ AND$

$HPair\ (Var\ n)\ (HPair\ (Var\ sn)\ (Var\ sn'))\ IN\ s\ AND$

$Var\ sl\ EQ\ HPair\ (Var\ sm)\ (Var\ sn)\ AND$

$Var\ sl'\ EQ\ Q-HPair\ (Var\ sm')\ (Var\ sn'))))))))$

by $(auto\ simp: eqvt-def\ SeqHRP-graph-aux-def\ flip-fresh-fresh)\ (metis\ obtain-fresh)$

nominal-termination $(eqvt)$

by $lexicographic-order$

lemma

shows $SeqHRP\ fresh\ iff\ [simp]:$

$a \ \# \ SeqHRP\ x\ x'\ s\ k \longleftrightarrow a \ \# \ x \wedge a \ \# \ x' \wedge a \ \# \ s \wedge a \ \# \ k$ **(is ?thesis1)**

and $eval\ fm\ SeqHRP\ [simp]:$

$eval\ fm\ e\ (SeqHRP\ x\ x'\ s\ k) \longleftrightarrow SeqHR\ \llbracket x \rrbracket e\ \llbracket x' \rrbracket e\ \llbracket s \rrbracket e\ \llbracket k \rrbracket e$ **(is ?thesis2)**

and $SeqHRP\ sf\ [iff]:\ Sigma\ fm\ (SeqHRP\ x\ x'\ s\ k)$ **(is ?thsf)**

and $SeqHRP\ imp\ OrdP: \{ SeqHRP\ x\ y\ s\ k \} \vdash OrdP\ k$ **(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 $\llbracket s \rrbracket e$, OF LstSeq-imp-Seq-succ]
Ord-trans [of - - succ $\llbracket k \rrbracket e$]
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 $atom\ l \# (s,k,t,i,sl,sl',m,n,sm,sm',sn,sn')$
 $atom\ sl \# (s,t,i,sl',m,n,sm,sm',sn,sn')$
 $atom\ sl' \# (s,t,i,m,n,sm,sm',sn,sn')$
 $atom\ m \# (s,t,i,n,sm,sm',sn,sn')$ $atom\ n \# (s,t,i,sm,sm',sn,sn')$
 $atom\ sm \# (s,t,i,sm',sn,sn')$ $atom\ sm' \# (s,t,i,sn,sn')$
 $atom\ sn \# (s,t,i,sn')$ $atom\ sn' \# (s,t,i)$

by (*metis obtain-fresh*)

thus *?thesis*

by (*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 $HR :: hf \Rightarrow hf \Rightarrow bool$

where $HR\ x\ x' \equiv \exists s\ k. SeqHR\ x\ x'\ s\ k$

nominal-function $HRP :: tm \Rightarrow tm \Rightarrow fm$

where $\llbracket atom\ s \# (x,x',k); atom\ k \# (x,x') \rrbracket \implies$
 $HRP\ x\ x' = Ex\ s\ (Ex\ k\ (SeqHRP\ x\ x'\ (Var\ s)\ (Var\ k)))$

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]:* $a \# HRP\ x\ x' \longleftrightarrow a \# x \wedge a \# x'$ (**is** *?thesis1*)

and *eval-fm-HRP [simp]:* $eval\ fm\ e\ (HRP\ x\ x') \longleftrightarrow HR\ \llbracket x \rrbracket e\ \llbracket x' \rrbracket e$ (**is**
?thesis2)

and *HRP-sf [iff]:* $Sigma\ fm\ (HRP\ x\ x')$ (**is** *?thsf*)

proof –

obtain $s::name$ **and** $k::name$ **where** $atom\ s \# (x,x',k)$ $atom\ k \# (x,x')$

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  $x = \text{Var } m$ ], simp)
apply (rule Ex-I [where  $x = \text{Var } n$ ], simp)
apply (rule Ex-I [where  $x = \text{Var } sm$ ], simp)
apply (rule Ex-I [where  $x = \text{Var } sm'$ ], simp)
apply (rule Ex-I [where  $x = \text{Var } sn$ ], simp)
apply (rule Ex-I [where  $x = \text{Var } sn'$ ], 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*: $\{\text{SeqHRP } x \ y \ s \ u, \text{SeqHRP } x \ y' \ s' \ u'\} \vdash y' \ EQ \ y$

proof –

obtain $i::\text{name}$ and $j::\text{name}$ and $j'::\text{name}$ and $k::\text{name}$ and $k'::\text{name}$ and $l::\text{name}$

and $m::\text{name}$ and $n::\text{name}$ and $sm::\text{name}$ and $sn::\text{name}$ and $sm'::\text{name}$ and $sn'::\text{name}$

and $m2::\text{name}$ and $n2::\text{name}$ and $sm2::\text{name}$ and $sn2::\text{name}$ and $sm2'::\text{name}$ and $sn2'::\text{name}$

where *atoms*: $\text{atom } i \# (s, s', y, y')$ $\text{atom } j \# (s, s', i, x, y, y')$ $\text{atom } j' \# (s, s', i, j, x, y, y')$

$\text{atom } k \# (s, s', x, y, y', u', i, j, j')$ $\text{atom } k' \# (s, s', x, y, y', k, i, j, j')$ $\text{atom } l \# (s, s', i, j, j', k, k')$

$\text{atom } m \# (s, s', i, j, j', k, k', l)$ $\text{atom } n \# (s, s', i, j, j', k, k', l, m)$

$\text{atom } sm \# (s, s', i, j, j', k, k', l, m, n)$ $\text{atom } sn \# (s, s', i, j, j', k, k', l, m, n, sm)$

$\text{atom } sm' \# (s, s', i, j, j', k, k', l, m, n, sm, sn)$ $\text{atom } sn' \# (s, s', i, j, j', k, k', l, m, n, sm, sn, sm')$

$\text{atom } m2 \# (s, s', i, j, j', k, k', l, m, n, sm, sn, sm', sn')$ $\text{atom } n2 \# (s, s', i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2)$

$\text{atom } sm2 \# (s, s', i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2, n2)$ $\text{atom } sn2 \# (s, s', i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2, n2, sm2)$

$\text{atom } sm2' \# (s, s', i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2, n2, sm2, sn2)$ $\text{atom } sn2' \# (s, s', i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2, n2, sm2, sn2, sm2')$

by (*metis obtain-fresh*)

have $\{\text{OrdP } (\text{Var } k)\}$

$\vdash \text{All } i \ (\text{All } j \ (\text{All } j' \ (\text{All } k' \ (\text{SeqHRP } (\text{Var } i) \ (\text{Var } j) \ s \ (\text{Var } k) \ \text{IMP} \ (\text{SeqHRP } (\text{Var } i) \ (\text{Var } j') \ s' \ (\text{Var } k') \ \text{IMP} \ \text{Var } j' \ EQ \ \text{Var } j))))))$

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 Reft Reft, 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 Reft Reft, 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) }
  ⊢ 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 (λf 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*: $\text{SeqHR } x1 \ x3 \ s1 \ k1 \implies \text{SeqHR } x2 \ x4 \ s2 \ k2 \implies \exists s \ k. \text{SeqHR } \langle x1, x2 \rangle \ (q\text{-HPair } x3 \ x4) \ s \ k$

by (*auto simp*: *SeqHR-def intro*: *BuildSeq2-combine*)

lemma *HR-H*: $\text{coding-hf } x \implies \text{HR } x \ [\text{HF } x]e$

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.premis*)

then have $x12: (x1, x) \in \text{hmem-rel } (x2, x) \in \text{hmem-rel}$

by (*auto simp*: *hmem-rel-iff-hmem-eclose*)

have $co12: \text{coding-hf } x1 \ \text{coding-hf } x2$ **using** *False step x*

by (*metis Ord-ord-of coding-hf-hpair*)**+**

hence $\text{HR } x1 \ [\text{HF } x1]e \ \text{HR } x2 \ [\text{HF } x2]e$

by (*auto simp*: *x12 step*)

thus *?thesis* **using** $x \ \text{SeqHR-hpair}$

by (*auto simp*: *HR-def q-defs*)

qed

qed

Lemma 6.2

lemma *HF-quot-coding-tm*: $\text{coding-tm } t \implies \text{HF } \llbracket t \rrbracket e = \llbracket t \rrbracket$

by (*induct t rule*: *coding-tm.induct*) (*auto, simp add*: *HPair-def quot-Eats*)

lemma *HR-quot-fm*: **fixes** $A::\text{fm}$ **shows** $\text{HR } \llbracket \llbracket A \rrbracket \rrbracket e \ \llbracket \llbracket \llbracket A \rrbracket \rrbracket \rrbracket e$

by (*metis HR-H HF-quot-coding-tm coding-tm-hf quot-fm-coding*)

lemma *prove-HRP*: **fixes** $A::\text{fm}$ **shows** $\{\} \vdash \text{HRP } \llbracket A \rrbracket \ \llbracket \llbracket A \rrbracket \rrbracket$

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 $\text{KRP} :: \text{tm} \Rightarrow \text{tm} \Rightarrow \text{tm} \Rightarrow \text{fm}$

where $\text{atom } y \ \# \ (v, x, x') \implies$

$\text{KRP } v \ x \ x' = \text{Ex } y \ (\text{HRP } x \ (\text{Var } y) \ \text{AND } \text{SubstFormP } v \ (\text{Var } y) \ x \ x')$

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*]: $a \ \# \ \text{KRP } v \ x \ x' \longleftrightarrow a \ \# \ v \wedge a \ \# \ x \wedge a \ \# \ x'$

proof –

```

obtain  $y::name$  where  $atom\ y \# (v,x,x')$ 
  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  $atom\ y \# (v,x,x',t,i)$ 
    by (metis obtain-fresh)
  thus ?thesis
    by (auto simp: KRP.simps [of y])
qed

```

```

declare KRP.simps [simp del]

```

```

lemma prove-SubstFormP:  $\{\} \vdash SubstFormP\ [Var\ i]\ [[A]]\ [A]\ [A(i::=[A])]$ 
  by (auto simp: supp-conv-fresh Sigma-fm-imp-thm ground-aux-def SubstForm-quot)

```

```

lemma prove-KRP:  $\{\} \vdash KRP\ [Var\ i]\ [A]\ [A(i::=[A])]$ 
  by (auto simp: KRP.simps [of y])
    intro!: Ex-I [where  $x=[[A]]$ ] 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  $atom\ u \# (v,x,y,y')$   $atom\ u' \# (v,x,y,y',u)$ 
    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:  $\{KRP\ [Var\ i]\ [\beta]\ (Var\ j)\} \vdash Var\ j EQ [\beta(i::=[\beta])]$ 
  by (metis KRP-unique cut0 prove-KRP)

```

8.4 The Diagonal Lemma and Gdel's Theorem

```

lemma diagonal:

```

```

  obtains  $\delta$  where  $\{\} \vdash \delta$  IFF  $\alpha(i::=[\delta])$   $supp\ \delta = supp\ \alpha - \{atom\ i\}$ 

```

```

proof –

```

```

  obtain  $k::name$  and  $j::name$ 
    where  $atoms: atom\ k \# (i,j,\alpha)$   $atom\ j \# (i,\alpha)$ 

```

```

    by (metis obtain-fresh)

```

```

  def  $\beta \equiv Ex\ j\ (KRP\ [Var\ i]\ (Var\ i)\ (Var\ j)\ AND\ \alpha(i::=Var\ j))$ 

