

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

Andrei Popescu Dmitriy Traytel

February 23, 2021

Abstract

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

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

Contents

1	Syntax of Terms and Formulas using Nominal Logic	5
1.1	Terms and Formulas	5
1.1.1	Hf is a pure permutation type	5
1.1.2	The datatypes	5
1.1.3	Substitution	6
1.1.4	Derived syntax	7
1.1.5	Derived logical connectives	8
1.2	Axioms and Theorems	9
1.2.1	Logical axioms	9
1.2.2	Concrete variables	9
1.2.3	The HF axioms	10
1.2.4	Equality axioms	10
1.2.5	The proof system	10
1.2.6	Derived rules of inference	10
1.2.7	The Deduction Theorem	13
1.2.8	Cut rules	14
1.3	Miscellaneous logical rules	15
1.3.1	Quantifier reasoning	18
1.3.2	Congruence rules	19
1.4	Equality reasoning	20
1.4.1	The congruence property for (<i>EQ</i>), and other basic properties of equality	20
1.4.2	The congruence property for (<i>IN</i>)	21
1.4.3	The congruence properties for <i>Eats</i> and <i>HPair</i>	22
1.4.4	Substitution for Equalities	23
1.4.5	Congruence Rules for Predicates	23
1.5	Zero and Falsity	24
1.5.1	The Formula <i>Fls</i>	25
1.5.2	More properties of <i>Zero</i>	26
1.5.3	Basic properties of <i>Eats</i>	27
1.6	Bounded Quantification involving <i>Eats</i>	29
1.7	Induction	29
2	De Bruijn Syntax, Quotations, Codes, V-Codes	31
2.1	de Bruijn Indices (locally-nameless version)	31
2.2	Characterising the Well-Formed de Bruijn Formulas	34
2.2.1	Well-Formed Terms	34
2.2.2	Well-Formed Formulas	35
2.3	Well formed terms and formulas (de Bruijn representation)	36
2.4	Quotations	38
2.4.1	Quotations of de Bruijn terms	38
2.4.2	Quotations of de Bruijn formulas	39
2.5	Definitions Involving Coding	40

2.5.1	The set Γ of Definition 1.1, constant terms used for coding	41
2.6	V-Coding for terms and formulas, for the Second Theorem	41
3	Basic Predicates	44
3.1	The Subset Relation	44
3.2	Extensionality	46
3.3	The Disjointness Relation	47
3.4	The Foundation Theorem	50
3.5	The Ordinal Property	51
3.6	Induction on Ordinals	54
3.7	Linearity of Ordinals	55
3.8	The predicate <i>OrdNotEqP</i>	56
3.9	Predecessor of an Ordinal	57
3.10	Case Analysis and Zero/SUCC Induction	58
3.11	The predicate <i>HFun_Sigma</i>	59
3.12	The predicate <i>HDomain_Incl</i>	61
3.13	<i>HPair</i> is Provably Injective	62
3.14	<i>SUCC</i> is Provably Injective	64
3.15	The predicate <i>LstSeqP</i>	65
4	Sigma-Formulas and Theorem 2.5	67
4.1	Ground Terms and Formulas	67
4.2	Sigma Formulas	68
4.2.1	Strict Sigma Formulas	68
4.2.2	Closure properties for Sigma-formulas	68
4.3	Lemma 2.2: Atomic formulas are Sigma-formulas	69
4.4	Universal Quantification Bounded by an Arbitrary Term	73
4.5	Lemma 2.3: Sequence-related concepts are Sigma-formulas	73
5	Predicates for Terms, Formulas and Substitution	75
5.1	Predicates for atomic terms	75
5.1.1	Free Variables	75
5.1.2	De Bruijn Indexes	75
5.1.3	Various syntactic lemmas	76
5.2	The predicate <i>SeqCTermP</i> , for Terms and Constants	76
5.3	The predicates <i>TermP</i> and <i>ConstP</i>	77
5.3.1	Definition	77
5.3.2	Correctness properties for constants	78
5.4	Abstraction over terms	78
5.4.1	Defining the syntax: main predicate	80
5.5	Substitution over terms	80
5.5.1	Defining the syntax	80
5.6	Abstraction over formulas	81
5.6.1	The predicate <i>AbstAtomicP</i>	81
5.6.2	The predicate <i>AbsMakeForm</i>	82
5.6.3	Defining the syntax: the main <i>AbstForm</i> predicate	83
5.7	Substitution over formulas	84
5.7.1	The predicate <i>SubstAtomicP</i>	84
5.7.2	The predicate <i>SubstMakeForm</i>	85
5.7.3	Defining the syntax: the main <i>SubstForm</i> predicate	86
5.8	The predicate <i>AtomicP</i>	87
5.9	The predicate <i>MakeForm</i>	87
5.10	The predicate <i>SeqFormP</i>	88
5.11	The predicate <i>FormP</i>	89

5.11.1	Definition	89
5.11.2	The predicate <i>VarNonOccFormP</i> (Derived from <i>SubstFormP</i>)	89
6	Formalizing Provability	90
6.1	Section 4 Predicates (Leading up to Pf)	90
6.1.1	The predicate <i>SentP</i> , for the Sentential (Boolean) Axioms	90
6.1.2	The predicate <i>Equality_axP</i> , for the Equality Axioms	90
6.1.3	The predicate <i>HF_axP</i> , for the HF Axioms	91
6.1.4	The specialisation axioms	91
6.1.5	The induction axioms	91
6.1.6	The predicate <i>AxiomP</i> , for any Axioms	92
6.1.7	The predicate <i>ModPonP</i> , for the inference rule Modus Ponens	92
6.1.8	The predicate <i>ExistsP</i> , for the existential rule	93
6.1.9	The predicate <i>SubstP</i> , for the substitution rule	93
6.1.10	The predicate <i>PrfP</i>	94
6.1.11	The predicate <i>PfP</i>	95
7	Syntactic Preliminaries for the Second Incompleteness Theorem	96
7.1	NotInDom	97
7.2	Restriction of a Sequence to a Domain	98
7.3	Applications to LstSeqP	99
7.4	Ordinal Addition	100
7.4.1	Predicate form, defined on sequences	100
7.4.2	Proving that these relations are functions	102
7.5	A Shifted Sequence	105
7.6	Union of Two Sets	107
7.7	Append on Sequences	109
7.8	LstSeqP and SeqAppendP	110
7.9	Substitution and Abstraction on Terms	111
7.9.1	Atomic cases	111
7.9.2	Non-atomic cases	113
7.9.3	Substitution over a constant	114
7.10	Substitution on Formulas	114
7.10.1	Membership	114
7.10.2	Equality	115
7.10.3	Negation	116
7.10.4	Disjunction	116
7.10.5	Existential	117
7.11	Constant Terms	117
7.12	Proofs	119
7.13	Formulas	120
7.14	Abstraction on Formulas	120
7.14.1	Membership	120
7.14.2	Equality	121
7.14.3	Negation	122
7.14.4	Disjunction	123
7.14.5	Existential	123
7.14.6	MakeForm	127
7.14.7	Negation	128
7.14.8	Disjunction	128
7.14.9	Existential	128

8 Pseudo-Coding: Section 7 Material	137
8.1 General Lemmas	137
8.2 Simultaneous Substitution	138
8.3 The Main Theorems of Section 7	141
9 Quotations of the Free Variables	143
9.1 Sequence version of the “Special p-Function, F^* ”	143
9.1.1 Defining the syntax: quantified body	143
9.1.2 Correctness properties	144
9.2 The “special function” itself	145
9.2.1 Correctness properties	146
9.3 The Operator <i>quote_all</i>	151
9.3.1 Definition and basic properties	151
9.3.2 Transferring theorems to the level of derivability	152
9.4 Star Property. Equality and Membership: Lemmas 9.3 and 9.4	154
9.5 Star Property. Universal Quantifier: Lemma 9.7	160
9.6 The Derivability Condition, Theorem 9.1	165
10 Uniqueness Results: Syntactic Relations are Functions	169
10.0.1 <i>SeqStTermP</i>	169
10.0.2 <i>SubstAtomicP</i>	173
10.0.3 <i>SeqSubstFormP</i>	173
10.0.4 <i>SubstFormP</i>	177
11 Section 6 Material and Gödel’s First Incompleteness Theorem	179
11.1 The Function W and Lemma 6.1	179
11.1.1 Predicate form, defined on sequences	179
11.1.2 Predicate form of W	180
11.1.3 Proving that these relations are functions	182
11.2 The Function HF and Lemma 6.2	184
11.2.1 Defining the syntax: quantified body	184
11.2.2 Defining the syntax: main predicate	185
11.2.3 Proving that these relations are functions	185
11.3 The Function K and Lemma 6.3	189
12 The Instantiation	191

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 \sharp *subst* *i* *x* *t* \iff (*atom* *i* \sharp *t* \wedge *j* \sharp *t*) \vee (*j* \sharp *x* \wedge (*j* \sharp *t* \vee *j* = *atom* *i*))

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

lemma *forget_subst_tm* [*simp*]: *atom* *a* \sharp *tm* \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* \sharp *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* \sharp *t* \implies *atom* *i* \sharp *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* \sharp (*i*, *x*) \implies (*Ex* *j* *A*)(*i*::=*x*) = *Ex* *j* (*A*(*i*::=*x*))

apply (*simp add: eqvt_def subst_fm_graph_aux_def*)

apply *auto* [16]

apply (*rule_tac* *y=a* **and** *c=(aa, b)* **in** *fm.strong_exhaust*)

apply (*auto simp: eqvt_at_def fresh_star_def fresh_Pair fresh_at_base*)

apply (*metis flip_at_base_simps(3) flip_fresh_fresh*)

done

nominal_termination (*eqvt*)

by *lexicographic_order*

lemma *size_subst_fm* [*simp*]: *size* (*A*(*i*::=*x*)) = *size* *A*

by (*nominal_induct* *A* *avoiding: i x* *rule: fm.strong_induct*) *auto*

lemma *forget_subst_fm* [*simp*]: *atom* *a* \sharp *A* \implies *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*]:
 $j \# (A(i::=x)) \longleftrightarrow (atom\ i \# A \wedge j \# A) \vee (j \# x \wedge (j \# A \vee j = atom\ i))$
by (*nominal_induct* *A* *avoiding*: *i x* *rule*: *fm.strong_induct*) (*auto simp*: *fresh_at_base*)

lemma *subst_fm_commute* [*simp*]:
 $atom\ j \# A \implies (A(i::=t))(j::=u) = A(i ::= subst\ j\ u\ t)$
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*:
 $atom\ i' \# (A, i, j, t) \implies (Ex\ i\ A)(j ::= t) = Ex\ i' \# (((i \leftrightarrow i') \cdot A)(j ::= t))$
by (*rule* *subst* [*of* *Ex* *i'* ($(i \leftrightarrow i') \cdot A$) *Ex* *i* *A*])
(auto simp: *Abs1_eq_iff_flip_def* *swap_commute*)

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

simproc_setup *subst_fm_renaming* ((*Ex* *i* *A*)(*j* ::= *t*)) = $\langle fn\ _ \Rightarrow fn\ ctxt \Rightarrow fn\ ctrm \Rightarrow$
let
 $val\ _ \$ (_ \$ i \$ A) \$ j \$ t = Thm.term_of\ ctrm$

 $val\ atoms = Simplifier.prem_of\ ctxt$
 $|> map_filter\ (fn\ thm \Rightarrow case\ Thm.prop_of\ thm\ of$
 $\ \ \ \ \ _ \$ (Const\ (@\{const_name\ fresh\}, _) \$ atm \$ _) \Rightarrow SOME\ (atm) | _ \Rightarrow NONE)$
 $|> distinct\ ((=))$

 $fun\ get_thm\ atm =$
let
 $val\ goal = HOLogic.mk_Trueprop\ (mk_fresh\ atm\ (HOLogic.mk_tuple\ [A, i, j, t]))$
in
 $SOME\ ((Goal.prove\ ctxt\ [] []\ goal\ (K\ (asm_full_simp_tac\ ctxt\ 1)))$
 $\ \ RS\ @\{thm\ subst_fm_Ex_with_renaming\}\ RS\ eq_reflection)$
 $handle\ ERROR\ _ \Rightarrow NONE$
end
in
 $get_first\ get_thm\ atoms$
end
 \rangle

1.1.4 Derived syntax

Ordered pairs

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

lemma *HPair_eqvt* [*eqvt*]: $(p \cdot HPair\ a\ b) = HPair\ (p \cdot a)\ (p \cdot b)$
by (*auto simp*: *HPair_def*)

lemma *fresh_HPpair* [*simp*]: $x \# HPair\ a\ b \longleftrightarrow (x \# a \wedge x \# b)$
by (*auto simp*: *HPair_def*)