```

```

  hence  $1: \{\} \vdash \beta(i::=[\beta])$  IFF  $(Ex\ j\ (KRP\ [Var\ i]\ (Var\ i)\ (Var\ j)\ AND\ \alpha(i::=Var\ j)))(i::=[\beta])$ 

```

```

    by (metis Iff-refl)

```

```

have 2:  $\{\} \vdash (Ex\ j\ (KRP\ [Var\ i]\ (Var\ i)\ (Var\ j)\ AND\ \alpha(i ::= Var\ j)))(i ::=$ 
 $[\beta])\ IFF$ 
       $Ex\ j\ (Var\ j\ EQ\ [\beta(i ::= [\beta])]\ AND\ \alpha(i ::= Var\ j))$ 
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:  $\{\} \vdash Ex\ j\ (Var\ j\ EQ\ [\beta(i ::= [\beta])]\ AND\ \alpha(i ::= Var\ j))\ IFF\ \alpha(i ::= [\beta(i ::= [\beta])])$ 
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  $x = [\beta(i ::= [\beta])]$ ])
done
have  $supp\ (\beta(i ::= [\beta])) = supp\ \alpha - \{atom\ i\}$  using atoms
by (auto simp: fresh-at-base ground-fm-aux-def  $\beta$ -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  $\delta$  where  $\{\} \vdash \delta\ IFF\ Neg\ (PfP\ [\delta])\ \neg \{\} \vdash \delta\ \neg \{\} \vdash Neg\ \delta$ 
      eval-fm e  $\delta$  ground-fm  $\delta$ 

```

proof –

```

fix  $i :: name$ 
obtain  $\delta$  where  $\{\} \vdash \delta\ IFF\ Neg\ ((PfP\ (Var\ i))(i ::= [\delta]))$ 
      and suppd:  $supp\ \delta = supp\ (Neg\ (PfP\ (Var\ i))) - \{atom\ i\}$ 
by (metis SyntaxN.Neg diagonal)
then have diag:  $\{\} \vdash \delta\ IFF\ Neg\ (PfP\ [\delta])$ 
by simp
then have np:  $\neg \{\} \vdash \delta \wedge \neg \{\} \vdash Neg\ \delta$ 
by (metis Iff-MP-same NegNeg-D Neg-D Neg-cong assms proved-iff-proved-PfP)
then have eval-fm e  $\delta$  using hfthm-sound [where  $e = e$ , OF diag]
by simp (metis Pf-quot-imp-is-proved)
moreover have ground-fm  $\delta$  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 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]:  $H \vdash \text{VarP } [ \text{Var } i ]$ 
proof -
  have {}  $\vdash \text{VarP } [ \text{Var } i ]$ 
  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:  $\{t \text{ EQ } v, \text{VarP } v, \text{IndP } t\} \vdash \text{Fls}$ 
proof -
  obtain  $m::\text{name}$  where  $\text{atom } m \# (t,v)$ 
  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]:  $H \vdash \text{OrdP } (\text{ORD-OF } n)$ 
proof -
  have {}  $\vdash \text{OrdP } (\text{ORD-OF } n)$ 
  by (induct n) (auto simp: OrdP-SUCC-I)
```

thus *?thesis*
by (rule *thin0*)
qed

lemma *Mem-HFun-Sigma-OrdP*: $\{HPair\ t\ u\ IN\ f,\ HFun\text{-Sigma}\ f\} \vdash OrdP\ t$

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* :: $tm \Rightarrow tm \Rightarrow fm$

where $atom\ z \# (t, r) \Longrightarrow NotInDom\ t\ r = All\ z\ (Neg\ (HPair\ t\ (Var\ z)\ IN\ r))$
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*]: $a \# NotInDom\ t\ r \longleftrightarrow a \# (t, r)$

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*: $H \vdash t\ EQ\ t' \Longrightarrow H \vdash r\ EQ\ r' \Longrightarrow H \vdash NotInDom\ t\ r$
IFF $NotInDom\ t'\ r'$

by (*rule P2-cong*) *auto*

lemma *NotInDom-Zero*: $H \vdash \text{NotInDom } t \text{ Zero}$

proof –

obtain $z::\text{name}$ **where** $\text{atom } z \# t$
by (*metis obtain-fresh*)
hence $\{\} \vdash \text{NotInDom } t \text{ Zero}$
by (*auto simp: fresh-Pair*)
thus *?thesis*
by (*rule thin0*)

qed

lemma *NotInDom-Fls*: $\{\text{HPair } d \ d' \text{ IN } r, \text{NotInDom } d \ r\} \vdash A$

proof –

obtain $z::\text{name}$ **where** $\text{atom } z \# (d,r)$
by (*metis obtain-fresh*)
hence $\{\text{HPair } d \ d' \text{ IN } r, \text{NotInDom } d \ r\} \vdash \text{Fls}$
by (*auto intro!: Ex-I [where x=d']*)
thus *?thesis*
by (*metis ExFalso*)

qed

lemma *NotInDom-Contra*: $H \vdash \text{NotInDom } d \ r \implies H \vdash \text{HPair } x \ y \text{ IN } r \implies \text{insert } (x \text{ EQ } d) \ H \vdash A$

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 $\llbracket \text{atom } x \# (y,f,k,g); \text{atom } y \# (f,k,g) \rrbracket \implies$

$\text{RestrictedP } f \ k \ g =$

$g \text{ SUBS } f \text{ AND}$

$\text{All } x \ (\text{All } y \ (\text{HPair } (\text{Var } x) \ (\text{Var } y) \text{ IN } g \text{ IFF}$

$(\text{Var } x) \text{ IN } k \text{ AND } \text{HPair } (\text{Var } x) \ (\text{Var } y) \text{ IN } f))$

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*]: $a \# \text{RestrictedP } f \ k \ g \longleftrightarrow a \# f \wedge a \# k \wedge a \# g$

proof –

obtain $x::\text{name}$ **and** $y::\text{name}$ **where** $\text{atom } x \# (y,f,k,g) \ \text{atom } y \# (f,k,g)$

by (*metis obtain-fresh*)

thus *?thesis*

by *auto*

qed

lemma *subst-fm-RestrictedP* [*simp*]:

$(\text{RestrictedP } f \ k \ g)(i::=u) = \text{RestrictedP } (\text{subst } i \ u \ f) \ (\text{subst } i \ u \ k) \ (\text{subst } i \ u \ g)$

proof –
obtain $x::name$ **and** $y::name$ **where** $atom\ x \# (y,f,k,g,i,u)$ $atom\ y \# (f,k,g,i,u)$
 by (metis obtain-fresh)
thus ?thesis
 by (auto simp: RestrictedP.simps [of x y])
qed

lemma RestrictedP-cong:
 $\llbracket H \vdash f\ EQ\ f'; H \vdash k\ EQ\ A'; H \vdash g\ EQ\ g' \rrbracket$
 $\implies H \vdash RestrictedP\ f\ k\ g\ IFF\ RestrictedP\ f'\ A'\ g'$
 by (rule P3-cong) auto

lemma RestrictedP-Zero: $H \vdash RestrictedP\ Zero\ k\ Zero$

proof –
obtain $x::name$ **and** $y::name$ **where** $atom\ x \# (y,k)$ $atom\ y \# (k)$
 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** $atom\ x \# (y,s,k,s',a,b)$ $atom\ y \# (s,k,s',a,b)$
 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** $atom\ x \# (y,s,k,s')$ $atom\ y \# (s,k,s')$
 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** $atom\ x \# (y,s,k,s',a,b)$ $atom\ y \# (s,k,s',a,b)$
 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:  $H \vdash \text{RestrictedP } s \ k \ t \implies H \vdash a \ IN \ t \implies \text{insert } (a \ IN \ s) \ H \vdash A \implies H \vdash A$ 
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  $\#$  (f,k)
  shows  $H \vdash \text{Ex } s \ (\text{RestrictedP } f \ k \ (\text{Var } s))$ 

```

```

lemma cut-RestrictedP:
  assumes s: atom s  $\#$  (f,k,A) and  $\forall C \in H. \text{atom } s \ \# \ C$ 
  shows  $\text{insert } (\text{RestrictedP } f \ k \ (\text{Var } s)) \ H \vdash A \implies H \vdash A$ 
  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  $\#$  (y,s,j,k,s') atom y  $\#$  (s,j,k,s') atom z  $\#$  (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  $H \vdash \text{HFun-Sigma } r \ H \vdash \text{NotInDom } d \ r \ H \vdash \text{OrdP } d$ 
  shows  $H \vdash \text{HFun-Sigma } (\text{Eats } r \ (\text{HPair } d \ d'))$ 

```

```

lemma HFun-Sigma-single [iff]:  $H \vdash \text{OrdP } d \implies H \vdash \text{HFun-Sigma } (\text{Eats } \text{Zero} \ (\text{HPair } d \ d'))$ 
by (metis HFun-Sigma-Eats HFun-Sigma-Zero NotInDom-Zero)

```

```

lemma LstSeqP-single [iff]:  $H \vdash \text{LstSeqP } (\text{Eats } \text{Zero} \ (\text{HPair } \text{Zero } x)) \ \text{Zero } x$ 
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* } \vdash *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'* } \vdash *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'* } \vdash *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* } \vdash *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* }

\vdash *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* \Rightarrow *tm* \Rightarrow *tm* \Rightarrow *tm* \Rightarrow *fm*

where $\llbracket \text{atom } l \# (sl, s, k, j); \text{atom } sl \# (s, j) \rrbracket \implies$
 $\text{SeqHaddP } s \ j \ k \ y = \text{LstSeqP } s \ k \ y \ \text{AND}$
 $\text{HPair Zero } j \ \text{IN } s \ \text{AND}$
 $\text{All2 } l \ k \ (\text{Ex } sl \ (\text{HPair } (\text{Var } l) \ (\text{Var } sl) \ \text{IN } s \ \text{AND}$
 $\text{HPair } (\text{SUCC } (\text{Var } l)) \ (\text{SUCC } (\text{Var } sl)) \ \text{IN } s))$
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*]: $a \# \text{SeqHaddP } s \ j \ k \ y \longleftrightarrow a \# s \wedge a \# j \wedge a \# k \wedge a \# y$

proof –

obtain *l::name* **and** *sl::name* **where** $\text{atom } l \# (sl, s, k, j) \ \text{atom } sl \# (s, j)$
by (*metis obtain-fresh*)
thus *?thesis*
by *force*

qed

lemma *SeqHaddP-subst* [*simp*]:

$(\text{SeqHaddP } s \ j \ k \ y)(i::=t) = \text{SeqHaddP } (\text{subst } i \ t \ s) \ (\text{subst } i \ t \ j) \ (\text{subst } i \ t \ k)$
 $(\text{subst } i \ t \ y)$

proof –

obtain *l::name* **and** *sl::name* **where** $\text{atom } l \# (s, k, j, sl, t, i) \ \text{atom } sl \# (s, k, j, t, i)$
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* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$

where $\llbracket \text{atom } s \# (x, y, z) \rrbracket \implies$

$\text{HaddP } x \ y \ z = \text{Ex } s \ (\text{SeqHaddP } (\text{Var } s) \ x \ y \ z)$

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*]: $a \# \text{HaddP } x \ y \ z \longleftrightarrow a \# x \wedge a \# y \wedge a \# z$

proof –

obtain *s::name* **where** $\text{atom } s \# (x, y, z)$
by (*metis obtain-fresh*)
thus *?thesis*
by *force*

qed

lemma *HaddP-subst* [*simp*]: $(\text{HaddP } x \ y \ z)(i::=t) = \text{HaddP } (\text{subst } i \ t \ x) \ (\text{subst } i \ t \ y) \ (\text{subst } i \ t \ z)$