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

lemma *subst_tm_HPpair* [*simp*]: $subst\ i\ x\ (HPair\ a\ b) = HPair\ (subst\ i\ x\ a)\ (subst\ i\ x\ b)$
by (*auto simp*: *HPair_def*)

Ordinals

definition

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

fun $ORD_OF :: nat \Rightarrow tm$

where

$ORD_OF 0 = Zero$
 $| ORD_OF (Suc k) = SUCC (ORD_OF k)$

lemma $SUCC_fresh_iff$ [simp]: $a \# SUCC t \longleftrightarrow a \# t$
by (simp add: $SUCC_def$)

lemma $SUCC_eqvt$ [eqvt]: $(p \cdot SUCC a) = SUCC (p \cdot a)$
by (simp add: $SUCC_def$)

lemma $SUCC_subst$ [simp]: $subst i t (SUCC k) = SUCC (subst i t k)$
by (simp add: $SUCC_def$)

lemma ORD_OF_fresh [simp]: $a \# ORD_OF n$
by (induct n) (auto simp: $SUCC_def$)

lemma ORD_OF_eqvt [eqvt]: $(p \cdot ORD_OF n) = ORD_OF (p \cdot n)$
by (induct n) (auto simp: $permutate_pure SUCC_eqvt$)

1.1.5 Derived logical connectives

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

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

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

Conjunction

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

lemma $Conj_eqvt$ [eqvt]: $p \cdot (A AND B) = (p \cdot A) AND (p \cdot B)$
by (simp add: $Conj_def$)

lemma $fresh_Conj$ [simp]: $a \# A AND B \longleftrightarrow (a \# A \wedge a \# B)$
by (auto simp: $Conj_def$)

lemma $supp_Conj$ [simp]: $supp (A AND B) = supp A \cup supp B$
by (auto simp: $Conj_def$)

lemma $size_Conj$ [simp]: $size (A AND B) = size A + size B + 4$
by (simp add: $Conj_def$)

lemma $Conj_injective_iff$ [iff]: $(A AND B) = (A' AND B') \longleftrightarrow (A = A' \wedge B = B')$
by (auto simp: $Conj_def$)

lemma $subst_fm_Conj$ [simp]: $(A AND B)(i::=x) = (A(i::=x) AND (B(i::=x)))$
by (auto simp: $Conj_def$)

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*)

1.2 Axioms and Theorems

1.2.1 Logical axioms

inductive_set *boolean_axioms* :: *fm set*
where

- Ident*: $A\ IMP\ A \in boolean_axioms$
- DisjI1*: $A\ IMP\ (A\ OR\ B) \in boolean_axioms$
- DisjCont*: $(A\ OR\ A)\ IMP\ A \in boolean_axioms$
- DisjAssoc*: $(A\ OR\ (B\ OR\ C))\ IMP\ ((A\ OR\ B)\ OR\ C) \in boolean_axioms$
- DisjConj*: $(C\ OR\ A)\ IMP\ ((Neg\ C)\ OR\ B)\ IMP\ (A\ OR\ B) \in boolean_axioms$

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

inductive_set *induction_axioms* :: *fm set* **where**
ind:

- atom* ($j::name$) $\# (i, A)$
- $\implies A(i::=Zero)\ IMP\ ((All\ i\ (All\ j\ (A\ IMP\ (A(i::=Var\ j)\ IMP\ A(i::=Eats(Var\ i)(Var\ j))))))$
- $IMP\ (All\ i\ A)$
- $\in induction_axioms$

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

$X4 \equiv \text{Abs_name } (\text{Atom } (\text{Sort } \text{"SyntaxN.name"} \ [])) 4$

1.2.3 The HF axioms

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

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

definition $HF_axioms \text{ where } HF_axioms = \{HF1, HF2\}$

1.2.4 Equality axioms

definition $refl_ax :: \text{fm where}$
 $refl_ax = \text{Var } X1 \text{ EQ } \text{Var } X1$

definition $eq_cong_ax :: \text{fm where}$
 $eq_cong_ax = ((\text{Var } X1 \text{ EQ } \text{Var } X2) \text{ AND } (\text{Var } X3 \text{ EQ } \text{Var } X4)) \text{ IMP}$
 $((\text{Var } X1 \text{ EQ } \text{Var } X3) \text{ IMP } (\text{Var } X2 \text{ EQ } \text{Var } X4))$

definition $mem_cong_ax :: \text{fm where}$
 $mem_cong_ax = ((\text{Var } X1 \text{ EQ } \text{Var } X2) \text{ AND } (\text{Var } X3 \text{ EQ } \text{Var } X4)) \text{ IMP}$
 $((\text{Var } X1 \text{ IN } \text{Var } X3) \text{ IMP } (\text{Var } X2 \text{ IN } \text{Var } X4))$

definition $eats_cong_ax :: \text{fm where}$
 $eats_cong_ax = ((\text{Var } X1 \text{ EQ } \text{Var } X2) \text{ AND } (\text{Var } X3 \text{ EQ } \text{Var } X4)) \text{ IMP}$
 $((\text{Eats } (\text{Var } X1) (\text{Var } X3)) \text{ EQ } (\text{Eats } (\text{Var } X2) (\text{Var } X4)))$

definition $equality_axioms :: \text{fm set where}$
 $equality_axioms = \{refl_ax, eq_cong_ax, mem_cong_ax, eats_cong_ax\}$

1.2.5 The proof system

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

consts $extra_axiom :: \text{fm}$

inductive $hfthm :: \text{fm set} \Rightarrow \text{fm} \Rightarrow \text{bool}$ (**infixl** \vdash 55)

where

- $Hyp: A \in H \Longrightarrow H \vdash A$
- $| Extra: H \vdash extra_axiom$
- $| Bool: A \in \text{boolean_axioms} \Longrightarrow H \vdash A$
- $| Eq: A \in \text{equality_axioms} \Longrightarrow H \vdash A$
- $| Spec: A \in \text{special_axioms} \Longrightarrow H \vdash A$
- $| HF: A \in HF_axioms \Longrightarrow H \vdash A$
- $| Ind: A \in \text{induction_axioms} \Longrightarrow H \vdash A$
- $| MP: H \vdash A \text{ IMP } B \Longrightarrow H' \vdash A \Longrightarrow H \cup H' \vdash B$
- $| Exists: H \vdash A \text{ IMP } B \Longrightarrow \text{atom } i \nmid B \Longrightarrow \forall C \in H. \text{atom } i \nmid C \Longrightarrow H \vdash (\text{Ex } i \ A) \text{ IMP } B$

1.2.6 Derived rules of inference

lemma $contraction: \text{insert } A (\text{insert } A \ H) \vdash B \Longrightarrow \text{insert } A \ H \vdash B$

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: $H \vdash B \implies \text{insert } A \ H \vdash B$
by (metis subset_insertI thin)

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

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

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

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

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

lemma rotate4:
 $\text{insert } A4 \ (\text{insert } A1 \ (\text{insert } A2 \ (\text{insert } A3 \ H))) \vdash B$
 $\implies \text{insert } A1 \ (\text{insert } A2 \ (\text{insert } A3 \ (\text{insert } A4 \ H))) \vdash B$
by (blast intro: thin)

lemma rotate5:
 $\text{insert } A5 \ (\text{insert } A1 \ (\text{insert } A2 \ (\text{insert } A3 \ (\text{insert } A4 \ H)))) \vdash B$
 $\implies \text{insert } A1 \ (\text{insert } A2 \ (\text{insert } A3 \ (\text{insert } A4 \ (\text{insert } A5 \ H)))) \vdash B$
by (blast intro: thin)

lemma rotate6:
 $\text{insert } A6 \ (\text{insert } A1 \ (\text{insert } A2 \ (\text{insert } A3 \ (\text{insert } A4 \ (\text{insert } A5 \ H)))) \vdash B$
 $\implies \text{insert } A1 \ (\text{insert } A2 \ (\text{insert } A3 \ (\text{insert } A4 \ (\text{insert } A5 \ (\text{insert } A6 \ H)))) \vdash B$
by (blast intro: thin)

lemma rotate7:
 $\text{insert } A7 \ (\text{insert } A1 \ (\text{insert } A2 \ (\text{insert } A3 \ (\text{insert } A4 \ (\text{insert } A5 \ (\text{insert } A6 \ H)))) \vdash B$
 $\implies \text{insert } A1 \ (\text{insert } A2 \ (\text{insert } A3 \ (\text{insert } A4 \ (\text{insert } A5 \ (\text{insert } A6 \ (\text{insert } A7 \ H)))) \vdash B$
by (blast intro: thin)

lemma rotate8:
 $\text{insert } A8 \ (\text{insert } A1 \ (\text{insert } A2 \ (\text{insert } A3 \ (\text{insert } A4 \ (\text{insert } A5 \ (\text{insert } A6 \ (\text{insert } A7 \ H)))) \vdash B$
 $\implies \text{insert } A1 \ (\text{insert } A2 \ (\text{insert } A3 \ (\text{insert } A4 \ (\text{insert } A5 \ (\text{insert } A6 \ (\text{insert } A7 \ (\text{insert } A8 \ H)))) \vdash B$
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)
    next
      case False
      hence H - {C} = H by auto
      thus ?thesis using Exists
        by (metis Imp_triv_I hfthm.Exists)
    qed
  qed

```

theorem *Imp_I* [intro!]: $insert\ A\ H \vdash B \implies H \vdash A\ IMP\ B$
 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*: $H \vdash A \implies insert\ A\ H \vdash B \implies H \vdash B$
 by (metis Un_absorb cut)

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

lemma *cut0*: $\{\} \vdash A \implies insert\ A\ H \vdash B \implies H \vdash B$
 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*: $\{A\} \vdash B \implies insert\ B\ H \vdash C \implies insert\ A\ H \vdash C$
 by (metis Assume cut1 cut_same rotate2 thin1)

lemma *cut2*: $\llbracket \{A,B\} \vdash C; H \vdash A; H \vdash B \rrbracket \Longrightarrow H \vdash C$
by (*metis Un_empty_right Un_insert_right cut cut_same*)

lemma *rcut2*: $\{A,B\} \vdash C \Longrightarrow \text{insert } C \ H \vdash D \Longrightarrow H \vdash B \Longrightarrow \text{insert } A \ H \vdash D$
by (*metis Assume cut2 cut_same insert_commute thin1*)

lemma *cut3*: $\llbracket \{A,B,C\} \vdash D; H \vdash A; H \vdash B; H \vdash C \rrbracket \Longrightarrow H \vdash D$
by (*metis MP_same cut2 Imp_I*)

lemma *cut4*: $\llbracket \{A,B,C,D\} \vdash E; H \vdash A; H \vdash B; H \vdash C; H \vdash D \rrbracket \Longrightarrow H \vdash E$
by (*metis MP_same cut3 [of B C D] Imp_I*)

1.3 Miscellaneous logical rules

lemma *Disj_I1*: $H \vdash A \Longrightarrow H \vdash A \text{ OR } B$
by (*metis Bool MP_same boolean_axioms.DisjI1*)

lemma *Disj_I2*: $H \vdash B \Longrightarrow H \vdash A \text{ OR } B$
by (*metis Disj_commute Disj_I1*)

lemma *Peirce*: $H \vdash (\text{Neg } A) \text{ IMP } A \Longrightarrow H \vdash A$
using *DisjConj* [*of Neg A A A*] *DisjCont* [*of A*]
by (*metis Bool MP_same boolean_axioms.Ident*)

lemma *Contra*: $\text{insert } (\text{Neg } A) \ H \vdash A \Longrightarrow H \vdash A$
by (*metis Peirce Imp_I*)

lemma *Imp_Neg_I*: $H \vdash A \text{ IMP } B \Longrightarrow H \vdash A \text{ IMP } (\text{Neg } B) \Longrightarrow H \vdash \text{Neg } A$
by (*metis DisjConj [of B Neg A Neg A] DisjCont Bool Disj_commute MP_same*)

lemma *NegNeg_I*: $H \vdash A \Longrightarrow H \vdash \text{Neg } (\text{Neg } A)$
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*: $H \vdash B \text{ AND } A \implies H \vdash A \text{ AND } B$
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** $(\bigwedge B. \text{atom } i \ \# \ B \implies \text{insert } A \ H \vdash B)$ **shows** $H \vdash \text{Neg } A$
by (*rule Imp_Neg_I [where B = Zero IN Zero]*) (*auto simp: assms*)

lemma *Neg_mono*: $\text{insert } A \ H \vdash B \implies \text{insert } (\text{Neg } B) \ H \vdash \text{Neg } A$
by (*rule Neg_I0*) (*metis Hyp Neg_D insert_commute insertI1 thin1*)

lemma *Conj_mono*: $\text{insert } A \ H \vdash B \implies \text{insert } C \ H \vdash D \implies \text{insert } (A \ \text{AND } C) \ H \vdash B \ \text{AND } D$
by (*metis Conj_E1 Conj_E2 Conj_I Hyp Un_absorb2 cut insertI1 subset_insertI*)

lemma *Disj_mono*:
assumes $\text{insert } A \ H \vdash B \ \text{insert } C \ H \vdash D$ **shows** $\text{insert } (A \ \text{OR } C) \ H \vdash B \ \text{OR } D$
proof –
{ **fix** $A \ B \ C \ H$
have $\text{insert } (A \ \text{OR } C) \ H \vdash (A \ \text{IMP } B) \ \text{IMP } C \ \text{OR } B$
by (*metis Bool Hyp MP_same boolean_axioms.DisjConj insertI1*)
hence $\text{insert } A \ H \vdash B \implies \text{insert } (A \ \text{OR } C) \ H \vdash C \ \text{OR } B$
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 $A: \text{insert } A \ H \vdash C$ **and** $B: \text{insert } B \ H \vdash C$ **shows** $\text{insert } (A \ \text{OR } B) \ H \vdash C$
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'*: $\text{insert } A \ H \vdash \text{Neg } A \implies H \vdash \text{Neg } A$
by (*metis Contra Neg_mono*)

lemma *NegNeg_E* [intro!]: $\text{insert } A \ H \vdash B \implies \text{insert } (\text{Neg } (\text{Neg } A)) \ H \vdash B$
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:  $H \vdash A$  and B:  $\text{insert } B \ H \vdash C$  shows  $\text{insert } (A \text{ IMP } B) \ H \vdash C$ 
proof -
  have  $\text{insert } (A \text{ IMP } B) \ H \vdash B$ 
  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  $\text{insert } C \ H \vdash A \text{ IMP } B \ \{A\} \vdash C$ 
  shows  $H \vdash A \text{ IMP } B$ 
  by (metis Contra Disj_I1 Neg_mono assms rcut1)

lemma Iff_I [intro!]:  $\text{insert } A \ H \vdash B \implies \text{insert } B \ H \vdash A \implies H \vdash A \text{ IFF } B$ 
  by (metis Iff_def Conj_I Imp_I)

lemma Iff_MP_same:  $H \vdash A \text{ IFF } B \implies H \vdash A \implies H \vdash B$ 
  by (metis Iff_def Conj_E1 MP_same)

lemma Iff_MP2_same:  $H \vdash A \text{ IFF } B \implies H \vdash B \implies H \vdash A$ 
  by (metis Iff_def Conj_E2 MP_same)

lemma Iff_refl [intro!]:  $H \vdash A \text{ IFF } A$ 
  by (metis Hyp Iff_I insertI1)

lemma Iff_sym:  $H \vdash A \text{ IFF } B \implies H \vdash B \text{ IFF } A$ 
  by (metis Iff_def Conj_commute)

lemma Iff_trans:  $H \vdash A \text{ IFF } B \implies H \vdash B \text{ IFF } C \implies H \vdash A \text{ IFF } C$ 
  unfolding Iff_def
  by (metis Conj_E1 Conj_E2 Conj_I Disj_Semicong_1 Disj_commute)

lemma Iff_E:
   $\text{insert } A \ (\text{insert } B \ H) \vdash C \implies \text{insert } (\text{Neg } A) \ (\text{insert } (\text{Neg } B) \ H) \vdash C \implies \text{insert } (A \text{ IFF } B) \ H \vdash C$ 
  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:  $\text{insert } B \ H \vdash C$  shows  $\text{insert } (A \text{ IFF } B) \ H \vdash C$ 
  by (metis Iff_def A B Conj_E Imp_E insert_commute thin1)

lemma Iff_E2:
  assumes A:  $H \vdash A$  and B:  $\text{insert } B \ H \vdash C$  shows  $\text{insert } (B \text{ IFF } A) \ H \vdash C$ 
  by (metis Iff_def A B Bool Conj_E2 Conj_mono Imp_E boolean_axioms.Ident)

lemma Iff_MP_left:  $H \vdash A \text{ IFF } B \implies \text{insert } A \ H \vdash C \implies \text{insert } B \ H \vdash C$ 
  by (metis Hyp Iff_E2 cut_same insertI1 insert_commute thin1)

lemma Iff_MP_left':  $H \vdash A \text{ IFF } B \implies \text{insert } B \ H \vdash C \implies \text{insert } A \ H \vdash C$ 
  by (metis Iff_MP_left Iff_sym)

lemma Swap:  $\text{insert } (\text{Neg } B) \ H \vdash A \implies \text{insert } (\text{Neg } A) \ H \vdash B$ 
  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 $Conj_cong: H \vdash A\ IFF\ A' \implies H \vdash B\ IFF\ B' \implies H \vdash A\ AND\ B\ IFF\ A'\ AND\ B'$
by (*metis Conj_def Disj_cong Neg_cong*)

lemma $Imp_cong: H \vdash A\ IFF\ A' \implies H \vdash B\ IFF\ B' \implies H \vdash (A\ IMP\ B)\ IFF\ (A'\ IMP\ B')$
by (*metis Disj_cong Neg_cong*)

lemma $Iff_cong: H \vdash A\ IFF\ A' \implies H \vdash B\ IFF\ B' \implies H \vdash (A\ IFF\ B)\ IFF\ (A'\ IFF\ B')$
by (*metis Iff_def Conj_cong Imp_cong*)

lemma $Ex_cong: H \vdash A\ IFF\ A' \implies \forall C \in H. atom\ i \ \# \ C \implies H \vdash (Ex\ i\ A)\ IFF\ (Ex\ i\ A')$
apply (*rule Iff_I*)
apply (*metis Ex_mono Hyp Iff_MP_same Un_absorb Un_insert_right insertI1 thin_Un*)
apply (*metis Ex_mono Hyp Iff_MP2_same Un_absorb Un_insert_right insertI1 thin_Un*)
done

lemma $All_cong: H \vdash A\ IFF\ A' \implies \forall C \in H. atom\ i \ \# \ C \implies H \vdash (All\ i\ A)\ IFF\ (All\ i\ A')$
by (*metis Ex_cong Neg_cong*)

lemma $Subst: H \vdash A \implies \forall B \in H. atom\ i \ \# \ B \implies H \vdash A\ (i::=x)$
by (*metis All_D All_I*)

1.4 Equality reasoning

1.4.1 The congruence property for (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 $v2$: *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 $\{\} \vdash (\text{Var } X1 \text{ EQ } \text{Var } X2 \text{ AND } \text{Var } X3 \text{ EQ } \text{Var } X4) \text{ IMP } (\text{Var } X1 \text{ EQ } \text{Var } X3 \text{ IMP } \text{Var } X2 \text{ EQ } \text{Var } X4)$
by (*rule Eq*) (*simp add: eq_cong_ax_def equality_axioms_def*)
hence $\{\} \vdash (\text{Var } X1 \text{ EQ } \text{Var } X2 \text{ AND } \text{Var } X3 \text{ EQ } \text{Var } X4) \text{ IMP } (\text{Var } X1 \text{ EQ } \text{Var } X3 \text{ IMP } \text{Var } X2 \text{ EQ } \text{Var } X4)$
by (*drule_tac i=X1 and x=Var X1 in Subst*) *simp_all*
hence $\{\} \vdash (\text{Var } X1 \text{ EQ } \text{Var } v2 \text{ AND } \text{Var } X3 \text{ EQ } \text{Var } X4) \text{ IMP } (\text{Var } X1 \text{ EQ } \text{Var } X3 \text{ IMP } \text{Var } v2 \text{ EQ } \text{Var } X4)$
by (*drule_tac i=X2 and x=Var v2 in Subst*) *simp_all*
hence $\{\} \vdash (\text{Var } X1 \text{ EQ } \text{Var } v2 \text{ AND } \text{Var } v3 \text{ EQ } \text{Var } X4) \text{ IMP } (\text{Var } X1 \text{ EQ } \text{Var } v3 \text{ IMP } \text{Var } v2 \text{ EQ } \text{Var } X4)$
using $v2$
by (*drule_tac i=X3 and x=Var v3 in Subst*) *simp_all*
hence $\{\} \vdash (\text{Var } X1 \text{ EQ } \text{Var } v2 \text{ AND } \text{Var } v3 \text{ EQ } \text{Var } v4) \text{ IMP } (\text{Var } X1 \text{ EQ } \text{Var } v3 \text{ IMP } \text{Var } v2 \text{ EQ } \text{Var } v4)$
using $v2 \ v3$
by (*drule_tac i=X4 and x=Var v4 in Subst*) *simp_all*
hence $\{\} \vdash (t \text{ EQ } \text{Var } v2 \text{ AND } \text{Var } v3 \text{ EQ } \text{Var } v4) \text{ IMP } (t \text{ EQ } \text{Var } v3 \text{ IMP } \text{Var } v2 \text{ EQ } \text{Var } v4)$
using $v2 \ v3 \ v4$
by (*drule_tac i=X1 and x=t in Subst*) *simp_all*
hence $\{\} \vdash (t \text{ EQ } t' \text{ AND } \text{Var } v3 \text{ EQ } \text{Var } v4) \text{ IMP } (t \text{ EQ } \text{Var } v3 \text{ IMP } t' \text{ EQ } \text{Var } v4)$
using $v2 \ v3 \ v4$
by (*drule_tac i=v2 and x=t' in Subst*) *simp_all*
hence $\{\} \vdash (t \text{ EQ } t' \text{ AND } u \text{ EQ } \text{Var } v4) \text{ IMP } (t \text{ EQ } u \text{ IMP } t' \text{ EQ } \text{Var } v4)$
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 $\{\} \vdash \text{Var } X1 \text{ EQ } \text{Var } X1$
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** $H \vdash x \text{ EQ } y \text{ H } \vdash y \text{ EQ } z$ **shows** $H \vdash x \text{ EQ } z$
proof –
have $\wedge H. H \vdash (x \text{ EQ } x \text{ AND } y \text{ EQ } z) \text{ IMP } (x \text{ EQ } y \text{ IMP } x \text{ EQ } z)$
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 $H \vdash t \text{ EQ } t' \text{ H } \vdash u \text{ EQ } u'$ **shows** $H \vdash t \text{ EQ } u \text{ IFF } t' \text{ EQ } u'$
proof –
{ **fix** $t \ t' \ u \ u'$
assume $H \vdash t \text{ EQ } t' \text{ H } \vdash u \text{ EQ } u'$
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 (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 $v2: \text{atom } v2 \ \# \ (t, X1, X3, X4)$
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 $\{\} \vdash (\text{Var } X1 \text{ EQ } \text{Var } X2 \text{ AND } \text{Var } X3 \text{ EQ } \text{Var } X4) \text{ IMP } (\text{Var } X1 \text{ IN } \text{Var } X3 \text{ IMP } \text{Var } X2 \text{ IN } \text{Var } X4)$
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=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 $\{\} \vdash (\text{Var } X1 \text{ EQ } \text{Var } v2 \text{ AND } \text{Var } v3 \text{ EQ } \text{Var } v4) \text{ IMP } (\text{Var } X1 \text{ IN } \text{Var } v3 \text{ IMP } \text{Var } v2 \text{ IN } \text{Var } v4)$
using $v2 \ v3$
by $(\text{drule_tac } i=X4 \text{ and } x=\text{Var } v4 \text{ in } \text{Subst}) \text{ simp_all}$
hence $\{\} \vdash (t \text{ EQ } \text{Var } v2 \text{ AND } \text{Var } v3 \text{ EQ } \text{Var } v4) \text{ IMP } (t \text{ IN } \text{Var } v3 \text{ IMP } \text{Var } v2 \text{ IN } \text{Var } v4)$
using $v2 \ v3 \ v4$
by $(\text{drule_tac } i=X1 \text{ and } x=t \text{ in } \text{Subst}) \text{ simp_all}$
hence $\{\} \vdash (t \text{ EQ } t' \text{ AND } \text{Var } v3 \text{ EQ } \text{Var } v4) \text{ IMP } (t \text{ IN } \text{Var } v3 \text{ IMP } t' \text{ IN } \text{Var } v4)$
using $v2 \ v3 \ v4$
by $(\text{drule_tac } i=v2 \text{ and } x=t' \text{ in } \text{Subst}) \text{ simp_all}$
hence $\{\} \vdash (t \text{ EQ } t' \text{ AND } u \text{ EQ } \text{Var } v4) \text{ IMP } (t \text{ IN } u \text{ IMP } t' \text{ IN } \text{Var } v4)$
using $v3 \ v4$
by $(\text{drule_tac } i=v3 \text{ and } x=u \text{ in } \text{Subst}) \text{ simp_all}$
thus *?thesis*
using $v4$
by $(\text{drule_tac } i=v4 \text{ and } x=u' \text{ in } \text{Subst}) \text{ simp_all}$
qed

lemma *Mem_cong*:

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

proof –

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

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

by $(\text{metis } \text{AssumeH}(2) \ \text{Conj_I} \ \text{MP_null} \ \text{Mem_cong1} \ \text{insert_commute})$

}

thus *?thesis*

by $(\text{metis } \text{Iff_def} \ \text{Conj_I} \ \text{cut2} \ \text{assms} \ \text{Sym})$

qed

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

lemma *Eats_cong1*: $\{\} \vdash (t \text{ EQ } t' \text{ AND } u \text{ EQ } u') \text{ IMP } (\text{Eats } t \ u \text{ EQ } \text{Eats } t' \ u')$

proof –

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

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 $(\text{metis } \text{obtain_fresh})$

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

by $(\text{metis } \text{eats_cong_ax_def} \ \text{equality_axioms_def} \ \text{insert_iff} \ \text{Eq})$

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

by $(\text{drule_tac } i=X2 \text{ and } x=\text{Var } v2 \text{ in } \text{Subst}) \text{ simp_all}$

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

using $v2$

by $(\text{drule_tac } i=X3 \text{ and } x=\text{Var } v3 \text{ in } \text{Subst}) \text{ simp_all}$

hence $\{\} \vdash (\text{Var } X1 \text{ EQ } \text{Var } v2 \text{ AND } \text{Var } v3 \text{ EQ } \text{Var } v4) \text{ IMP } (\text{Eats } (\text{Var } X1) (\text{Var } v3) \text{ EQ } \text{Eats } (\text{Var } v2) (\text{Var } v4))$

using $v2 \ v3$

by $(\text{drule_tac } i=X4 \text{ and } x=\text{Var } v4 \text{ in } \text{Subst}) \text{ simp_all}$

hence $\{\} \vdash (t \text{ EQ } \text{Var } v2 \text{ AND } \text{Var } v3 \text{ EQ } \text{Var } v4) \text{ IMP } (\text{Eats } t \ (\text{Var } v3) \text{ EQ } \text{Eats } (\text{Var } v2) \ (\text{Var } v4))$

using $v2 \ v3 \ v4$

by $(\text{drule_tac } i=X1 \text{ and } x=t \text{ in } \text{Subst}) \text{ simp_all}$

hence $\{\} \vdash (t \text{ EQ } t' \text{ AND } \text{Var } v3 \text{ EQ } \text{Var } v4) \text{ IMP } (\text{Eats } t \ (\text{Var } v3) \text{ EQ } \text{Eats } t' \ (\text{Var } v4))$

using $v2 \ v3 \ v4$

by $(\text{drule_tac } i=v2 \text{ and } x=t' \text{ in } \text{Subst}) \text{ simp_all}$

hence $\{\} \vdash (t \text{ EQ } t' \text{ AND } u \text{ EQ } \text{Var } v_4) \text{ IMP } (Eats \ t \ u \ \text{EQ } Eats \ t' \ (\text{Var } v_4))$
using $v_3 \ v_4$
by $(drule_tac \ i=v_3 \ \text{and } x=u \ \text{in } \text{Subst}) \ \text{simp_all}$
thus $?thesis$
using v_4
by $(drule_tac \ i=v_4 \ \text{and } x=u' \ \text{in } \text{Subst}) \ \text{simp_all}$
qed

lemma $Eats_cong: \llbracket H \vdash t \ \text{EQ } t'; H \vdash u \ \text{EQ } u' \rrbracket \implies H \vdash Eats \ t \ u \ \text{EQ } Eats \ t' \ u'$
by $(metis \ \text{Conj_I} \ \text{anti_deduction} \ Eats_cong1 \ \text{cut1})$

lemma $HPair_cong: \llbracket H \vdash t \ \text{EQ } t'; H \vdash u \ \text{EQ } u' \rrbracket \implies H \vdash \text{HPair } \ t \ u \ \text{EQ } \text{HPair } \ t' \ u'$
by $(metis \ \text{HPair_def} \ Eats_cong \ \text{Refl})$

lemma $SUCC_cong: H \vdash t \ \text{EQ } t' \implies H \vdash \text{SUCC } \ t \ \text{EQ } \text{SUCC } \ t'$
by $(metis \ Eats_cong \ \text{SUCC_def})$

1.4.4 Substitution for Equalities

lemma $Eq_subst_tm_Iff: \{t \ \text{EQ } u\} \vdash \text{subst } \ i \ t \ \text{tm} \ \text{EQ } \text{subst } \ i \ u \ \text{tm}$
by $(\text{induct } \ \text{tm} \ \text{rule: } \ \text{tm.induct}) \ (\text{auto } \ \text{simp: } \ Eats_cong)$

lemma $Eq_subst_fm_Iff: \text{insert } (t \ \text{EQ } u) \ H \vdash A(i::=t) \ \text{IFF } A(i::=u)$

proof –

have $\{t \ \text{EQ } u\} \vdash A(i::=t) \ \text{IFF } A(i::=u)$
by $(\text{nominal_induct } \ A \ \text{avoiding: } \ i \ t \ u \ \text{rule: } \ \text{fm.strong_induct})$
 $(\text{auto } \ \text{simp: } \ \text{Disj_cong} \ \text{Neg_cong} \ \text{Ex_cong} \ \text{Mem_cong} \ \text{Eq_cong} \ \text{Eq_subst_tm_Iff})$
thus $?thesis$
by $(metis \ \text{Assume } \ \text{cut1})$

qed

lemma $\text{Var_Eq_subst_Iff: } \text{insert } (\text{Var } \ i \ \text{EQ } \ t) \ H \vdash A(i::=t) \ \text{IFF } A$
by $(metis \ \text{Eq_subst_fm_Iff} \ \text{Iff_sym} \ \text{subst_fm_id})$

lemma $\text{Var_Eq_imp_subst_Iff: } H \vdash \text{Var } \ i \ \text{EQ } \ t \implies H \vdash A(i::=t) \ \text{IFF } A$
by $(metis \ \text{Var_Eq_subst_Iff} \ \text{cut_same})$

1.4.5 Congruence Rules for Predicates

lemma $P1_cong:$

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

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

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

proof –

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

by $(metis \ \text{obtain_fresh})$

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

by $(\text{rule } \ \text{Eq_subst_fm_Iff})$

thus $?thesis$ **using** $\text{assms } \ i$

by $(metis \ \text{cut_same} \ \text{subst.simps}(2))$

qed

lemma $P2_cong:$

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

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

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 EQ x' } \vdash P x y IFF P x' y
 by (rule P1_cong [where tms=[y,x']@tms]) (auto simp: fresh_Cons sub)
 hence {x EQ x', y EQ y'} \vdash P x y IFF P x' y'
 by (metis Assume Iff_trans cut1 rotate2 yy')
 thus ?thesis
 by (metis cut2 eq)
 qed

lemma P3_cong:

fixes tms :: tm list
 assumes sub: $\bigwedge i t u v x. \text{atom } i \# \text{tms} \implies$
 $(P t u v)(i::=x) = P (\text{subst } i x t) (\text{subst } i x u) (\text{subst } i x v)$
 and eq: $H \vdash x \text{EQ } x' \ H \vdash y \text{EQ } y' \ H \vdash z \text{EQ } z'$
 shows $H \vdash P x y z \text{ IFF } P x' y' z'$

proof –

obtain i::name where i: atom i # (z,z',y,y',x,x')
 by (metis obtain_fresh)
 have tl: { y EQ y', z EQ z' } \vdash P x' y z IFF P x' y' z'
 by (rule P2_cong [where tms=[z,z',y,y',x,x']@tms]) (auto simp: fresh_Cons sub)
 have hd: { x EQ x' } \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: $\bigwedge i t1 t2 t3 t4 x. \text{atom } i \# \text{tms} \implies$
 $(P t1 t2 t3 t4)(i::=x) = P (\text{subst } i x t1) (\text{subst } i x t2) (\text{subst } i x t3) (\text{subst } i x t4)$
 and eq: $H \vdash x1 \text{EQ } x1' \ H \vdash x2 \text{EQ } x2' \ H \vdash x3 \text{EQ } x3' \ H \vdash x4 \text{EQ } x4'$
 shows $H \vdash P x1 x2 x3 x4 \text{ IFF } P x1' x2' x3' x4'$

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 $H \vdash (t \text{EQ } \text{Zero}) \text{ IFF } (\text{All } i (\text{Neg } ((\text{Var } i) \text{IN } t)))$

proof –

obtain i'::name where i': atom i' # (t, X0, X1, i)
 by (rule obtain_fresh)
 have {} $\vdash ((\text{Var } X0) \text{EQ } \text{Zero}) \text{ IFF } (\text{All } X1 (\text{Neg } ((\text{Var } X1) \text{IN } (\text{Var } X0))))$
 by (simp add: HF HF_axioms_def HF1_def)
 then have {} $\vdash (((\text{Var } X0) \text{EQ } \text{Zero}) \text{ IFF } (\text{All } X1 (\text{Neg } ((\text{Var } X1) \text{IN } (\text{Var } X0)))))(X0 ::= t)$
 by (rule Subst) simp