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*: $\llbracket H \vdash t\ EQ\ t'; H \vdash u\ EQ\ u'; H \vdash v\ EQ\ v' \rrbracket \implies H \vdash HaddP\ t\ u\ v\ IFF\ HaddP\ t'\ u'\ v'$
by (*rule P3-cong*) *auto*

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

lemma *HaddP-Zero2*: $H \vdash HaddP\ x\ Zero\ x$
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 $\{\} \vdash HaddP\ x\ Zero\ x$
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*: $\{HaddP\ x\ y\ z\} \vdash OrdP\ y$
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*: $\{HaddP\ x\ y\ z\} \vdash HaddP\ x\ (SUCC\ y)\ (SUCC\ z)$

9.4.2 Proving that these relations are functions

lemma *SeqHaddP-Zero-E*: $\{SeqHaddP\ s\ w\ Zero\ z\} \vdash w\ EQ\ z$
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 $\{SeqHaddP\ s\ j\ (SUCC\ k)\ y\} \vdash Ex\ y'\ (SeqHaddP\ s\ j\ k\ (Var\ y'))\ AND\ y$

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 $H \vdash SeqHaddP\ s\ j\ (SUCC\ k)\ y\ atom\ y' \# (s,j,k,y)$
shows $H \vdash Ex\ y' (SeqHaddP\ s\ j\ k\ (Var\ y')\ AND\ y\ EQ\ SUCC\ (Var\ y'))$
by (*metis SeqHaddP-SUCC-lemma* [*THEN cut1*] *assms*)

lemma *SeqHaddP-unique*: $\{OrdP\ x, SeqHaddP\ s\ w\ x\ y, SeqHaddP\ s'\ w\ x\ y'\} \vdash y' EQ\ y$

lemma *HaddP-unique*: $\{HaddP\ w\ x\ y, HaddP\ w\ x\ y'\} \vdash y' EQ\ y$

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 $\{OrdP\ x, HaddP\ w\ x\ y, HaddP\ w\ x\ y'\} \vdash y' EQ\ y$
 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** $H \vdash OrdP\ x$ **shows** $H \vdash HaddP\ Zero\ x\ x$

proof –

fix $k::name$
have $\{OrdP\ (Var\ k)\} \vdash HaddP\ Zero\ (Var\ k)\ (Var\ k)$
 by (*rule OrdInd2H* [**where** $i=k$]) (*auto intro: HaddP-Zero2 HaddP-SUCC2*
 [*THEN cut1*])
hence $\{\} \vdash OrdP\ (Var\ k)\ IMP\ HaddP\ Zero\ (Var\ k)\ (Var\ k)$
 by (*metis Imp-I*)
hence $\{\} \vdash (OrdP\ (Var\ k)\ IMP\ HaddP\ Zero\ (Var\ k)\ (Var\ k))(k::=x)$
 by (*rule Subst*) *auto*
hence $\{\} \vdash OrdP\ x\ IMP\ HaddP\ Zero\ x\ x$
 by *simp*
thus *?thesis* **using** *assms*
 by (*metis MP-same thin0*)

qed

lemma *HaddP-Zero-D1*: *insert* (*HaddP Zero x y*) $H \vdash x EQ\ y$

by (*metis Assume HaddP-imp-OrdP HaddP-Zero1 HaddP-unique* [*THEN cut2*])

rcut1)

lemma *HaddP-Zero-D2*: *insert (HaddP x Zero y) H* \vdash *x EQ y*
by (*metis Assume HaddP-Zero2 HaddP-unique [THEN cut2]*)

lemma *HaddP-SUCC-Ex2*:

assumes *H* \vdash *HaddP x (SUCC y) z atom z' # (x,y,z)*

shows *H* \vdash *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* } \vdash *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* } \vdash *HaddP (SUCC x) y (SUCC z)*

lemma *HaddP-commute*: { *HaddP x y z, OrdP x* } \vdash *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)*

\vdash *Ex i (HaddP x y (Var i) AND z EQ SUCC (Var i))*

proof –

have { *HaddP (SUCC x) y z, OrdP x* } \vdash *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* } \vdash *y' EQ y*

lemma *Mem-imp-subtract*:

lemma *HaddP-OrdP*:

assumes *H* \vdash *HaddP x y z H* \vdash *OrdP x* **shows** *H* \vdash *OrdP z*

lemma *HaddP-Mem-cancel-left*:

assumes *H* \vdash *HaddP x y' z' H* \vdash *HaddP x y z H* \vdash *OrdP x*

shows *H* \vdash *z' IN z IFF y' IN y*

lemma *HaddP-Mem-cancel-right-Mem*:
assumes $H \vdash \text{HaddP } x' y z' H \vdash \text{HaddP } x y z H \vdash x' \text{ IN } x H \vdash \text{OrdP } x$
shows $H \vdash z' \text{ IN } z$
proof –
have $H \vdash \text{OrdP } x'$
by (*metis Ord-IN-Ord assms(3) assms(4)*)
hence $H \vdash \text{HaddP } y x' z' H \vdash \text{HaddP } y x z$
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 $H \vdash \text{HaddP } k1 k2 k H \vdash \text{OrdP } k1$
 $\text{insert } (x \text{ IN } k1) H \vdash A$
 $\text{insert } (\text{Var } i \text{ IN } k2) (\text{insert } (\text{HaddP } k1 (\text{Var } i) x) H) \vdash A$
and $i: \text{atom } (i::\text{name}) \# (k1, k2, k, x, A)$ **and** $\forall C \in H. \text{atom } i \# C$
shows $\text{insert } (x \text{ IN } k) H \vdash A$

lemma *HaddP-Mem-contra*:
assumes $H \vdash \text{HaddP } x y z H \vdash z \text{ IN } x H \vdash \text{OrdP } x$
shows $H \vdash A$
proof –
obtain $i::\text{name}$ **and** $j::\text{name}$ **and** $k::\text{name}$
where $\text{atoms: atom } i \# (x, y, z) \text{ atom } j \# (i, x, y, z) \text{ atom } k \# (i, j, x, y, z)$
by (*metis obtain-fresh*)
have $\{\text{OrdP } (\text{Var } i)\} \vdash \text{All } j (\text{HaddP } (\text{Var } i) y (\text{Var } j) \text{ IMP } \text{Neg } ((\text{Var } j) \text{ IN } (\text{Var } i)))$
(is - \vdash ?scheme)
proof (*rule OrdInd2H*)
show $\{\} \vdash \text{?scheme}(i::\text{Zero})$
using *atoms by auto*
next
show $\{\} \vdash \text{All } i (\text{OrdP } (\text{Var } i) \text{ IMP } \text{?scheme} \text{ IMP } \text{?scheme}(i::\text{SUCC } (\text{Var } i)))$
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 $\{\text{OrdP } (\text{Var } i)\} \vdash (\text{HaddP } (\text{Var } i) y (\text{Var } j) \text{ IMP } \text{Neg } ((\text{Var } j) \text{ IN } (\text{Var } i)))(j::\text{z})$
by (*metis All-D*)
hence $\{\} \vdash \text{OrdP } (\text{Var } i) \text{ IMP } \text{HaddP } (\text{Var } i) y z \text{ IMP } \text{Neg } (z \text{ IN } (\text{Var } i))$
using *atoms by simp (metis Imp-I)*
hence $\{\} \vdash (\text{OrdP } (\text{Var } i) \text{ IMP } \text{HaddP } (\text{Var } i) y z \text{ IMP } \text{Neg } (z \text{ IN } (\text{Var } i)))$


```

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-contr, blast)
    apply (metis Assume Mem-SUCC-I2 OrdP-IN-SUCC-D Sym-L thin1 thin2,
blast)
  apply (blast intro: HaddP-Mem-contr Mem-SUCC-Refl OrdP-Trans)
  done
  thus ?thesis
  by (rule cut2) (auto intro: assms)

```

qed

9.5 A Shifted Sequence

nominal-function $ShiftP :: tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$

where $\llbracket atom\ x \# (x',y,z,f,del,k); atom\ x' \# (y,z,f,del,k); atom\ y \# (z,f,del,k); atom\ z \# (f,del,g,k) \rrbracket \implies$

$ShiftP\ f\ k\ del\ g =$

$All\ z\ (Var\ z\ IN\ g\ IFF$

$(Ex\ x\ (Ex\ x'\ (Ex\ y\ ((Var\ z)\ EQ\ HPair\ (Var\ x')\ (Var\ y)\ AND$

$HaddP\ del\ (Var\ x)\ (Var\ x')\ AND$

$HPair\ (Var\ x)\ (Var\ y)\ IN\ f\ AND\ Var\ x\ IN\ k))))))$

by $(auto\ simp: eqvt-def\ ShiftP-graph-aux-def\ flip-fresh-fresh)\ (metis\ obtain-fresh)$

nominal-termination $(eqvt)$

by $lexicographic-order$

lemma $ShiftP-fresh-iff\ [simp]: a \# ShiftP\ f\ k\ del\ g \longleftrightarrow a \# f \wedge a \# k \wedge a \# del \wedge a \# g$

proof –

obtain $x::name$ **and** $x':name$ **and** $y::name$ **and** $z::name$

where $atom\ x \# (x',y,z,f,del,k)\ atom\ x' \# (y,z,f,del,k)$

$atom\ y \# (z,f,del,k)\ atom\ z \# (f,del,g,k)$

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 $atom\ x \# (x',y,z,f,del,k,i,u)\ atom\ x' \# (y,z,f,del,k,i,u)$

$atom\ y \# (z,f,del,k,i,u)\ atom\ z \# (f,del,g,k,i,u)$

by $(metis\ obtain-fresh)$

thus $?thesis$

by $(auto\ simp: ShiftP.simps\ [of\ x\ x'\ y\ z])$

qed

lemma $ShiftP-Zero: \{\} \vdash ShiftP\ Zero\ k\ d\ Zero$

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*:

$\{ShiftP\ f\ k\ del\ g,\ HPair\ a\ b\ IN\ f,\ HaddP\ del\ a\ a',\ a\ IN\ k\} \vdash HPair\ a'\ b\ IN\ g$

proof –

obtain $x::name$ **and** $x':name$ **and** $y::name$ **and** $z::name$
where $atom\ x \# (x',y,z,f,del,k,a,a',b)$ $atom\ x' \# (y,z,f,del,k,a,a',b)$
 $atom\ y \# (z,f,del,k,a,a',b)$ $atom\ z \# (f,del,g,k,a,a',b)$
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 $atom\ u \# (f,k,del,a,b)$
shows $\{ShiftP\ f\ k\ del\ g,\ HPair\ a\ b\ IN\ g\} \vdash Ex\ u\ ((Var\ u)\ IN\ k\ AND\ HaddP\ del\ (Var\ u)\ a\ AND\ HPair\ (Var\ u)\ b\ IN\ f)$