```

hence {} ⊢ (t EQ Zero) IFF (All i' (Neg ((Var i') IN t))) using i'
  by simp
also have ... = (FRESH i'. (t EQ Zero) IFF (All i' (Neg ((Var i') IN t))))
  using i' by simp
also have ... = (t EQ Zero) IFF (All i (Neg ((Var i) IN t)))
  using assms by simp
finally show ?thesis
  by (metis empty_subsetI thin)
qed

```

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

proof –

```

obtain i::name where atom i ≠ Zero
  by (rule obtain_fresh)
hence {} ⊢ All i (Neg ((Var i) IN Zero))
  by (metis Mem_Zero_iff Iff_MP_same Refl)
hence {} ⊢ 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 ≡ Zero IN Zero

lemma Fls_eqvt [eqvt]: (p · 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 ⊢ Fls ⇒ H ⊢ Neg A
unfolding Fls_def
by (rule Neg_I0) (metis Mem_Zero_E cut_same)

lemma Neg_E [intro!]: H ⊢ A ⇒ insert (Neg A) H ⊢ 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 ⊢ A ⇒ insert B H ⊢ Fls ⇒ H ⊢ Neg (A IMP B)
by (metis NegNeg_I Neg_Disj_I Neg_I)

```

lemma Neg_Imp_E [intro!]: insert (Neg B) (insert A H) ⊢ C ⇒ insert (Neg (A IMP B)) H ⊢ C
apply (rule cut_same [where A=A])
apply (metis Assume Disj_I1 NegNeg_D Neg_mono)
apply (metis Swap Imp_I rotate2 thin1)
done

```

```

declare Neg_Imp_E [THEN rotate2, intro!]
declare Neg_Imp_E [THEN rotate3, intro!]
declare Neg_Imp_E [THEN rotate4, intro!]
declare Neg_Imp_E [THEN rotate5, intro!]
declare Neg_Imp_E [THEN rotate6, intro!]
declare Neg_Imp_E [THEN rotate7, intro!]
declare Neg_Imp_E [THEN rotate8, intro!]

```

```

lemma Fls_E [intro!]: insert Fls H ⊢ 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 ⊢ (Neg Fls)
by (metis Fls_E Neg_I)

```

```

lemma ExFalso: H ⊢ Fls ⇒ H ⊢ A
by (metis Neg_D truth_provable)

```

1.5.2 More properties of Zero

```

lemma Eq_Zero_D:
assumes H ⊢ t EQ Zero H ⊢ u IN t shows H ⊢ A
proof –
obtain i::name where i: atom i # t
by (rule obtain_fresh)
with assms have an: H ⊢ (All i (Neg ((Var i) IN t)))
by (metis Iff_MP_same Mem_Zero_iff)
have H ⊢ Neg (u IN t) using All_D [OF an, of u] i
by simp
thus ?thesis using assms
by (metis Neg_D)
qed

```

```

lemma Eq_Zero_thm:
assumes atom i # t shows {All i (Neg ((Var i) IN t))} ⊢ 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 ⊢ Fls and i1: atom i # t and i2: ∀ B ∈ H. atom i # B
shows H ⊢ t EQ Zero
proof –
have H ⊢ 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 $H \vdash (z \text{ EQ } \text{Eats } t \ u) \text{ IFF } (\text{All } i \ (\text{Var } i \ \text{IN } z \ \text{IFF } \text{Var } i \ \text{IN } t \ \text{OR } \text{Var } i \ \text{EQ } u))$

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 $\{\} \vdash ((\text{Var } X0) \text{ EQ } (\text{Eats } (\text{Var } X1) (\text{Var } X2))) \text{ IFF}$

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

by (simp add: HF HF_axioms_def HF2_def)

hence $\{\} \vdash ((\text{Var } X0) \text{ EQ } (\text{Eats } (\text{Var } X1) (\text{Var } X2))) \text{ IFF}$

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

by (drule_tac *i=X0* **and** *x=Var X0* **in** *Subst*) simp_all

hence $\{\} \vdash ((\text{Var } X0) \text{ EQ } (\text{Eats } (\text{Var } v1) (\text{Var } X2))) \text{ IFF}$

$(\text{All } X3 \ (\text{Var } X3 \ \text{IN } \text{Var } X0 \ \text{IFF } \text{Var } X3 \ \text{IN } \text{Var } v1 \ \text{OR } \text{Var } X3 \ \text{EQ } \text{Var } X2))$

using *v1* **by** (drule_tac *i=X1* **and** *x=Var v1* **in** *Subst*) simp_all

hence $\{\} \vdash ((\text{Var } X0) \text{ EQ } (\text{Eats } (\text{Var } v1) (\text{Var } v2))) \text{ IFF}$

$(\text{All } X3 \ (\text{Var } X3 \ \text{IN } \text{Var } X0 \ \text{IFF } \text{Var } X3 \ \text{IN } \text{Var } v1 \ \text{OR } \text{Var } X3 \ \text{EQ } \text{Var } v2))$

using *v1 v2* **by** (drule_tac *i=X2* **and** *x=Var v2* **in** *Subst*) simp_all

hence $\{\} \vdash (((\text{Var } X0) \text{ EQ } (\text{Eats } (\text{Var } v1) (\text{Var } v2)))) \text{ IFF}$

$(\text{All } X3 \ (\text{Var } X3 \ \text{IN } \text{Var } X0 \ \text{IFF } \text{Var } X3 \ \text{IN } \text{Var } v1 \ \text{OR } \text{Var } X3 \ \text{EQ } \text{Var } v2))(X0 ::= z)$

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 ... = (*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 ... = $(z \text{ EQ } \text{Eats } t \ u) \text{ IFF } (\text{All } i \ (\text{Var } i \ \text{IN } z \ \text{IFF } \text{Var } i \ \text{IN } t \ \text{OR } \text{Var } i \ \text{EQ } u))$

using *assms i'* **by** simp

finally show *?thesis*

by (rule *thin0*)

qed

lemma *Eq_Eats_I*:

$H \vdash \text{All } i \ (\text{Var } i \ \text{IN } z \ \text{IFF } \text{Var } i \ \text{IN } t \ \text{OR } \text{Var } i \ \text{EQ } u) \implies \text{atom } i \ \# \ (z,t,u) \implies H \vdash z \text{ EQ } \text{Eats } t \ u$

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::name* **where** *atom i* $\#$ (*Eats t u, t, u*)

by (rule obtain_fresh)

thus *?thesis*

using *Iff_MP_same [OF Eq_Eats_iff, THEN All_D]*

by *auto*

qed

lemma *Mem_Eats_I1*: $H \vdash u \text{ IN } t \implies H \vdash u \text{ IN Eats } t \ z$
by (metis *Disj_I1 Iff_MP2_same Mem_Eats_Iff*)

lemma *Mem_Eats_I2*: $H \vdash u \text{ EQ } z \implies H \vdash u \text{ IN Eats } t \ z$
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 $\text{insert } (u \text{ IN Eats } t \ z) \ H \vdash C$
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 [THEN rotate3]*
Mem_Eats_E [THEN rotate4] Mem_Eats_E [THEN rotate5]
Mem_Eats_E [THEN rotate6] Mem_Eats_E [THEN rotate7] Mem_Eats_E [THEN
rotate8]
declare *Mem_Eats_EH* [intro!]

lemma *Mem_SUCC_I1*: $H \vdash u \text{ IN } t \implies H \vdash u \text{ IN SUCC } t$
by (metis *Mem_Eats_I1 SUCC_def*)

lemma *Mem_SUCC_I2*: $H \vdash u \text{ EQ } t \implies H \vdash u \text{ IN SUCC } t$
by (metis *Mem_Eats_I2 SUCC_def*)

lemma *Mem_SUCC_Refl* [simp]: $H \vdash k \text{ IN SUCC } k$
by (metis *Mem_SUCC_I2 Refl*)

lemma *Mem_SUCC_E*:
assumes $\text{insert } (u \text{ IN } t) \ H \vdash C$ and $\text{insert } (u \text{ EQ } t) \ H \vdash C$ shows $\text{insert } (u \text{ IN SUCC } t) \ H \vdash C$
by (metis *assms Mem_Eats_E SUCC_def*)

lemmas *Mem_SUCC_EH* = *Mem_SUCC_E Mem_SUCC_E [THEN rotate2] Mem_SUCC_E [THEN*
rotate3] Mem_SUCC_E [THEN rotate4] Mem_SUCC_E [THEN rotate5]
Mem_SUCC_E [THEN rotate6] Mem_SUCC_E [THEN rotate7] Mem_SUCC_E [THEN
rotate8]

lemma *Eats_EQ_Zero_E*: $\text{insert } (Eats \ t \ u \ \text{EQ} \ \text{Zero}) \ H \vdash A$
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*: $\text{insert } (\text{Zero} \ \text{EQ} \ Eats \ t \ u) \ H \vdash A$
by (metis *Eats_EQ_Zero_E Sym_L*)

lemmas *Eats_EQ_Zero_E2H* = *Eats_EQ_Zero_E2 Eats_EQ_Zero_E2 [THEN rotate2] Eats_EQ_Zero_E2*
[THEN rotate3] Eats_EQ_Zero_E2 [THEN rotate4] Eats_EQ_Zero_E2 [THEN rotate5]
Eats_EQ_Zero_E2 [THEN rotate6] Eats_EQ_Zero_E2 [THEN rotate7] Eats_EQ_Zero_E2
[THEN rotate8]
declare *Eats_EQ_Zero_E2H* [intro!]

1.6 Bounded Quantification involving *Eats*

lemma *All2_cong*: $H \vdash t \text{ EQ } t' \implies H \vdash A \text{ IFF } A' \implies \forall C \in H. \text{atom } i \# C \implies H \vdash (\text{All2 } i \ t \ A) \text{ IFF } (\text{All2 } i \ t' \ A')$

by (*metis All_cong Imp_cong Mem_cong Refl*)

lemma *All2_Zero_E* [*intro!*]: $H \vdash B \implies \text{insert } (\text{All2 } i \ \text{Zero } A) \ H \vdash B$

by (*rule thin1*)

lemma *All2_Eats_I_D*:

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

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*:

$\llbracket \text{atom } i \# (t, u); H \vdash \text{All2 } i \ t \ A; H \vdash A(i::=u) \rrbracket \implies H \vdash (\text{All2 } i \ (\text{Eats } t \ u) \ A)$

by (*rule cut2 [OF All2_Eats_I_D], auto*)

lemma *All2_Eats_E1*:

$\llbracket \text{atom } i \# (t, u); \forall C \in H. \text{atom } i \# C \rrbracket \implies \text{insert } (\text{All2 } i \ (\text{Eats } t \ u) \ A) \ H \vdash \text{All2 } i \ t \ A$

by *auto* (*metis Assume Ex_I Imp_E Mem_Eats_I1 Neg_mono subst_fm_id*)

lemma *All2_Eats_E2*:

$\llbracket \text{atom } i \# (t, u); \forall C \in H. \text{atom } i \# C \rrbracket \implies \text{insert } (\text{All2 } i \ (\text{Eats } t \ u) \ A) \ H \vdash A(i::=u)$

by (*rule All_E [where x=u]*) (*auto intro: ContraProve Mem_Eats_I2*)

lemma *All2_Eats_E*:

assumes *i*: $\text{atom } i \# (t, u)$

and *B*: $\text{insert } (\text{All2 } i \ t \ A) \ (\text{insert } (A(i::=u)) \ H) \vdash B$

shows $\text{insert } (\text{All2 } i \ (\text{Eats } t \ u) \ A) \ H \vdash B$

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*:

$\text{atom } i \# t \implies H \vdash \text{All2 } i \ t \ A \implies H \vdash A(i::=t) \implies H \vdash (\text{All2 } i \ (\text{SUCC } t) \ A)$

by (*simp add: SUCC_def All2_Eats_I*)

lemma *All2_SUCC_E*:

assumes $\text{atom } i \# t$

and $\text{insert } (\text{All2 } i \ t \ A) \ (\text{insert } (A(i::=t)) \ H) \vdash B$

shows $\text{insert } (\text{All2 } i \ (\text{SUCC } t) \ A) \ H \vdash B$

by (*simp add: SUCC_def All2_Eats_E assms*)

lemma *All2_SUCC_E'*:

assumes $H \vdash u \text{ EQ } \text{SUCC } t$

and $\text{atom } i \# t \ \forall C \in H. \text{atom } i \# C$

and $\text{insert } (\text{All2 } i \ t \ A) \ (\text{insert } (A(i::=t)) \ H) \vdash B$

shows $\text{insert } (\text{All2 } i \ u \ A) \ H \vdash B$

by (*metis All2_SUCC_E Iff_MP_left' Iff_refl All2_cong assms*)

1.7 Induction

lemma *Ind*:

assumes *j*: $\text{atom } (j::\text{name}) \# (i, A)$

and *prems*: $H \vdash A(i::=\text{Zero}) \ H \vdash \text{All } i \ (\text{All } j \ (A \ \text{IMP} \ (A(i::=\text{Var } j) \ \text{IMP} \ A(i::=\text{Eats}(\text{Var } i)(\text{Var } j))))))$

shows $H \vdash A$
proof –
have $\{A(i::=Zero), \text{All } i (\text{All } j (A \text{ IMP } (A(i::= Var j) \text{ IMP } A(i::= Eats(Var i)(Var j))))))\} \vdash \text{All } i A$
by (*metis j hfthm.Ind ind anti_deduction insert_commute*)
hence $H \vdash (\text{All } i A)$
by (*metis cut2 prems*)
thus *?thesis*
by (*metis All_E' Assume subst_fm_id*)
qed
end

Chapter 2

De Bruijn Syntax, Quotations, Codes, V-Codes

```
theory Coding
imports SyntaxN
begin
```

```
declare fresh_Nil [iff]
```

2.1 de Bruijn Indices (locally-nameless version)

```
nominal_datatype dbtm = DBZero | DBVar name | DBInd nat | DBEats dbtm dbtm
```

```
nominal_datatype dbfm =
  DBMem dbtm dbtm
| DBEq dbtm dbtm
| DBDisj dbfm dbfm
| DBNeg dbfm
| DBEx dbfm
```

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

```
fun lookup :: name list  $\Rightarrow$  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 # e  $\Longrightarrow$  name  $\notin$  set e
  by (metis List.finite_set fresh_finite_set_at_base fresh_set)
```

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

```
lemma lookup_in:
  x  $\in$  set e  $\Longrightarrow$   $\exists k$ . lookup e n x = DBInd k  $\wedge$  n  $\leq$  k  $\wedge$  k  $<$  n + length e
apply (induct e arbitrary: n)
apply (auto intro: Suc_leD)
apply (metis Suc_leD add_Suc_right add_Suc_shift)
done
```