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 $H \vdash ShiftP\ f\ k\ del\ g\ H \vdash a\ IN\ g$
 $atom\ x \# (x',y,a,f,del,k)$ $atom\ x' \# (y,a,f,del,k)$ $atom\ y \# (a,f,del,k)$
shows $H \vdash (Ex\ x\ (Ex\ x'\ (Ex\ y\ (a\ EQ\ HPair\ (Var\ x')\ (Var\ y)\ AND\ HaddP\ del\ (Var\ x)\ (Var\ x')\ AND\ HPair\ (Var\ x)\ (Var\ y)\ IN\ f\ AND\ Var\ x\ IN\ k))))$
(is - \vdash ?concl)

proof –

obtain $z::name$ **where** $atom\ z \# (x,x',y,f,del,g,k,a)$
by (*metis obtain-fresh*)
hence $\{ShiftP\ f\ k\ del\ g,\ a\ IN\ g\} \vdash ?concl$ **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 # (u',v,f,k,del,g,c) atom u' # (v,f,k,del,g,c) atom v # (f,k,del,g,c)*

shows

{*ShiftP f k del g,*
Neg (Ex u (Ex u' (Ex v (c EQ HPair (Var u) (Var v) AND Var u IN k AND
HaddP del (Var u) (Var u'))))))}
 \vdash *ShiftP (Eats f c) k del g*

lemma *exists-ShiftP*:

assumes *t: atom t # (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 # (x,y,z) \implies 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 # UnionP x y z \longleftrightarrow a # x \wedge a # y \wedge a # z*