```
lemma lookup_fresh: x # lookup e n y  $\longleftrightarrow$  y  $\in$  set e  $\vee$  x  $\neq$  atom y
```

by (induct arbitrary: n rule: lookup.induct) (auto simp: pure_fresh fresh_at_base)

lemma *lookup_eqvt*[*eqvt*]: $(p \cdot \text{lookup } xs \ n \ x) = \text{lookup } (p \cdot xs) \ (p \cdot n) \ (p \cdot x)$
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*]: $i \# \text{trans_tm } e \ t \longleftrightarrow i \# t \vee i \in \text{atom } ' \text{set } e$
by (induct t rule: tm.induct, auto simp: lookup_fresh fresh_at_base)

lemma *trans_tm_forget*: $\text{atom } i \# t \Longrightarrow \text{trans_tm } [i] \ t = \text{trans_tm } [] \ t$
by (induct t rule: tm.induct, auto simp: fresh_Pair)

nominal_function (invariant $\lambda(xs, _) y. \text{atom } ' \text{set } xs \ \#* \ y$)
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)
| *atom* k $\# e \Longrightarrow \text{trans_fm } e \ (\text{Ex } k \ A) = \text{DBEx } (\text{trans_fm } (k\#e) \ A)$
supply [[*simproc* del: *defined_all*]]
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]: (trans_tm e t = trans_tm e u)  $\longleftrightarrow$  t = u
proof (induct t arbitrary: e u rule: tm.induct)
  case Zero show ?case
    apply (cases u rule: tm.exhaust, auto)
    apply (metis dbtm.distinct(1) dbtm.distinct(3) lookup_in lookup_notin)
    done
next
  case (Var i) show ?case
    apply (cases u rule: tm.exhaust, auto)
    apply (metis dbtm.distinct(1) dbtm.distinct(3) lookup_in lookup_notin)
    apply (metis dbtm.distinct(10) dbtm.distinct(11) lookup_in lookup_notin)
    done
next
  case (Eats tm1 tm2) thus ?case
    apply (cases u rule: tm.exhaust, auto)
    apply (metis dbtm.distinct(12) dbtm.distinct(9) lookup_in lookup_notin)
    done
qed

lemma trans_fm_inject [iff]: (trans_fm e A = trans_fm e B)  $\longleftrightarrow$  A = B
proof (nominal_induct A avoiding: e B rule: fm.strong_induct)
  case (Mem tm1 tm2) thus ?case
    by (rule fm.strong_exhaust [where y=B and c=e]) (auto simp: fresh_star_def)
next
  case (Eq tm1 tm2) thus ?case
    by (rule fm.strong_exhaust [where y=B and c=e]) (auto simp: fresh_star_def)
next
  case (Disj fm1 fm2) show ?case
    by (rule fm.strong_exhaust [where y=B and c=e]) (auto simp: Disj fresh_star_def)
next
  case (Neg fm) show ?case
    by (rule fm.strong_exhaust [where y=B and c=e]) (auto simp: Neg fresh_star_def)
next
  case (Ex name fm)
    thus ?case using [[simproc del: alpha_lst]]
    proof (cases rule: fm.strong_exhaust [where y=B and c=(e, name)], simp_all add: fresh_star_def)
      fix name'::name and fm'::fm
      assume name': atom name'  $\#$  (e, name)
      assume atom name  $\#$  fm'  $\vee$  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  $\leftrightarrow$  name')  $\cdot$  trans_fm (name'  $\#$  e) fm' = trans_fm (name'  $\#$  e) fm'
        by (simp add: flip_fresh_fresh name)
      have eq2: (name  $\leftrightarrow$  name')  $\cdot$  ([[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  $\leftrightarrow$  name')  $\cdot$  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  $(i \leftrightarrow c) \cdot A = (j \leftrightarrow c) \cdot B$ 
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  $((i \leftrightarrow c) \cdot (\text{trans\_fm [i] A})) = ((j \leftrightarrow c) \cdot \text{trans\_fm [j] B})$ 
    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  $(i \leftrightarrow c) \cdot A = (j \leftrightarrow c) \cdot B$  by simp
qed

```

2.2 Characterising the Well-Formed de Bruijn Formulas

2.2.1 Well-Formed Terms

```

inductive wf_dbtm :: dbtm ⇒ 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  $\exists t::tm. x = \text{trans\_tm [] } t$ 
using assms
proof (induct rule: wf_dbtm.induct)
  case Zero thus ?case
    by (metis trans_tm.simps(1))
next
  case (Var i) thus ?case
    by (metis lookup.simps(1) trans_tm.simps(2))
next
  case (Eats dt1 dt2) thus ?case
    by (metis trans_tm.simps(3))
qed

```

lemma *wf_dbtm_trans_tm*: *wf_dbtm (trans_tm [] t)*

by (induct t rule: tm.induct) auto

theorem *wf_dbtm_iff_is_tm*: $wf_dbtm\ x \longleftrightarrow (\exists t::tm.\ x = trans_tm\ []\ t)$
 by (metis *wf_dbtm_imp_is_tm wf_dbtm_trans_tm*)

nominal_function *abst_dbtm* :: $name \Rightarrow nat \Rightarrow dbtm \Rightarrow dbtm$
 where
 | *abst_dbtm* name i DBZero = DBZero
 | *abst_dbtm* name i (DBVar name') = (if name = name' then DBInd i else DBVar name')
 | *abst_dbtm* name i (DBInd j) = DBInd j
 | *abst_dbtm* name i (DBEats t1 t2) = DBEats (*abst_dbtm* name i t1) (*abst_dbtm* name i t2)
apply (simp add: eqvt_def *abst_dbtm_graph_aux_def*, auto)
apply (metis *dbtm.exhaust*)
done

nominal_termination (*eqvt*)
 by *lexicographic_order*

nominal_function *subst_dbtm* :: $dbtm \Rightarrow name \Rightarrow dbtm \Rightarrow dbtm$
 where
 | *subst_dbtm* u i DBZero = DBZero
 | *subst_dbtm* u i (DBVar name) = (if i = name then u else DBVar name)
 | *subst_dbtm* u i (DBInd j) = DBInd j
 | *subst_dbtm* u i (DBEats t1 t2) = DBEats (*subst_dbtm* u i t1) (*subst_dbtm* u i t2)
by (auto simp: eqvt_def *subst_dbtm_graph_aux_def*) (metis *dbtm.exhaust*)

nominal_termination (*eqvt*)
 by *lexicographic_order*

lemma *fresh_iff_non_subst_dbtm*: $subst_dbtm\ DBZero\ i\ t = t \longleftrightarrow atom\ i\ \#\ t$
 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
 | *abst_dbfm* name i (DBMem t1 t2) = DBMem (*abst_dbtm* name i t1) (*abst_dbtm* name i t2)
 | *abst_dbfm* name i (DBEq t1 t2) = DBEq (*abst_dbtm* name i t1) (*abst_dbtm* name i t2)
 | *abst_dbfm* name i (DBDisj A1 A2) = DBDisj (*abst_dbfm* name i A1) (*abst_dbfm* name i A2)
 | *abst_dbfm* name i (DBNeg A) = DBNeg (*abst_dbfm* name i A)
 | *abst_dbfm* name i (DBEx A) = DBEx (*abst_dbfm* name (i+1) A)
apply (simp add: eqvt_def *abst_dbfm_graph_aux_def*, auto)
apply (metis *dbfm.exhaust*)
done

nominal_termination (*eqvt*)
 by *lexicographic_order*

nominal_function *subst_dbfm* :: $dbtm \Rightarrow name \Rightarrow dbfm \Rightarrow dbfm$
 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:  $i \neq j \Longrightarrow$  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_dbtm_imp_is_tm)
next
  case (Eq t1 t2) thus ?case
  by (metis trans_fm.simps(2) wf_dbtm_imp_is_tm)
next
  case (Disj fm1 fm2) thus ?case
  by (metis trans_fm.simps(3))
next
  case (Neg fm) thus ?case
  by (metis trans_fm.simps(4))
next
  case (Ex fm name) thus ?case
  apply auto
  apply (rule_tac x=Ex name A in exI)
  apply (auto simp: abst_trans_fm)
  done
qed

```

```

lemma wf_dbfm_trans_fm: wf_dbfm (trans_fm [] A)
  apply (nominal_induct A rule: fm.strong_induct)
  apply (auto simp: wf_dbtm_trans_tm abst_trans_fm)
  apply (metis abst_trans_fm wf_dbfm.Ex)
  done

```

```

lemma wf_dbfm_iff_is_fm: wf_dbfm x  $\longleftrightarrow$  ( $\exists A::fm. x = trans\_fm \ [] \ A$ )
  by (metis wf_dbfm_imp_is_fm wf_dbfm_trans_fm)

```

```

lemma dbtm_abst_ignore [simp]:
  abst_dbtm name i (abst_dbtm name j t) = abst_dbtm name j t
  by (induct t rule: dbtm.induct) auto

```

```

lemma abst_dbtm_fresh_ignore [simp]: atom name  $\#$  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_dbtm.simps(2))
  done

```

```

lemma subst_fm_trans_commute [simp]:
  subst_dbfm (trans_tm [] u) name (trans_fm [] A) = trans_fm [] (A (name::= u))
apply (nominal_induct A avoiding: name u rule: fm.strong_induct)
apply (auto simp: lookup_notin abst_trans_fm [symmetric])
apply (metis dbfm_abst_swap_subst fresh_at_base(2) fresh_trans_tm_iff)
done

```

```

lemma subst_fm_trans_commute_eq:
  du = trans_tm [] u  $\implies$  subst_dbfm du i (trans_fm [] A) = trans_fm [] (A(i::=u))
by (metis subst_fm_trans_commute)

```

2.4 Quotations

```

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

```

```

lemma fresh_HTuple [simp]: x  $\#$  HTuple n
by (induct n) auto

```

```

lemma HTuple_eqvt[eqvt]: (p  $\cdot$  HTuple n) = HTuple (p  $\cdot$  n)
by (induct n, auto simp: HPair_eqvt permute_pure)

```

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]: nat_of_name n1 = 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]: nat_of_name (Abs_name (Atom (Sort "SyntaxN.name" [])) n)
= n
by (auto simp: nat_of_name_def atom_name_def Abs_name_inverse)

```

```

lemma nat_of_name_name_eq [simp]: nat_of_name (name_of_nat n) = n
by (simp add: name_of_nat_def)

```

```

lemma name_of_nat_nat_of_name [simp]: name_of_nat (nat_of_name i) = i
by (metis nat_of_name_inject nat_of_name_name_eq)

```

```

lemma HPair_neq_ORD_OF [simp]: HPair x y  $\neq$  ORD_OF i
by (metis HPair_def ORD_OF.elims SUCC_def tm.distinct(3) tm.eq_iff(3))

```

Infinite support, so we cannot use nominal primrec.

```

function quot_dbtm :: dbtm  $\Rightarrow$  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*

2.4.2 Quotations of de Bruijn formulas

Infinite support, so we cannot use nominal primrec.

```
function quot_dbfm :: dbfm  $\Rightarrow$  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:  $n > 0 \implies$  HTuple n = HPair Zero (HTuple (n - 1))
by (metis Suc_diff_1 HTuple.simps(2))
```

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

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]:  $s \#$  (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)
```

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** $s \# \langle A \rangle$
by (simp add: quot_fm_def)

lemma *quot_fm_permute* [simp]: **fixes** $A::fm$ **shows** $p \cdot \langle A \rangle = \langle A \rangle$
by (metis fresh_star_def perm_supp_eq quot_fm_fresh)

lemma *quot_Mem*: $\langle x \text{ IN } y \rangle = \text{HPair } (\text{HTuple } 0) (\text{HPair } (\langle x \rangle) (\langle y \rangle))$
by (simp add: quot_fm_def quot_tm_def)

lemma *quot_Eq*: $\langle x \text{ EQ } y \rangle = \text{HPair } (\text{HTuple } 2) (\text{HPair } (\langle x \rangle) (\langle y \rangle))$
by (simp add: quot_fm_def quot_tm_def)

lemma *quot_Disj*: $\langle A \text{ OR } B \rangle = \text{HPair } (\text{HTuple } 3) (\text{HPair } (\langle A \rangle) (\langle B \rangle))$
by (simp add: quot_fm_def)

lemma *quot_Neg*: $\langle \text{Neg } A \rangle = \text{HPair } (\text{HTuple } 4) (\langle A \rangle)$
by (simp add: quot_fm_def)

lemma *quot_Ex*: $\langle \text{Ex } i \ A \rangle = \text{HPair } (\text{HTuple } 5) (\text{quot_dbfm } (\text{trans_fm } [i] \ A))$
by (simp add: 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

abbreviation *Q_Eats* :: $tm \Rightarrow tm \Rightarrow tm$
where $Q_Eats \ t \ u \equiv \text{HPair } (\text{HTuple } (\text{Suc } 0)) (\text{HPair } \ t \ u)$

abbreviation *Q_Succ* :: $tm \Rightarrow tm$
where $Q_Succ \ t \equiv Q_Eats \ t \ t$

lemma *quot_Succ*: $\langle \text{SUCC } x \rangle = Q_Succ \ \langle x \rangle$
by (auto simp: SUCC_def quot_Eats)

abbreviation *Q_HPPair* :: $tm \Rightarrow tm \Rightarrow tm$
where $Q_HPair \ t \ u \equiv$
 $Q_Eats \ (Q_Eats \ \text{Zero} \ (Q_Eats \ (Q_Eats \ \text{Zero} \ u) \ t))$
 $(Q_Eats \ (Q_Eats \ \text{Zero} \ t) \ t)$

abbreviation *Q_Mem* :: $tm \Rightarrow tm \Rightarrow tm$
where $Q_Mem \ t \ u \equiv \text{HPair } (\text{HTuple } 0) (\text{HPair } \ t \ u)$

abbreviation *Q_Eq* :: $tm \Rightarrow tm \Rightarrow tm$
where $Q_Eq \ t \ u \equiv \text{HPair } (\text{HTuple } 2) (\text{HPair } \ t \ u)$

abbreviation *Q_Disj* :: $tm \Rightarrow tm \Rightarrow tm$
where $Q_Disj \ t \ u \equiv \text{HPair } (\text{HTuple } 3) (\text{HPair } \ t \ u)$

abbreviation *Q_Neg* :: $tm \Rightarrow tm$
where $Q_Neg \ t \equiv \text{HPair } (\text{HTuple } 4) \ t$

abbreviation *Q_Conj* :: $tm \Rightarrow tm \Rightarrow tm$
where $Q_Conj \ t \ u \equiv Q_Neg \ (Q_Disj \ (Q_Neg \ t) \ (Q_Neg \ u))$

abbreviation *Q_Imp* :: $tm \Rightarrow tm \Rightarrow tm$
where $Q_Imp \ t \ u \equiv Q_Disj \ (Q_Neg \ t) \ u$

abbreviation $Q_Ex :: tm \Rightarrow tm$
where $Q_Ex\ t \equiv HPair\ (HTuple\ 5)\ t$

abbreviation $Q_All :: tm \Rightarrow tm$
where $Q_All\ t \equiv Q_Neg\ (Q_Ex\ (Q_Neg\ t))$

lemma $quot_subst_eq: \langle A(i::=t) \rangle = quot_dbfm\ (subst_dbfm\ (trans_tm\ []\ t)\ i\ (trans_fm\ []\ A))$
by $(metis\ quot_fm_def\ subst_fm_trans_commute)$

lemma $Q_Succ_cong: H \vdash x\ EQ\ x' \Longrightarrow H \vdash Q_Succ\ x\ EQ\ Q_Succ\ x'$
by $(metis\ HPair_cong\ Refl)$

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

inductive $coding_tm :: tm \Rightarrow bool$
where
 $Ord: \exists i. x = ORD_OF\ i \Longrightarrow coding_tm\ x$
 $| HPair: coding_tm\ x \Longrightarrow coding_tm\ y \Longrightarrow coding_tm\ (HPair\ x\ y)$

declare $coding_tm.intros [intro]$

lemma $coding_tm_Zero [intro]: coding_tm\ Zero$
by $(metis\ ORD_OF.simps(1)\ Ord)$

lemma $coding_tm_HTuple [intro]: coding_tm\ (HTuple\ k)$
by $(induct\ k,\ auto)$

inductive_simps $coding_tm_HPair [simp]: coding_tm\ (HPair\ x\ y)$

lemma $quot_dbtm_coding [simp]: coding_tm\ (quot_dbtm\ t)$
apply $(induct\ t\ rule: dbtm.induct,\ auto)$
apply $(metis\ ORD_OF.simps(2)\ Ord)$
done

lemma $quot_dbfm_coding [simp]: coding_tm\ (quot_dbfm\ fm)$
by $(induct\ fm\ rule: dbfm.induct,\ auto)$

lemma $quot_fm_coding: fixes\ A::fm\ shows\ coding_tm\ \langle A \rangle$
by $(metis\ quot_dbfm_coding\ quot_fm_def)$

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

Infinite support, so we cannot use nominal primrec.

function $vquot_dbtm :: name\ set \Rightarrow dbtm \Rightarrow tm$
where
 $vquot_dbtm\ V\ DBZero = Zero$
 $| vquot_dbtm\ V\ (DBVar\ name) = (if\ name \in V\ then\ Var\ name$
 $\quad\quad\quad else\ ORD_OF\ (Suc\ (nat_of_name\ name)))$
 $| vquot_dbtm\ V\ (DBInd\ k) = HPair\ (HTuple\ 6)\ (ORD_OF\ k)$
 $| vquot_dbtm\ V\ (DBEats\ t\ u) = HPair\ (HTuple\ 1)\ (HPair\ (vquot_dbtm\ V\ t)\ (vquot_dbtm\ V\ u))$
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** [*t*]{ } = «*t*»
by (simp add: *vquot_tm_def quot_tm_def*)

lemma *vquot_dbtm_eq*: *atom* '*V* \cap *supp* *t* = *atom* '*W* \cap *supp* *t* \Longrightarrow *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** $i \# [A]V \longleftrightarrow i \# A \vee i \notin atom\ 'V$
by (simp add: *vquot_fm_def*)

lemma *vquot_dbfm_empty* [simp]: *vquot_dbfm* {} *A* = *quot_dbfm* *A*
by (induct *A* rule: *dbfm.induct*) auto

lemma *vquot_fm_empty* [simp]: **fixes** *A*::*fm* **shows** [*A*]{ } = «*A*»
by (simp add: *vquot_fm_def quot_fm_def*)

lemma *vquot_dbfm_eq*: *atom* '*V* \cap *supp* *A* = *atom* '*W* \cap *supp* *A* \Longrightarrow *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** $atom\ i \notin\ supp\ A \implies \llbracket A \rrbracket (insert\ i\ V) = \llbracket A \rrbracket V$

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 subst_fm_Subset [simp]: (t SUBS u)(i::=x) = (subst i x t) SUBS (subst i x u)
  proof -
    obtain j::name where atom j  $\#$  (i,x,t,u)
      by (rule obtain_fresh)
    thus ?thesis
      by (auto simp: Subset.simps [of j])
  qed
```

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

```
lemma Subset_D:
  assumes major: H  $\vdash$  t SUBS u and minor: H  $\vdash$  a IN t shows H  $\vdash$  a IN u
  proof -
    obtain i::name where i: atom i  $\#$  (t, u)
      by (rule obtain_fresh)
    hence H  $\vdash$  (Var i IN t IMP Var i IN u) (i::=a)
      by (metis Subset.simps major All_D)
    thus ?thesis
      using i by simp (metis MP_same minor)
```

qed

lemma *Subset_E*: $H \vdash t \text{ SUBS } u \implies H \vdash a \text{ IN } t \implies \text{insert } (a \text{ IN } u) H \vdash A \implies H \vdash A$
by (metis *Subset_D 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 (rule *P2_cong*) 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 (metis *Assume Subset_D cut_same insert_absorb*)

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

lemma *Zero_SubsetE*: $H \vdash A \implies \text{insert } (\text{Zero SUBS } X) H \vdash A$
by (rule *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 (rule *obtain_fresh*)
 have $\{t \text{ SUBS } \text{Zero}\} \vdash t \text{ EQ } \text{Zero}$
 proof (rule *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 {Eats x y SUBS z} ⊢ y IN z
  by (metis Subset_D Assume Mem_Eats_I2 Refl)
qed
next
show {x SUBS z AND y IN z} ⊢ Eats x y SUBS z using i
  by (auto intro!: Subset_I [where i=i] intro: Subset_D Mem_cong [THEN Iff_MP2_same])
qed
thus ?thesis
  by (rule thin0)
qed

```

```

lemma Eats_Subset_I [intro!]: H ⊢ x SUBS z ⇒ H ⊢ y IN z ⇒ H ⊢ Eats x y SUBS z
  by (metis Conj_I Eats_Subset_Iff Iff_MP2_same)

```

```

lemma Eats_Subset_E [intro!]:
  insert (x SUBS z) (insert (y IN z) H) ⊢ C ⇒ insert (Eats x y SUBS z) H ⊢ 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 ⊢ x SUBS Eats x y
  by (metis Conj_E1 Eats_Subset_Iff Iff_MP_same Subset_refl)

```

```

lemma SUCC_Subset_I [intro!]: H ⊢ x SUBS z ⇒ H ⊢ x IN z ⇒ H ⊢ SUCC x SUBS z
  by (metis Eats_Subset_I SUCC_def)

```

```

lemma SUCC_Subset_E [intro!]:
  insert (x SUBS z) (insert (x IN z) H) ⊢ C ⇒ insert (SUCC x SUBS z) H ⊢ C
  by (metis Eats_Subset_E SUCC_def)

```

```

lemma Subset_trans0: { a SUBS b, b SUBS c } ⊢ a SUBS c

```

```

proof –
  obtain i::name where [simp]: atom i ‡ (a,b,c)
  by (rule obtain_fresh)
  show ?thesis
  by (rule Subset_I [of i]) (auto intro: Subset_D)
qed

```

```

lemma Subset_trans: H ⊢ a SUBS b ⇒ H ⊢ b SUBS c ⇒ H ⊢ a SUBS c
  by (metis Subset_trans0 cut2)

```

```

lemma Subset_SUCC: H ⊢ a SUBS (SUCC a)
  by (metis SUCC_def Subset_Eats_I)

```

```

lemma All2_Subset_lemma: atom l ‡ (k',k) ⇒ {P} ⊢ P' ⇒ {All2 l k P, k' SUBS k} ⊢ All2 l k' P'
  apply auto
  apply (rule Ex_I [where x = Var l])
  apply (auto intro: ContraProve Set_MP cut1)
  done

```

```

lemma All2_Subset: [H ⊢ All2 l k P; H ⊢ k' SUBS k; {P} ⊢ P'; atom l ‡ (k', k)] ⇒ H ⊢ All2 l k' P'
  by (rule cut2 [OF All2_Subset_lemma]) auto

```

3.2 Extensionality

```

lemma Extensionality: H ⊢ x EQ y IFF x SUBS y AND y SUBS x

```

```

proof –
  obtain i::name and j::name and k::name

```



```

where atoms: atom i # (x,y) atom j # (i,x,y) atom k # (i,j,y)
by (metis obtain_fresh)
have {} ⊢ (Var i EQ y IFF Var i SUBS y AND y SUBS Var i) (is {} ⊢ ?scheme)
proof (rule Ind [of j])
  show atom j # (i, ?scheme) using atoms
  by simp
next
show {} ⊢ ?scheme(i::=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 ⇒ tm ⇒ fm
where atom z # (t, u) ⇒ Disjoint t u = All2 z t (Neg ((Var z) IN u))
by (auto simp: eqvt_def Disjoint_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)

```

nominal_termination (*eqvt*)
 by *lexicographic_order*

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

lemma *Disjoint_fresh_iff* [*simp*]: $a \# \text{Disjoint } t \ u \longleftrightarrow a \# t \wedge a \# u$
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*]:
 $(\text{Disjoint } t \ u)(i::=x) = \text{Disjoint } (\text{subst } i \ x \ t) \ (\text{subst } i \ x \ u)$
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*: $H \vdash t \ EQ \ t' \Longrightarrow H \vdash u \ EQ \ u' \Longrightarrow H \vdash \text{Disjoint } t \ u \ IFF \text{Disjoint } t' \ u'$
 by (*rule P2_cong*) *auto*

lemma *Disjoint_I*:
assumes *insert ((Var i) IN t) (insert ((Var i) IN u) H) ⊢ Fls*
atom i # (t,u) ∀ B ∈ H. atom i # B
shows $H \vdash \text{Disjoint } t \ u$
 by (*subst Disjoint.simps [of i]*) (*auto simp: assms insert_commute*)

lemma *Disjoint_E*:
assumes *major: H ⊢ Disjoint t u and minor: H ⊢ a IN t H ⊢ a IN u* **shows** $H \vdash A$
proof –
obtain *i::name* **where** *i: atom i # (t, u)*
 by (*rule obtain_fresh*)
hence $H \vdash (\text{Var } i \ IN \ t \ IMP \ Neg \ (\text{Var } i \ IN \ u)) \ (i::=a)$
 by (*metis Disjoint.simps major All_D*)
thus *?thesis using i*
 by *simp (metis MP_same Neg_D minor)*
qed

lemma *Disjoint_commute*: $\{ \text{Disjoint } t \ u \} \vdash \text{Disjoint } u \ t$
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*: $H \vdash \text{Disjoint } t \ u \Longrightarrow H \vdash \text{Disjoint } u \ t$
 by (*metis Disjoint_commute cut1*)

lemma *Disjoint_commute_D*: $\text{insert } (\text{Disjoint } t \ u) \ H \vdash A \Longrightarrow \text{insert } (\text{Disjoint } u \ t) \ H \vdash A$
 by (*metis Assume Disjoint_commute_I cut_same insert_commute thin1*)

lemma *Zero_Disjoint_I1* [*iff*]: $H \vdash \text{Disjoint } Zero \ t$
proof –

```

obtain i::name where i: atom i # t
  by (rule obtain_fresh)
hence  $\{\} \vdash \text{Disjoint Zero } t$ 
  by (auto intro: Disjoint_I [of i])
thus ?thesis
  by (metis thin0)
qed