proof –

obtain *i::name* **where** *atom i # (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 # (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 # 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*: $\{ \text{UnionP } x \ y \ z, a \ \text{IN } x \} \vdash a \ \text{IN } z$
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*: $\{ \text{UnionP } x \ y \ z, a \ \text{IN } y \} \vdash a \ \text{IN } z$
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*: $\{ \text{UnionP } x \ y \ z, a \ \text{IN } z \} \vdash a \ \text{IN } x \ \text{OR } a \ \text{IN } y$
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 $H \vdash \text{UnionP } x \ y \ z$
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* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $\llbracket \text{atom } g1 \ \# \ (g2, f1, k1, f2, k2, g); \text{atom } g2 \ \# \ (f1, k1, f2, k2, g) \rrbracket \Longrightarrow$
 $\text{SeqAppendP } f1 \ k1 \ f2 \ k2 \ g =$
 $(\text{Ex } g1 \ (\text{Ex } g2 \ (\text{RestrictedP } f1 \ k1 \ (\text{Var } g1) \ \text{AND}$
 $\text{ShiftP } f2 \ k2 \ k1 \ (\text{Var } g2) \ \text{AND}$
 $\text{UnionP } (\text{Var } g1) \ (\text{Var } g2) \ g)))$
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*]:
 $a \ \# \ \text{SeqAppendP } f1 \ k1 \ f2 \ k2 \ g \longleftrightarrow a \ \# \ f1 \ \wedge \ a \ \# \ k1 \ \wedge \ a \ \# \ f2 \ \wedge \ a \ \# \ k2 \ \wedge \ a \ \# \ g$
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  $H \vdash Ex\ g\ (SeqAppendP\ f1\ k1\ f2\ k2\ (Var\ g))$ 
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  $\{\} \vdash Ex\ g\ (SeqAppendP\ f1\ k1\ f2\ k2\ (Var\ g))$ 
  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:  $\{SeqAppendP\ f1\ k1\ f2\ k2\ g, HPair\ x\ y\ IN\ f1, x\ IN\ k1\} \vdash HPair\ x\ y\ IN\ g$ 
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
  by (auto simp: SeqAppendP.simps [of g1 g2] intro: UnionP-Mem1 [THEN cut2] RestrictedP-Mem [THEN cut3])

```

qed

lemma *SeqAppendP-Mem2*: $\{SeqAppendP\ f1\ k1\ f2\ k2\ g, HaddP\ k1\ x\ x', x\ IN\ k2, HPair\ x\ y\ IN\ f2\} \vdash HPair\ x'\ y\ IN\ g$

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*

by (*auto simp: SeqAppendP.simps [of g1 g2] intro: UnionP-Mem2 [THEN cut2] ShiftP-Mem1 [THEN cut4]*)

qed

lemma *SeqAppendP-Mem-E*:

assumes $H \vdash SeqAppendP\ f1\ k1\ f2\ k2\ g$

and $insert\ (HPair\ x\ y\ IN\ f1)\ (insert\ (x\ IN\ k1)\ H) \vdash A$

and $insert\ (HPair\ (Var\ u)\ y\ IN\ f2)\ (insert\ (HaddP\ k1\ (Var\ u)\ x)\ (insert\ (Var\ u\ IN\ k2)\ H)) \vdash A$

and $u: atom\ u \# (f1, k1, f2, k2, x, y, g, A) \forall C \in H. atom\ u \# C$

shows $insert\ (HPair\ x\ y\ IN\ g)\ H \vdash A$

9.8 LstSeqP and SeqAppendP

lemma *HDomain-Incl-SeqAppendP*: — The And eliminates the need to prove *cut5*

$\{SeqAppendP\ f1\ k1\ f2\ k2\ g, HDomain-Incl\ f1\ k1\ AND\ HDomain-Incl\ f2\ k2, HaddP\ k1\ k2\ k, OrdP\ k1\} \vdash HDomain-Incl\ g\ k$

declare *SeqAppendP.simps [simp del]*

lemma *HFun-Sigma-SeqAppendP*:

$\{SeqAppendP\ f1\ k1\ f2\ k2\ g, HFun-Sigma\ f1, HFun-Sigma\ f2, OrdP\ k1\} \vdash HFun-Sigma\ g$

lemma *LstSeqP-SeqAppendP*:

assumes $H \vdash SeqAppendP\ f1\ (SUCC\ k1)\ f2\ (SUCC\ k2)\ g$

$H \vdash LstSeqP\ f1\ k1\ y1\ H \vdash LstSeqP\ f2\ k2\ y2\ H \vdash HaddP\ k1\ k2\ k$

shows $H \vdash LstSeqP\ g\ (SUCC\ k)\ y2$

proof –

have $\{SeqAppendP\ f1\ (SUCC\ k1)\ f2\ (SUCC\ k2)\ g, LstSeqP\ f1\ k1\ y1, LstSeqP\ f2\ k2\ y2, HaddP\ k1\ k2\ k\}$

$\vdash LstSeqP\ g\ (SUCC\ k)\ y2$

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  $\{VarP\ v\} \vdash Ex\ s\ (Ex\ k\ (SeqStTermP\ v\ i\ v\ i\ (Var\ s)\ (Var\ k)))$ 
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  $\{VarP\ v, VarP\ w, Neg\ (v\ EQ\ w)\} \vdash Ex\ s\ (Ex\ k\ (SeqStTermP\ v\ i\ w\ w\ (Var\ s)\ (Var\ k)))$ 
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 $\{VarP\ v\} \vdash Ex\ s\ (Ex\ k\ (SeqStTermP\ v\ i\ Zero\ Zero\ (Var\ s)\ (Var\ k)))$
corollary *SubstTermP-Zero*: $\{TermP\ t\} \vdash SubstTermP\ [Var\ v]\ t\ Zero\ Zero$
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*: $\{VarP\ v,\ TermP\ t\} \vdash SubstTermP\ v\ t\ v\ t$
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*: $\{VarP\ v,\ VarP\ w,\ Neg\ (v\ EQ\ w),\ TermP\ t\} \vdash SubstTermP\ v\ t\ w\ w$
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 $\{VarP\ v,\ IndP\ t\} \vdash Ex\ s\ (Ex\ k\ (SeqStTermP\ v\ i\ t\ t\ (Var\ s)\ (Var\ k)))$
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*: $\{VarP\ v, IndP\ w, TermP\ t\} \vdash SubstTermP\ v\ t\ w\ w$

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 $\{SeqStTermP\ v\ i\ t1\ u1\ s1\ k1, SeqStTermP\ v\ i\ t2\ u2\ s2\ k2\}$
 $\vdash Ex\ s\ (Ex\ k\ (SeqStTermP\ v\ i\ (Q-Eats\ t1\ t2)\ (Q-Eats\ u1\ u2)\ (Var\ s)$
 $(Var\ k)))$

theorem *SubstTermP-Eats*:

$\{SubstTermP\ v\ i\ t1\ u1, SubstTermP\ v\ i\ t2\ u2\} \vdash SubstTermP\ v\ i\ (Q-Eats\ t1\ t2)\ (Q-Eats\ u1\ u2)$

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 $\{SeqConstP\ s\ k\ c\}$

$\vdash c\ EQ\ Zero\ OR$

$Ex\ m\ (Ex\ n\ (Ex\ sm\ (Ex\ sn\ (Var\ m\ IN\ k\ AND\ Var\ n\ IN\ k\ AND$
 $SeqConstP\ s\ (Var\ m)\ (Var\ sm)\ AND$

$SeqConstP\ s\ (Var\ n)\ (Var\ sn)\ AND$
 $c\ EQ\ Q-Eats\ (Var\ sm)\ (Var\ sn))))$

lemma *SeqConstP-imp-SubstTermP*: $\{SeqConstP\ s\ kk\ c,\ TermP\ t\} \vdash SubstTermP$
 $[Var\ w]\ t\ c\ c$

theorem *SubstTermP-Const*: $\{ConstP\ c,\ TermP\ t\} \vdash SubstTermP\ [Var\ w]\ t\ c\ c$

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*:
 $\{SubstTermP\ v\ i\ x\ x',\ SubstTermP\ v\ i\ y\ y'\} \vdash SubstAtomicP\ v\ i\ (Q-Mem\ x\ y)$
 $(Q-Mem\ x'\ y')$

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 $\{SubstTermP\ v\ i\ x\ x',\ SubstTermP\ v\ i\ y\ y'\}$
 $\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q-Mem\ x\ y)\ (Q-Mem\ x'\ y')\ (Var\ s)$
 $(Var\ k)))$

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'}  $\vdash$  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'}  $\vdash$  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 $sk: \text{atom } s \# (k, x, y, x', y', v, i) \text{ atom } k \# (x, y, x', y', v, i)$
shows $\{ \text{SubstTermP } v \ i \ x \ x', \text{ SubstTermP } v \ i \ y \ y' \}$
 $\vdash \text{Ex } s \ (\text{Ex } k \ (\text{SeqSubstFormP } v \ i \ (Q\text{-Eq } x \ y) \ (Q\text{-Eq } x' \ y') \ (\text{Var } s) \ (\text{Var } k)))$

proof –
let $?vs = (s, k, x, y, x', y', v, i)$
obtain $l::\text{name}$ **and** $sl::\text{name}$ **and** $sl'::\text{name}$ **and** $m::\text{name}$ **and** $n::\text{name}$ **and** $sm::\text{name}$ **and** $sm'::\text{name}$ **and** $sn::\text{name}$ **and** $sn'::\text{name}$
where $\text{atom } l \# (?vs, sl, sl', m, n, sm, sm', sn, sn')$
 $\text{atom } sl \# (?vs, sl', m, n, sm, sm', sn, sn')$ $\text{atom } sl' \# (?vs, m, n, sm, sm', sn, sn')$
 $\text{atom } m \# (?vs, n, sm, sm', sn, sn')$ $\text{atom } n \# (?vs, sm, sm', sn, sn')$
 $\text{atom } sm \# (?vs, sm', sn, sn')$ $\text{atom } sm' \# (?vs, sn, sn')$
 $\text{atom } sn \# (?vs, sn')$ $\text{atom } sn' \# ?vs$
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*:
 $\{ \text{SubstTermP } v \ i \ x \ x', \text{ SubstTermP } v \ i \ y \ y' \} \vdash \text{SubstFormP } v \ i \ (Q\text{-Eq } x \ y) \ (Q\text{-Eq } x' \ y')$

proof –
obtain $k1::\text{name}$ **and** $s1::\text{name}$ **and** $k2::\text{name}$ **and** $s2::\text{name}$ **and** $k::\text{name}$ **and** $s::\text{name}$
where $\text{atom } s1 \# (v, i, x, y, x', y')$ $\text{atom } k1 \# (v, i, x, y, x', y', s1)$
 $\text{atom } s2 \# (v, i, x, y, x', y', k1, s1)$ $\text{atom } k2 \# (v, i, x, y, x', y', s2, k1, s1)$
 $\text{atom } s \# (v, i, x, y, x', y', k2, s2, k1, s1)$ $\text{atom } k \# (v, i, x, y, x', y', s, k2, s2, k1, s1)$

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 $\{SeqSubstFormP\ v\ i\ x\ x'\ s1\ k1, TermP\ i, VarP\ v\}$

$\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q-Neg\ x)\ (Q-Neg\ x')\ (Var\ s)\ (Var\ k)))$

theorem *SubstFormP-Neg*: $\{SubstFormP\ v\ i\ x\ x'\} \vdash SubstFormP\ v\ i\ (Q-Neg\ x)\ (Q-Neg\ x')$

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 $\{SeqSubstFormP\ v\ i\ x\ x'\ s1\ k1,$

$SeqSubstFormP\ v\ i\ y\ y'\ s2\ k2, TermP\ i, VarP\ v\}$

$\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q-Disj\ x\ y)\ (Q-Disj\ x'\ y')\ (Var\ s)\ (Var\ k)))$

theorem *SubstFormP-Disj*:

$\{SubstFormP\ v\ i\ x\ x', SubstFormP\ v\ i\ y\ y'\} \vdash SubstFormP\ v\ i\ (Q-Disj\ x\ y)\ (Q-Disj\ x'\ y')$

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 $\{SeqSubstFormP\ v\ i\ x\ x'\ s1\ k1,\ TermP\ i,\ VarP\ v\}$
 $\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q-Ex\ x)\ (Q-Ex\ x')\ (Var\ s)\ (Var\ k)))$
theorem *SubstFormP-Ex*: $\{SubstFormP\ v\ i\ x\ x'\} \vdash SubstFormP\ v\ i\ (Q-Ex\ x)$
 $(Q-Ex\ x')$
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*: $\{\} \vdash ConstP\ Zero$
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 $\{SeqConstP\ s1\ k1\ t1,\ SeqConstP\ s2\ k2\ t2\}$
 $\vdash Ex\ s\ (Ex\ k\ (SeqConstP\ (Var\ s)\ (Var\ k)\ (Q-Eats\ t1\ t2)))$

theorem *ConstP-Eats*: $\{ConstP\ t1,\ ConstP\ t2\} \vdash ConstP\ (Q-Eats\ t1\ t2)$

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 $atom\ s\ \#\ (k,s1,s2,k1,k2,\alpha1,\alpha2,\beta)$ $atom\ k\ \#\ (s1,s2,k1,k2,\alpha1,\alpha2,\beta)$
shows $\{PrfP\ s1\ k1\ \alpha1,\ PrfP\ s2\ k2\ \alpha2,\ ModPonP\ \alpha1\ \alpha2\ \beta\ OR\ ExistsP\ \alpha1\ \beta$
 $OR\ SubstP\ \alpha1\ \beta\}$
 $\vdash Ex\ k\ (Ex\ s\ (PrfP\ (Var\ s)\ (Var\ k)\ \beta))$

corollary *PfP-inference*: $\{PfP\ \alpha1,\ PfP\ \alpha2,\ ModPonP\ \alpha1\ \alpha2\ \beta\ OR\ ExistsP\ \alpha1$
 $\beta\ OR\ SubstP\ \alpha1\ \beta\} \vdash PfP\ \beta$

proof –
obtain $k1::name$ **and** $s1::name$ **and** $k2::name$ **and** $s2::name$ **and** $k::name$ **and** $s::name$
where $atom\ s1 \# (\alpha1, \alpha2, \beta)$ $atom\ k1 \# (\alpha1, \alpha2, \beta, s1)$
 $atom\ s2 \# (\alpha1, \alpha2, \beta, k1, s1)$ $atom\ k2 \# (\alpha1, \alpha2, \beta, s2, k1, s1)$
 $atom\ s \# (\alpha1, \alpha2, \beta, k2, s2, k1, s1)$
 $atom\ k \# (\alpha1, \alpha2, \beta, s, k2, s2, k1, s1)$
by (*metis obtain-fresh*)
thus *?thesis*
apply (*simp add: Pfp.simps [of k s β] Pfp.simps [of k1 s1 $\alpha1$] Pfp.simps [of k2 s2 $\alpha2$]*)
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 $H \vdash Pfp\ y$ $H \vdash SubstFormP\ x\ t\ y\ z$
shows $H \vdash Pfp\ z$

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*: $\llbracket H \vdash Pfp\ (Q\ Imp\ x\ y); H \vdash Pfp\ x \rrbracket \implies H \vdash Pfp\ y$
by (*force intro: Pfp-inference [of x, THEN cut3] Disj-I1 simp add: ModPonP-def*)

corollary *Pfp-implies-ModPon-Pfp-quot*: $\llbracket H \vdash Pfp\ [\alpha\ IMP\ \beta]; H \vdash Pfp\ [\alpha] \rrbracket \implies H \vdash Pfp\ [\beta]$
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 ∨ Q i} = {f i |i. P i} ∪ {f i |i. Q i}
by auto
```

```
abbreviation Q-Subset :: tm ⇒ tm ⇒ tm
  where Q-Subset t u ≡ (Q-All (Q-Imp (Q-Mem (Q-Ind Zero) t) (Q-Mem (Q-Ind
Zero) u)))
```

```
lemma NEQ-quot-tm: i ≠ j ⇒ { } ⊢ [Var i] NEQ [Var j]
  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 ≠ j ⇒ insert ([Var i] EQ [Var j]) H ⊢ Fls
  by (metis (full-types) NEQ-quot-tm Assume OrdNotEqP-E cut2 thin0)
```

```
lemma perm-commute: a ‡ p ⇒ a' ‡ p ⇒ (a ⇔ a') + p = p + (a ⇔ a')
  by (rule plus-perm-eq) (simp add: supp-swap fresh-def)
```

```
lemma perm-self-inverseI: [¬p = q; a ‡ p; a' ‡ p] ⇒ - ((a ⇔ a') + p) = (a ⇔
a') + q
  by (simp-all add: perm-commute fresh-plus-perm minus-add)
```

```
lemma fresh-image:
  fixes f :: 'a ⇒ 'b::fs shows finite A ⇒ i ‡ f ' A ↔ (∀ x∈A. i ‡ f x)
  by (induct rule: finite-induct) (auto simp: fresh-finite-insert)
```

```

lemma atom-in-atom-image [simp]:  $\text{atom } j \in \text{atom } 'V \longleftrightarrow j \in V$ 
  by auto

lemma fresh-star-empty [simp]:  $\{\} \#* bs$ 
  by (simp add: fresh-star-def)

declare fresh-star-insert [simp]

lemma fresh-star-finite-insert:
  fixes  $S :: ('a::fs) \text{ set}$  shows  $\text{finite } S \implies a \#* \text{insert } x S \longleftrightarrow a \#* x \wedge a \#* S$ 
  by (auto simp: fresh-star-def fresh-finite-insert)

lemma fresh-finite-Diff-single [simp]:
  fixes  $V :: \text{name set}$  shows  $\text{finite } V \implies a \# (V - \{j\}) \longleftrightarrow (a \# j \longrightarrow a \# V)$ 
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]:  $\text{finite } A \implies i \# \text{atom } 'A \longleftrightarrow i \# A$ 
  by (induct rule: finite-induct) (auto simp: fresh-finite-insert)

lemma atom-fresh-star-atom-set-conv:  $\llbracket \text{atom } i \# bs; \text{finite } bs \rrbracket \implies bs \#* i$ 
by (metis fresh-finite-atom-set fresh-ineq-at-base fresh-star-def)

lemma notin-V:
  assumes  $p: \text{atom } i \# p$  and  $V: \text{finite } V \text{ atom } ' (p \cdot V) \#* V$ 
  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 ::  $tm \Rightarrow \text{name set} \Rightarrow (\text{name} \Rightarrow tm) \Rightarrow tm$ 
  where  $\text{ssubst } t V F = \text{Finite-Set.fold } (\lambda i. \text{subst } i (F i)) t V$ 

definition make-F ::  $\text{name set} \Rightarrow \text{perm} \Rightarrow \text{name} \Rightarrow tm$ 
  where  $\text{make-F } Vs p \equiv \lambda i. \text{if } i \in Vs \text{ then } \text{Var } (p \cdot i) \text{ else } \text{Var } i$ 

lemma ssubst-empty [simp]:  $\text{ssubst } t \{\} F = t$ 
  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  $p: atom \ ' (p \cdot Vs) \#^* Vs$ 
  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  $atom-fresh-perm: \llbracket x \in Vs; y \in Vs \rrbracket \Longrightarrow atom\ x \# p \cdot y$ 
  by ( $metis\ imageI\ Vs\ p\ fresh-finite-set-at-base\ fresh-star-def\ mem-permute-iff\ fresh-at-base(2)$ )

lemma  $fresh-pj: \llbracket a \# p; j \in Vs \rrbracket \Longrightarrow a \# p \cdot j$ 
  by ( $metis\ atom-fresh-perm\ fresh-at-base(2)\ fresh-perm\ fresh-permute-left\ pinv$ )

lemma  $fresh-Vs: a \# p \Longrightarrow a \# Vs$ 
  by ( $metis\ Vs\ fresh-def\ fresh-perm\ fresh-permute-iff\ fresh-star-def\ p\ permute-finite\ supp-finite-set-at-base$ )

lemma  $fresh-pVs: a \# p \Longrightarrow a \# p \cdot Vs$ 
  by ( $metis\ fresh-Vs\ fresh-perm\ fresh-permute-left\ pinv$ )

lemma assumes  $V \subseteq Vs\ a \# p$ 
  shows  $fresh-pV\ [simp]: a \# p \cdot V$  and  $fresh-V\ [simp]: a \# V$ 
  using  $fresh-pVs\ fresh-Vs\ assms$ 
  apply ( $auto\ simp: fresh-def$ )
  apply ( $metis\ (full-types)\ Vs\ finite-V\ permute-finite\ set-mp\ subset-Un-eq\ supp-of-finite-union\ union-eqt$ )
  by ( $metis\ Vs\ finite-V\ set-mp\ subset-Un-eq\ supp-of-finite-union$ )

lemma  $qp-insert:$ 
  fixes  $i::name$  and  $i'::name$ 
  assumes  $atom\ i \# p\ atom\ i' \# (i,p)$ 
  shows  $quote-perm\ ((atom\ i \Leftrightarrow atom\ i') + p)\ (insert\ i\ Vs)$ 
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* ($\lambda i. subst\ i\ (F\ i)$)

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 \implies

$ssubst\ t\ (insert\ i\ V)\ F = (if\ i \in V\ then\ ssubst\ t\ V\ F$
 $else\ subst\ i\ (F\ i)\ (ssubst\ t\ V\ F))$

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* $\implies ssubst\ Zero\ V\ F = Zero$

by (*induct V rule: finite-induct*) (*auto simp: ssubst-insert*)

lemma *ssubst-Eats* [*simp*]: *finite V* $\implies ssubst\ (Eats\ t\ u)\ V\ F = Eats\ (ssubst\ t\ V\ F)\ (ssubst\ u\ V\ F)$

by (*induct V rule: finite-induct*) (*auto simp: ssubst-insert*)

lemma *ssubst-SUCC* [*simp*]: *finite V* $\implies ssubst\ (SUCC\ t)\ V\ F = SUCC\ (ssubst\ t\ V\ F)$

by (*metis SUCC-def ssubst-Eats*)

lemma *ssubst-ORD-OF* [*simp*]: *finite V* $\implies ssubst\ (ORD-OF\ n)\ V\ F = ORD-OF\ n$

by (*induction n*) *auto*

lemma *ssubst-HPair* [*simp*]:
 $finite\ V \implies ssubst\ (HPair\ t\ u)\ V\ F = HPair\ (ssubst\ t\ V\ F)\ (ssubst\ u\ V\ F)$
by (*simp add: HPair-def*)

lemma *ssubst-HTuple* [*simp*]: $finite\ V \implies ssubst\ (HTuple\ n)\ V\ F = (HTuple\ n)$
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 $finite\ V\ a \# p \cdot V\ a \# t$
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 $finite\ V\ atom\ i \# t\ atom\ (p \cdot i) \# t$
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*:
 $\llbracket finite\ V; atom\ i \# p \cdot V \rrbracket$
 $\implies ssubst\ [Ex\ i\ A]\ (insert\ i\ V)\ (insert\ i\ V)\ F = ssubst\ [Ex\ i\ A]\ V\ V\ F$
by (*simp add: ssubst-insert-if insert-absorb vquot-fm-insert fresh-ssubst*)

lemma *ground-ssubst-eq*: $\llbracket finite\ V; supp\ t = \{\} \rrbracket \implies ssubst\ t\ V\ F = t$
by (*induct V rule: finite-induct*) (*auto simp: ssubst-insert fresh-def*)

lemma *ssubst-quot-tm* [*simp*]:
fixes $t::tm$ **shows** $finite\ V \implies ssubst\ [t]\ V\ F = [t]$
by (*simp add: ground-ssubst-eq supp-conv-fresh*)

lemma *ssubst-quot-fm* [*simp*]:
fixes $A::fm$ **shows** $finite\ V \implies ssubst\ [A]\ V\ F = [A]$
by (*simp add: ground-ssubst-eq supp-conv-fresh*)

lemma *atom-in-p-Vs*: $\llbracket i \in p \cdot V; V \subseteq Vs \rrbracket \implies i \in p \cdot Vs$
by (*metis (full-types) True-eqvt set-mp subset-eqvt*)

10.3 The Main Theorems of Section 7

lemma *SubstTermP-vquot-dbtm*:

assumes $w: w \in Vs - V$ **and** $V: V \subseteq Vs$ $V' = p \cdot V$
and $s: \text{supp } dbtm \subseteq \text{atom } ' Vs$

shows

$\text{insert } (ConstP (F w)) \{ConstP (F i) \mid i. i \in V\}$
 $\vdash \text{SubstTermP } [Var w] (F w)$
 $(\text{ssubst } (vquot-dbtm V dbtm) V F)$
 $(\text{subst } w (F w) (\text{ssubst } (vquot-dbtm (\text{insert } w V) dbtm) V F))$

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** $i \notin Vs$ **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 $w: w \in V_s - V$ **and** $V: V \subseteq V_s$ $V' = p \cdot V$
and $s: \text{supp } \text{dbfm} \subseteq \text{atom } ' V_s$
shows
 $\text{insert } (\text{ConstP } (F w)) \{ \text{ConstP } (F i) \mid i. i \in V \}$
 $\vdash \text{SubstFormP } [\text{Var } w] (F w)$
 $(\text{ssubst } (\text{vquot-dbfm } V \text{ dbfm}) V F)$
 $(\text{subst } w (F w) (\text{ssubst } (\text{vquot-dbfm } (\text{insert } w V) \text{ dbfm}) V F))$

using $w s$
proof (*induct dbfm rule: dbfm.induct*)
case ($\text{DBMem } t u$) **thus** ?*case using* V
by (*auto intro: SubstTermP-vquot-dbtm SubstFormP-Mem [THEN cut2]*)
next
case ($\text{DBEq } t u$) **thus** ?*case using* V
by (*auto intro: SubstTermP-vquot-dbtm SubstFormP-Eq [THEN cut2]*)
next
case ($\text{DBDisj } A B$) **thus** ?*case using* V
by (*auto intro: SubstFormP-Disj [THEN cut2]*)
next
case ($\text{DBNeg } A$) **thus** ?*case using* V
by (*auto intro: SubstFormP-Neg [THEN cut1]*)
next
case ($\text{DBEx } A$) **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 $w: w \in V_s - V$ **and** $V: V \subseteq V_s$ $V' = p \cdot V$
and $s: \text{supp } A \subseteq \text{atom } ' V_s$
shows
 $\text{insert } (\text{ConstP } (F w)) \{ \text{ConstP } (F i) \mid i. i \in V \}$
 $\vdash \text{SubstFormP } [\text{Var } w] (F w)$
 $(\text{ssubst } [A] V V F)$
 $(\text{ssubst } [A] (\text{insert } w V) (\text{insert } w V) F)$

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 $\beta::fm$
assumes $\beta: \{ \} \vdash \text{PfP } [\beta]$
and $V: V \subseteq V_s$
and $s: \text{supp } \beta \subseteq \text{atom } ' V_s$
shows $\{ \text{ConstP } (F i) \mid i. i \in V \} \vdash \text{PfP } (\text{ssubst } [\beta] V V F)$

```

proof –
  show ?thesis using finite-V [OF V] V
  proof induction
    case empty thus ?case
    by (auto simp:  $\beta$ )
  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  $\Rightarrow$  hf  $\Rightarrow$  hf  $\Rightarrow$  hf  $\Rightarrow$  bool
where SeqQuote x x' s k  $\equiv$ 
  BuildSeq2 ( $\lambda y y'. y=0 \wedge y' = 0$ )
  ( $\lambda u u' v v' w w'. u = v \triangleleft w \wedge u' = q\text{-Eats } v' w'$ ) s k x x'
```

11.1.1 Defining the syntax: quantified body

```
nominal-function SeqQuoteP :: tm  $\Rightarrow$  tm  $\Rightarrow$  tm  $\Rightarrow$  tm  $\Rightarrow$  fm
where  $\llbracket$  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  $\rrbracket \Longrightarrow$ 
  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
```

$$\text{HPair } (Var\ n) \ (\text{HPair } (Var\ sn) \ (Var\ sn')) \ \text{IN } s \ \text{AND}$$

$$\text{Var } sl \ \text{EQ } \text{Eats } (Var\ sm) \ (Var\ sn) \ \text{AND}$$

$$\text{Var } sl' \ \text{EQ } \text{Q-Eats } (Var\ sm') \ (Var\ sn'))))))))$$
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*]:

$a \# \text{SeqQuoteP } t \ u \ s \ k \longleftrightarrow a \# t \wedge a \# u \wedge a \# s \wedge a \# k$ (**is** *?thesis1*)

and *eval-fm-SeqQuoteP* [*simp*]:

$\text{eval-fm } e \ (\text{SeqQuoteP } t \ u \ s \ k) \longleftrightarrow \text{SeqQuote } \llbracket t \rrbracket e \ \llbracket u \rrbracket e \ \llbracket s \rrbracket e \ \llbracket k \rrbracket e$ (**is** *?thesis2*)

and *SeqQuoteP-sf* [*iff*]:

$\text{Sigma-fm } (\text{SeqQuoteP } t \ u \ s \ k)$ (**is** *?thsf*)

and *SeqQuoteP-imp-OrdP*:

$\{ \text{SeqQuoteP } t \ u \ s \ k \} \vdash \text{OrdP } k$ (**is** *?thord*)

and *SeqQuoteP-imp-LstSeqP*:

$\{ \text{SeqQuoteP } t \ u \ s \ k \} \vdash \text{LstSeqP } s \ k \ (\text{HPair } t \ u)$ (**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*:

$\text{atom } l \# (s, k, sl, sl', m, n, sm, sm', sn, sn')$

$\text{atom } sl \# (s, sl', m, n, sm, sm', sn, sn')$ $\text{atom } sl' \# (s, m, n, sm, sm', sn, sn')$

$\text{atom } m \# (s, n, sm, sm', sn, sn')$ $\text{atom } n \# (s, sm, sm', sn, sn')$

$\text{atom } sm \# (s, sm', sn, sn')$ $\text{atom } sm' \# (s, sn, sn')$

$\text{atom } sn \# (s, sn')$ $\text{atom } sn' \# s$

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 $\llbracket s \rrbracket e$, OF LstSeq-imp-Seq-succ]

Ord-trans [of - - succ $\llbracket k \rrbracket e$]

cong: conj-cong)

qed

lemma *SeqQuoteP-subst* [*simp*]:

$(\text{SeqQuoteP } t \ u \ s \ k)(j::=w) =$

$\text{SeqQuoteP } (\text{subst } j \ w \ t) \ (\text{subst } j \ w \ u) \ (\text{subst } j \ w \ s) \ (\text{subst } j \ w \ k)$

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 $\text{atom } l \# (s, k, w, j, sl, sl', m, n, sm, sm', sn, sn')$

$\text{atom } sl \# (s, w, j, sl', m, n, sm, sm', sn, sn')$ $\text{atom } sl' \# (s, w, j, m, n, sm, sm', sn, sn')$

$\text{atom } m \# (s, w, j, n, sm, sm', sn, sn')$ $\text{atom } n \# (s, w, j, sm, sm', sn, sn')$

$\text{atom } sm \# (s, w, j, sm', sn, sn')$ $\text{atom } sm' \# (s, w, j, sn, sn')$

```

      atom sn # (s,w,j,sn') atom sn' # (s,w,j)
    by (metis obtain-fresh)
  thus ?thesis
    by (force simp add: 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  $x = \text{Var } sm \uparrow$ ], simp)
apply (rule Ex-I [where  $x = \text{Var } sn$ ], simp)
apply (rule Ex-I [where  $x = \text{Var } sn \uparrow$ ], simp)
apply (simp-all add: SeqQuoteP.simps [of  $l\ s - sl\ sl'\ m\ n\ sm\ sm'\ sn\ sn \uparrow$ ])
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* :: $hf \Rightarrow hf \Rightarrow \text{bool}$
where *Quote* $x\ x' \equiv \exists s\ k. \text{SeqQuote } x\ x'\ s\ k$

11.2.1 Defining the syntax

nominal-function *QuoteP* :: $tm \Rightarrow tm \Rightarrow \text{fm}$
where $\llbracket \text{atom } s \ \sharp (t,u,k); \text{atom } k \ \sharp (t,u) \rrbracket \Longrightarrow$
QuoteP $t\ u = \text{Ex } s (\text{Ex } k (\text{SeqQuoteP } t\ u (\text{Var } s) (\text{Var } k)))$
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*]: $a \ \sharp \text{QuoteP } t\ u \longleftrightarrow a \ \sharp t \wedge a \ \sharp u$ (**is** *?thesis1*)
and *eval-fm-QuoteP* [*simp*]: $\text{eval-fm } e (\text{QuoteP } t\ u) \longleftrightarrow \text{Quote } \llbracket t \rrbracket e \llbracket u \rrbracket e$ (**is** *?thesis2*)

and *QuoteP-sf* [*iff*]: *Sigma-fm* (*QuoteP* $t\ u$) (**is** *?thsf*)

proof —

obtain $s::\text{name}$ **and** $k::\text{name}$ **where** $\text{atom } s \ \sharp (t,u,k)$ $\text{atom } k \ \sharp (t,u)$

by (*metis obtain-fresh*)

thus *?thesis1* *?thesis2* *?thsf*

by (*auto simp: Quote-def*)

qed

lemma *QuoteP-subst* [*simp*]:

$(\text{QuoteP } t\ u)(j::w) = \text{QuoteP } (\text{subst } j\ w\ t) (\text{subst } j\ w\ u)$

proof —

obtain $s::\text{name}$ **and** $k::\text{name}$ **where** $\text{atom } s \ \sharp (t,u,w,j,k)$ $\text{atom } k \ \sharp (t,u,w,j)$

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  $x = \text{SUCC} (\text{SUCC} (\text{Var } k'))$ ])
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  $t2$   $u2$   $s2$   $k2$ }
     $\vdash$  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  $x = \text{Eats } t1 \ t2$ ])
using assms atoms apply simp
apply (rule Ex-I [where  $x = \text{Q-Eats } u1 \ u2$ ], 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  $x = \text{SUCC} (\text{Var } k')$ ], 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*: $\{QuoteP\ t1\ u1, QuoteP\ t2\ u2\} \vdash QuoteP\ (Eats\ t1\ t2)$
 $(Q-Eats\ u1\ u2)$

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',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  $\sharp$  (k, ?scheme) using atoms
    by simp
next
  show {}  $\vdash$  All k (OrdP (Var k) IMP (All2 l (Var k) (?scheme(k::= Var l))
IMP ?scheme))
    apply (rule All-I Imp-I)+
    using atoms
    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)}
   $\vdash$  (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)}  $\vdash$  All j' (SeqQuoteP x (Var j') (Var s) (Var k) IMP
ConstP (Var j'))
  using atoms by simp
hence {OrdP(Var k)}  $\vdash$  (SeqQuoteP x (Var j') (Var s) (Var k) IMP ConstP
(Var j')) (j'::=y)
  by (metis All-D)
hence {OrdP(Var k)}  $\vdash$  SeqQuoteP x y (Var s) (Var k) IMP ConstP y
  using atoms by simp
hence {SeqQuoteP x y (Var s) (Var k)}  $\vdash$  ConstP y
  by (metis Imp-cut SeqQuoteP-imp-OrdP anti-deduction)
thus {QuoteP x y}  $\vdash$  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** $atom\ s' \# (k',t,u,s,k)$ $atom\ k' \# (t,u,s,k)$
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* :: $[perm, name\ set] \Rightarrow fm\ set$

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]*: $finite\ V \Longrightarrow i \# quote-all\ p\ V \longleftrightarrow i \# V \wedge i \# p \cdot V$

by (*induct rule: finite-induct*) (*auto simp: fresh-finite-insert*)

lemma *fresh-quote-all-mem*: $\llbracket A \in quote-all\ p\ V; finite\ V; i \# V; i \# p \cdot V \rrbracket \Longrightarrow i \# A$

by (*metis Set.set-insert finite-insert finite-quote-all fresh-finite-insert fresh-quote-all*)

lemma *quote-all-perm-eq*:

assumes $finite\ V$ $atom\ i \# (p, V)$ $atom\ i' \# (p, V)$

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  $quote\text{-}all\ p\ Us \vdash A$ 
proof -
  show ?thesis using finite-V [OF  $\langle Us \subseteq Vs \rangle$ ] 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  $\beta: \{\} \vdash \beta$ 
  and  $V: V \subseteq Vs$ 
  and  $s: supp\ \beta \subseteq atom\ 'Vs$ 
  shows  $quote\text{-}all\ p\ V \vdash PfP (ssubst\ [\beta]\ V\ V\ F)$ 
proof -
  have  $\{\} \vdash PfP\ [\beta]$ 
  by (metis  $\beta$  proved-iff-proved-PfP)
  hence  $\{ConstP (F i) \mid i. i \in V\} \vdash PfP (ssubst\ [\beta]\ V\ V\ F)$ 
  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  $A: \{\} \vdash \alpha\ IMP\ \beta$ 
  and  $V: V \subseteq Vs$ 
  and  $s: supp\ \alpha \subseteq atom\ 'Vs\ supp\ \beta \subseteq atom\ 'Vs$ 
  shows  $quote\text{-}all\ p\ V \vdash PfP (ssubst\ [\alpha]\ V\ V\ F)\ IMP\ PfP (ssubst\ [\beta]\ V\ V\ F)$ 
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 $A: \{\} \vdash \alpha 1 \text{ IMP } \alpha 2 \text{ IMP } \beta$

and $V: V \subseteq Vs$

and $s: \text{supp } \alpha 1 \subseteq \text{atom } ' Vs \text{ supp } \alpha 2 \subseteq \text{atom } ' Vs \text{ supp } \beta \subseteq \text{atom } ' Vs$

shows $\text{quote-all } p \ V \vdash \text{PfP } (\text{ssubst } [\alpha 1] \ V \ V \ F) \ \text{IMP} \ \text{PfP } (\text{ssubst } [\alpha 2] \ V \ V \ F) \ \text{IMP} \ \text{PfP } (\text{ssubst } [\beta] \ V \ V \ F)$

using *quote-all-PfP-ssubst [OF A V] s*

by (*force simp: V quot-fm-def intro: PfP-implies-ModPon-PfP [OF PfP-implies-ModPon-PfP] thin1*)

lemma *quote-all-Disj-I1-PfP-ssubst:*

assumes $V \subseteq Vs \text{ supp } \alpha \subseteq \text{atom } ' Vs \text{ supp } \beta \subseteq \text{atom } ' Vs$

and *prems:* $H \vdash \text{PfP } (\text{ssubst } [\alpha] \ V \ V \ F) \ \text{quote-all } p \ V \subseteq H$

shows $H \vdash \text{PfP } (\text{ssubst } [\alpha \ \text{OR} \ \beta] \ V \ V \ F)$

proof –

have $\{\} \vdash \alpha \ \text{IMP} \ (\alpha \ \text{OR} \ \beta)$

by (*blast intro: Disj-I1*)

hence $\text{quote-all } p \ V \vdash \text{PfP } (\text{ssubst } [\alpha] \ V \ V \ F) \ \text{IMP} \ \text{PfP } (\text{ssubst } [\alpha \ \text{OR} \ \beta] \ V \ V \ F)$

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 $V \subseteq Vs \text{ supp } \alpha \subseteq \text{atom } ' Vs \text{ supp } \beta \subseteq \text{atom } ' Vs$

and *prems:* $H \vdash \text{PfP } (\text{ssubst } [\beta] \ V \ V \ F) \ \text{quote-all } p \ V \subseteq H$

shows $H \vdash \text{PfP } (\text{ssubst } [\alpha \ \text{OR} \ \beta] \ V \ V \ F)$

proof –

have $\{\} \vdash \beta \ \text{IMP} \ (\alpha \ \text{OR} \ \beta)$

by (*blast intro: Disj-I2*)

hence $\text{quote-all } p \ V \vdash \text{PfP } (\text{ssubst } [\beta] \ V \ V \ F) \ \text{IMP} \ \text{PfP } (\text{ssubst } [\alpha \ \text{OR} \ \beta] \ V \ V \ F)$

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 $V \subseteq Vs \text{ supp } \alpha \subseteq \text{atom } ' Vs \text{ supp } \beta \subseteq \text{atom } ' Vs$

and *prems:* $H \vdash \text{PfP } (\text{ssubst } [\alpha] \ V \ V \ F) \ H \vdash \text{PfP } (\text{ssubst } [\beta] \ V \ V \ F) \ \text{quote-all } p \ V \subseteq H$

shows $H \vdash \text{PfP } (\text{ssubst } [\alpha \ \text{AND} \ \beta] \ V \ V \ F)$

proof –

have $\{\} \vdash \alpha \ \text{IMP} \ \beta \ \text{IMP} \ (\alpha \ \text{AND} \ \beta)$

by *blast*

hence $\text{quote-all } p \ V$

$\vdash \text{PfP } (\text{ssubst } [\alpha] \ V \ V \ F) \ \text{IMP} \ \text{PfP } (\text{ssubst } [\beta] \ V \ V \ F) \ \text{IMP} \ \text{PfP } (\text{ssubst } [\alpha \ \text{AND} \ \beta] \ V \ V \ F)$

```

[ $\alpha$  AND  $\beta$ ] 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 [ $\alpha$ ] V V F) IMP PfP (ssubst [Neg  $\alpha$ ] V V F) IMP PfP
(ssubst [Fls] 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:  $\llbracket atom\ i \# p \cdot V; V \subseteq Vs \rrbracket \implies atom\ i \# ssubst\ (vquot-dbtm\ V\ t)\ V\ F$ 
  by (induct t rule: dbtm.induct) (auto simp: F-unfold fresh-image permute-set-eq-image)

lemma fresh-ssubst-dbfm:  $\llbracket atom\ i \# p \cdot V; V \subseteq Vs \rrbracket \implies atom\ i \# ssubst\ (vquot-dbfm\ V\ A)\ V\ F$ 
  by (nominal-induct A rule: dbfm.strong-induct) (auto simp: fresh-ssubst-dbtm)

lemma fresh-ssubst-fm:
  fixes  $A::fm$  shows  $\llbracket atom\ i \# p \cdot V; V \subseteq Vs \rrbracket \implies atom\ i \# ssubst\ ([A]\ V)\ V\ F$ 
  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  $atom\ i \# (j, j', i', si, ki, sj, kj)$   $atom\ i' \# (j, j', si, ki, sj, kj)$ 
     $atom\ j \# (j', si, ki, sj, kj)$   $atom\ j' \# (si, ki, sj, kj)$ 
     $atom\ si \# (ki, sj, kj)$   $atom\ sj \# (ki, kj)$ 
  shows {SeqQuoteP (Var i) (Var i') (Var si) ki, SeqQuoteP (Var j) (Var j') (Var sj) kj}
     $\vdash (Var\ i\ IN\ Var\ j\ IMP\ PfP\ (Q-Mem\ (Var\ i')\ (Var\ j')))$  AND
     $(Var\ i\ SUBS\ Var\ j\ IMP\ PfP\ (Q-Subset\ (Var\ i')\ (Var\ j')))$ 
  proof -
    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:  $atom\ lj \# (li, l, i, j, j', i', si, ki, sj, kj, i, i', k, m, n, sm, sm', sn, sn')$ 

```



```

atom li # (l,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 Reft, 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 - Reft, 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]:  $sm \in V' sn \in V' sm \notin Vs sn \notin Vs$  using atoms'
  by (auto simp: V'-def fresh-finite-set-at-base [symmetric])
hence Q-All2-Eats: quote-all p' V'
   $\vdash$  PfP (ssubst [All2 i (Var sm)  $\alpha$ ] V' V' F') IMP
  PfP (ssubst [ $\alpha(i::=Var sn)$ ] V' V' F') IMP
  PfP (ssubst [All2 i (Eats (Var sm) (Var sn))  $\alpha$ ] 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:  $\bigwedge name. name \in Vs \implies F' name = F name$ 
  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  $\subseteq$  atom ' V' supp t  $\subseteq$  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  $\subseteq$  atom ' V' supp A  $\subseteq$  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]  $\alpha$ )) V' F' =
  ssubst (vquot-dbfm Vs (trans-fm [i]  $\alpha$ )) 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}  $\subseteq$  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}  $\subseteq$  atom ' Vs
  hence subst i' (Var sn') (ssubst (vquot-dbfm (insert i Vs) A) (insert i Vs) Fi)
=

```

$$\text{ssubst } (v\text{quot-dbfm } V' (\text{subst-dbfm } (DBVar \text{ sn}) i A)) V' F'$$
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*
 $(\alpha \text{ IMP PfP } (\text{ssubst } [\alpha](\text{insert } i \text{ Vs}) (\text{insert } i \text{ Vs}) Fi))$

using *IH* **by** *auto*

hence *quote-all p Vs*
 \vdash (*QuoteP* (*Var i*) (*Var i'*) *IMP*
 $(\alpha \text{ IMP PfP } (\text{ssubst } [\alpha](\text{insert } i \text{ Vs}) (\text{insert } i \text{ Vs}) Fi))) (i' ::= \text{Var}$

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*
 $(\alpha \text{ IMP PfP } (\text{subst } i' (\text{Var sn}') (\text{ssubst } [\alpha](\text{insert } i \text{ Vs}) (\text{insert } i \text{ Vs}) Fi)))$

using *atoms* **by** *simp*

moreover **have** *subst i' (Var sn')* $(\text{ssubst } [\alpha](\text{insert } i \text{ Vs}) (\text{insert } i \text{ Vs}) Fi)$
 $= \text{ssubst } [\alpha(i ::= \text{Var sn})] V' V' F'$

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* $(\alpha \text{ IMP PfP } (\text{ssubst } [\alpha(i ::= \text{Var sn})] V'$

V' F'))

by *simp*

hence *quote-all p Vs*
 \vdash (*QuoteP* (*Var i*) (*Var sn')* *IMP* $(\alpha \text{ IMP PfP } (\text{ssubst } [\alpha(i ::= \text{Var sn})] V'$

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*
 $(\alpha(i ::= \text{Var sn}) \text{ IMP PfP } (\text{subst } i (\text{Var sn}) (\text{ssubst } [\alpha(i ::= \text{Var sn})] V' V'$

F')))

using *atoms atoms'* **by** *simp*

moreover **have** *subst i (Var sn)* $(\text{ssubst } [\alpha(i ::= \text{Var sn})] V' V' F')$
 $= \text{ssubst } [\alpha(i ::= \text{Var sn})] V' V' F'$

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* $(\alpha(i ::= \text{Var sn}) \text{ IMP PfP } (\text{ssubst}$

$[\alpha(i ::= \text{Var sn})] V' V' F'))$

using *atoms atoms'* **by** *simp*

hence *star0: insert (QuoteP (Var sn) (Var sn')) (quote-all p Vs)*
 $\vdash \alpha(i ::= \text{Var sn}) \text{ IMP PfP } (\text{ssubst } [\alpha(i ::= \text{Var sn})] V' V' F')$

by (*rule anti-deduction*)

```

have subst-i-star: quote-all p' V' ⊢ α(i ::= Var sn) IMP Pfp (ssubst [α(i ::= Var
sn)] 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}))
  ⊢ All j (All j' (SeqQuoteP (Var j) (Var j') (Var s) (Var k) IMP
          All2 i (Var j) α IMP Pfp (ssubst [All2 i (Var j) α] Vs Vs F)))
  (is - ⊢ ?scheme)
proof (rule OrdIndH [where j=l])
  show atom l # (k, ?scheme) using atoms atoms' j j' fresh-pVs
  by (simp add: fresh-Pair F-unfold)
next
have substj: ∧ t j. atom j # α ⇒ atom (p · j) # α ⇒
          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 ⊆ 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 ∈ Vs atom sm # p · Vs atom sm' # p · Vs atom sn # p · Vs atom
sn' # p · Vs
    using atoms' Vs by (auto simp: fresh-pVs)
  hence atom sm # p · w atom sm' # p · w atom sn # p · w atom sn' # p · 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')} ∪
          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')}

```

```

      ∪ 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')}
      ∪ quote-all p (Vs - {j}) .
  have pjV: p · j ∉ Vs
  by (metis j perm-exits-Vs)
  hence jpV: atom j # p · Vs
  by (simp add: fresh-permute-left pinv fresh-finite-set-at-base)
  show quote-all p (Vs - {j}) ⊢ 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

```

$ss\text{-noprimes } j'$
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 $PfP\ (ssubst\ [All2\ i\ (Var\ sm)\ \alpha]\ V'\ V'\ F')$
apply (force intro!: Imp-E [THEN rotate3] simp add: vquot-fm-def substj j')
 $ss\text{-noprimes}$
— Proving $PfP\ (ssubst\ [\alpha(i::=Var\ sn)]\ V'\ V'\ F')$
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 $i\ (Var\ j)\ \alpha\ IMP\ PfP\ (ssubst\ [All2\ i\ (Var\ j)\ \alpha]\ Vs\ Vs\ F))$)
($j::=Var\ j$)
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 $i\ (Var\ j)\ \alpha\ IMP\ PfP\ (ssubst\ [All2\ i\ (Var\ j)\ \alpha]\ Vs\ Vs\ F))\ (j'::=Var\ j)$
 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 $p\ Vs$)
 $\vdash\ \alpha\ IMP\ PfP\ (ssubst\ [\alpha](insert\ i\ Vs)\ (insert\ i\ Vs)\ Fi)$
and supp (All2 $i\ (Var\ j)\ \alpha) \subseteq atom\ 'Vs$
and j : atom $j \# (i, \alpha)$ **and** i : atom $i \# p$ **and** i' : atom $i' \# (i, p, \alpha)$
and pi : $pi = (atom\ i \rightleftharpoons atom\ i') + p$
and Fi : $Fi = make-F\ (insert\ i\ Vs)\ pi$
shows insert (All2 $i\ (Var\ j)\ \alpha$) (quote-all $p\ Vs$) $\vdash\ PfP\ (ssubst\ [All2\ i\ (Var\ j)\ \alpha]\ Vs\ Vs\ F)$
proof —
have sp : supp $\alpha - \{atom\ i\} \subseteq atom\ 'Vs$ **and** jV : $j \in Vs$
using assms
by (auto simp: fresh-def supp-Pair)
obtain $s::name$ **and** $k::name$
where atoms: atom $s \# (k, i, j, p \cdot j, \alpha, p)$ atom $k \# (i, j, p \cdot j, \alpha, p)$
by (metis obtain-fresh)

```

hence  $ii$ :  $atom\ i\ \sharp\ (j, p \cdot j, s, k, p)$  using  $i\ j$ 
  by ( $simp\ add$ :  $fresh\ Pair$ ) ( $metis\ fresh\ at\ base(2)\ fresh\ perm\ fresh\ permute\ left$ 
 $pinv$ )
have  $jj$ :  $atom\ j\ \sharp\ (p \cdot j, s, k, \alpha)$  using  $atoms\ j$ 
  by ( $auto\ simp$ :  $fresh\ Pair$ ) ( $metis\ atom\ fresh\ perm\ jV$ )
have  $pj$ :  $atom\ (p \cdot j)\ \sharp\ (s, k, \alpha)$  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  $A = QuoteP\ (Var\ j)\ (Var\ (p \cdot j))$ ])
  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 $SpecI$: $H \vdash A \text{ IMP } Ex\ i\ A$

by ($metis\ Imp\ I\ Assume\ Ex\ I\ subst\ fm\ id$)

lemma $star$:

fixes $p :: perm$ **and** $F :: name \Rightarrow tm$

assumes C : $ss\ fm\ \alpha$

and p : $atom\ ' (p \cdot V)\ \sharp^*\ V\ -p = p$

and V : $finite\ V\ supp\ \alpha \subseteq atom\ ' V$

and F : $F = make\ F\ V\ p$

shows $insert\ \alpha\ (quote\ all\ p\ V) \vdash Pfp\ (ssubst\ [\alpha]\ V\ V\ F)$

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 : $atom\ i\ \sharp\ j\ \{i, j\} \subseteq V$ **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 $insert\ (Var\ i\ IN\ Var\ j)\ (quote\ all\ p\ V) \vdash Pfp\ (Q\ Mem\ (Var\ (p \cdot i))$
 $(Var\ (p \cdot j)))$

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' ≡ (atom i ⇔ atom i') + p
  def F' ≡ make-F (insert i V) p'
  have p'-apply [simp]: !!v. p' · v = (if v=i then i' else if v=i' then i else p · 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' · V = p · V
  by (metis i' p'-def permute-plus fresh-Pair qp.fresh-pVs swap-fresh-fresh (atom
i # p))
  have i: i ∉ V i ∉ p · V atom i # V atom i # p · V atom i # p' · 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 ⊆ V i ∉ W i' ∉ 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'  $\implies$  Quote z z'  $\implies$  Quote (y  $\triangleleft$  z) (q-Eats y' z')
  by (auto simp: Quote-def SeqQuote-def intro: BuildSeq2-combine)
```

```
lemma Quote-q-Succ [intro]: Quote y y'  $\implies$  Quote (succ y) (q-Succ y')
```

```
  by (auto simp: succ-def q-Succ-def)
```

```
lemma HR-imp-eq-H: HR x z  $\implies$  z =  $\llbracket$ HF x $\rrbracket$ 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  $\implies$  Ord x  $\implies$  WR x y
```

```
  by (metis HF-Ord HR-imp-eq-H WR-iff-eq-W)
```

```
lemma WR-Quote: WR (ord-of i) y  $\implies$  Quote (ord-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-Eats x y
```

```
  by (simp add: q-Eats-def)
```

```
lemma HR-imp-Quote: coding-hf x  $\implies$  HR x y  $\implies$  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*:
 $[[\{\} \vdash \alpha \text{ IMP } \beta; \text{ground-fm } \alpha; \text{ground-fm } \beta]] \implies \{\text{PfP } [\alpha]\} \vdash \text{PfP } [\beta]$
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*: $\text{ground-fm } \alpha \implies \{\} \vdash \text{PfP } [\alpha] \text{ IMP } \text{PfP } [\text{Neg } \alpha] \text{ IMP } \text{PfP } [\text{Fls}]$
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 $\neg \{\} \vdash \text{Fls}$
shows $\neg \{\} \vdash \text{Neg } (\text{PfP } [\text{Fls}])$
proof –
from *assms Goedel-I* **obtain** δ
where *diag*: $\{\} \vdash \delta \text{ IFF } \text{Neg } (\text{PfP } [\delta]) \neg \{\} \vdash \delta$
and *gnd*: $\text{ground-fm } \delta$
by *metis*
have $\{\text{PfP } [\delta]\} \vdash \text{PfP } [\text{PfP } [\delta]]$
by *(auto simp: Provability ground-fm-aux-def supp-conv-fresh)*
moreover **have** $\{\text{PfP } [\delta]\} \vdash \text{PfP } [\text{Neg } (\text{PfP } [\delta])]$
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** $\text{ground-fm } (\text{PfP } [\delta])$
by *(auto simp: ground-fm-aux-def supp-conv-fresh)*
ultimately **have** $\{\text{PfP } [\delta]\} \vdash \text{PfP } [\text{Fls}]$ **using** *PfP-quot-contra*
by *(metis (no-types) anti-deduction cut2)*
thus $\neg \{\} \vdash \text{Neg } (\text{PfP } [\text{Fls}])$
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>.