```

```

lemma Zero_Disjoint_I2 [iff]:  $H \vdash \text{Disjoint } t \text{ Zero}$ 
  by (metis Disjoint_commute Zero_Disjoint_I1 cut1)

```

```

lemma Disjoint_Eats_D1:  $\{ \text{Disjoint } (\text{Eats } x \ y) \ z \} \vdash \text{Disjoint } x \ z$ 
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:  $\{ \text{Disjoint } (\text{Eats } x \ y) \ z \} \vdash \text{Neg}(y \ IN \ z)$ 
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:
   $\text{insert } (\text{Disjoint } x \ z) (\text{insert } (\text{Neg}(y \ IN \ z)) \ H) \vdash A \implies \text{insert } (\text{Disjoint } (\text{Eats } x \ y) \ z) \ H \vdash A$ 
  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:
   $\text{insert } (\text{Disjoint } z \ x) (\text{insert } (\text{Neg}(y \ IN \ z)) \ H) \vdash A \implies \text{insert } (\text{Disjoint } z \ (\text{Eats } x \ y)) \ H \vdash A$ 
  by (metis Disjoint_Eats_E Disjoint_commute_D)

```

```

lemma Disjoint_Eats_Imp:  $\{ \text{Disjoint } x \ z, \text{Neg}(y \ IN \ z) \} \vdash \text{Disjoint } (\text{Eats } x \ y) \ z$ 
proof –
  obtain i::name where atom i # (x,y,z)
    by (rule obtain_fresh)
  then show ?thesis
    by (auto intro: Disjoint_I [of i] Disjoint_E [THEN rotate3]
      Mem_cong [OF Assume Refl, THEN Iff_MP_same])
qed

```

```

lemma Disjoint_Eats_I [intro!]:  $H \vdash \text{Disjoint } x \ z \implies \text{insert } (y \ IN \ z) \ H \vdash \text{Fls} \implies H \vdash \text{Disjoint } (\text{Eats } x \ y) \ z$ 
  by (metis Neg_I cut2 [OF Disjoint_Eats_Imp])

```

```

lemma Disjoint_Eats_I2 [intro!]:  $H \vdash \text{Disjoint } z \ x \implies \text{insert } (y \ IN \ z) \ H \vdash \text{Fls} \implies H \vdash \text{Disjoint } z \ (\text{Eats } x \ y)$ 
  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)) }  $\vdash$  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* \implies {} \vdash *All2 i z* (*Neg* (*Disjoint* (*Var i*) *z*)) *IMP z EQ Zero*

```

apply auto
apply (rule Eq_Zero_I)
apply (rule cut_same [where A = (Neg ((Var i) IN z) AND Disjoint (Var i) z)])
apply (rule Foundation_lemma [THEN cut1], auto)
done

```

lemma *Mem_Neg_refl*: {} \vdash *Neg* (*x IN x*)

```

proof –
  obtain i::name where i: atom i  $\#$  x
    by (metis obtain_fresh)
  have {}  $\vdash$  Disjoint x (Eats Zero x)
    apply (rule cut_same [OF Foundation [where z = Eats Zero x]]) using i
    apply auto
    apply (rule cut_same [where A = Disjoint x (Eats Zero x)])
    apply (metis Assume thin1 Disjoint_cong [OF Assume Refl, THEN Iff_MP_same])
    apply (metis Assume AssumeH(4) Disjoint_E Mem_Eats_I2 Refl)
    done
  thus ?thesis
    by (metis Disjoint_Eats_D2 Disjoint_commute cut_same)
qed

```

lemma *Mem_refl_E* [*intro!*]: *insert* (*x IN x*) *H* \vdash *A*

```

by (metis Disj_I1 Mem_Neg_refl anti_deduction thin0)

```

lemma *Mem_non_refl*: **assumes** *H* \vdash *x IN x* **shows** *H* \vdash *A*

```

by (metis Mem_refl_E assms cut_same)

```

lemma *Mem_Neg_sym*: { *x IN y*, *y IN x* } \vdash *Fls*

```

proof –
  obtain i::name where i: atom i  $\#$  (x,y)
    by (metis obtain_fresh)
  have {}  $\vdash$  Disjoint x (Eats Zero y) OR Disjoint y (Eats Zero x)
    apply (rule cut_same [OF Foundation [where i=i and z = Eats (Eats Zero y) x]]) using i
    apply (auto intro!: Disjoint_Eats_E2 [THEN rotate2])
    apply (rule Disj_I2, auto)
    apply (metis Assume EQ_imp_SUBS2 Subset_D insert_commute)
    apply (blast intro!: Disj_I1 Disjoint_cong [OF Hyp Refl, THEN Iff_MP_same])
    done
  thus ?thesis

```

by (auto intro: cut0 Disjoint_Eats_E2)
qed

lemma Mem_not_sym: insert (x IN y) (insert (y IN x) H) ⊢ A
by (rule cut_thin [OF Mem_Neg_sym]) auto

3.5 The Ordinal Property

nominal_function OrdP :: tm ⇒ fm
where $\llbracket \text{atom } y \# (x, z); \text{atom } z \# x \rrbracket \implies$
OrdP x = All2 y x ((Var y) SUBS x AND All2 z (Var y) ((Var z) SUBS (Var y)))
by (auto simp: eqvt_def OrdP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)

nominal_termination (eqvt)
by lexicographic_order

lemma
shows OrdP_fresh_iff [simp]: a # OrdP x ↔ a # x (is ?thesis1)
proof –
obtain z::name and y::name where atom z # x atom y # (x, z)
by (metis obtain_fresh)
thus ?thesis1
by (auto simp: OrdP.simps [of y _ z] Ord_def Transset_def)
qed

lemma subst_fm_OrdP [simp]: (OrdP t)(i:=x) = OrdP (subst i x t)
proof –
obtain z::name and y::name where atom z # (t,i,x) atom y # (t,i,x,z)
by (metis obtain_fresh)
thus ?thesis
by (auto simp: OrdP.simps [of y _ z])
qed

lemma OrdP_cong: H ⊢ x EQ x' ⇒ H ⊢ OrdP x IFF OrdP x'
by (rule P1_cong) auto

lemma OrdP_Mem_lemma:
assumes z: atom z # (k,l) and l: insert (OrdP k) H ⊢ l IN k
shows insert (OrdP k) H ⊢ l SUBS k AND All2 z l (Var z SUBS l)
proof –
obtain y::name where y: atom y # (k,l,z)
by (metis obtain_fresh)
have insert (OrdP k) H
⊢ (Var y IN k IMP (Var y SUBS k AND All2 z (Var y) (Var z SUBS Var y)))(y:=l)
by (rule All_D) (simp add: OrdP.simps [of y _ z] y z Assume)
also have ... = l IN k IMP (l SUBS k AND All2 z l (Var z SUBS l))
using y z by simp
finally show ?thesis
by (metis MP_same l)
qed

lemma OrdP_Mem_E:
assumes atom z # (k,l)
insert (OrdP k) H ⊢ l IN k
insert (l SUBS k) (insert (All2 z l (Var z SUBS l)) H) ⊢ A
shows insert (OrdP k) H ⊢ A
apply (rule OrdP_Mem_lemma [THEN cut_same])
apply (auto simp: insert_commute)

```

apply (blast intro: assms thin1)+
done

lemma OrdP_Mem_imp_Subset:
  assumes  $k: H \vdash k \text{ IN } l$  and  $l: H \vdash \text{OrdP } l$  shows  $H \vdash k \text{ SUBS } l$ 
  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:  $\{ k' \text{ IN } k, \text{OrdP } k \} \vdash \text{SUCC } k' \text{ SUBS } k$ 
  by auto (metis Assume thin1 OrdP_Mem_imp_Subset)

lemma SUCC_Subset_Ord:  $H \vdash k' \text{ IN } k \implies H \vdash \text{OrdP } k \implies H \vdash \text{SUCC } k' \text{ SUBS } k$ 
  by (blast intro!: cut2 [OF SUCC_Subset_Ord_lemma])

lemma OrdP_Trans_lemma:  $\{ \text{OrdP } k, i \text{ IN } j, j \text{ IN } k \} \vdash i \text{ IN } k$ 
proof -
  obtain  $m::\text{name}$  where  $\text{atom } m \# (i,j,k)$ 
  by (metis obtain_fresh)
  thus ?thesis
  by (auto intro: OrdP_Mem_E [of m k j] Subset_D [THEN rotate3])
qed

lemma OrdP_Trans:  $H \vdash \text{OrdP } k \implies H \vdash i \text{ IN } j \implies H \vdash j \text{ IN } k \implies H \vdash i \text{ IN } k$ 
  by (blast intro: cut3 [OF OrdP_Trans_lemma])

lemma Ord_IN_Ord0:
  assumes  $l: H \vdash l \text{ IN } k$ 
  shows  $\text{insert } (\text{OrdP } k) H \vdash \text{OrdP } l$ 
proof -
  obtain  $z::\text{name}$  and  $y::\text{name}$  where  $z: \text{atom } z \# (k,l)$  and  $y: \text{atom } y \# (k,l,z)$ 
  by (metis obtain_fresh)
  have  $\{ \text{Var } y \text{ IN } l, \text{OrdP } k, l \text{ IN } k \} \vdash \text{All2 } z \ ( \text{Var } y) \ ( \text{Var } z \text{ SUBS } \text{Var } y)$  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  $\{ \text{OrdP } k, l \text{ IN } k \} \vdash \text{OrdP } l$  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  $x = \text{Var } y$ , 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  $\text{insert } (\text{Var } y \text{ IN } x) H \vdash (\text{Var } y) \text{ SUBS } x$ 
  and  $\text{insert } (\text{Var } z \text{ IN } \text{Var } y) (\text{insert } (\text{Var } y \text{ IN } x) H) \vdash (\text{Var } z) \text{ SUBS } (\text{Var } y)$ 
  and  $\text{atom } y \# (x, z) \forall B \in H. \text{atom } y \# B \ \text{atom } z \# x \forall B \in H. \text{atom } z \# B$ 
  shows  $H \vdash \text{OrdP } x$ 
  using  $\text{assms}$  by auto

```

```

lemma OrdP_Zero [simp]:  $H \vdash \text{OrdP Zero}$ 
proof -
  obtain  $y::\text{name}$  and  $z::\text{name}$  where  $\text{atom } y \# z$ 
  by (rule obtain_fresh)
  hence  $\{\} \vdash \text{OrdP Zero}$ 
  by (auto intro: OrdP_I [of  $y \_ \_ z$ ])
  thus ?thesis
  by (metis thin0)
qed

lemma OrdP_SUCC_I0:  $\{\text{OrdP } k\} \vdash \text{OrdP (SUCC } k)$ 
proof -
  obtain  $w::\text{name}$  and  $y::\text{name}$  and  $z::\text{name}$  where  $\text{atoms: atom } w \# (k,y,z) \text{ atom } y \# (k,z) \text{ atom } z \# k$ 
  by (metis obtain_fresh)
  have 1:  $\{\text{Var } y \text{ IN SUCC } k, \text{OrdP } k\} \vdash \text{Var } y \text{ SUBS SUCC } k$ 
  apply (rule Mem_SUCC_E)
  apply (rule OrdP_Mem_E [of  $w \_ \text{Var } y, \text{THEN rotate2}$ ]) using atoms
  apply auto
  apply (metis Assume_Subset_SUCC Subset_trans)
  apply (metis EQ_imp_SUBS Subset_SUCC Subset_trans)
  done
  have in_case:  $\{\text{Var } y \text{ IN } k, \text{Var } z \text{ IN Var } y, \text{OrdP } k\} \vdash \text{Var } z \text{ SUBS Var } y$ 
  apply (rule OrdP_Mem_E [of  $w \_ \text{Var } y, \text{THEN rotate3}$ ]) using atoms
  apply (auto intro: All2_E [THEN thin1])
  done
  have  $\{\text{Var } y \text{ EQ } k, \text{Var } z \text{ IN } k, \text{OrdP } k\} \vdash \text{Var } z \text{ SUBS Var } y$ 
  by (metis AssumeH(2) AssumeH(3) EQ_imp_SUBS2 OrdP_Mem_imp_Subset Subset_trans)
  hence eq_case:  $\{\text{Var } y \text{ EQ } k, \text{Var } z \text{ IN Var } y, \text{OrdP } k\} \vdash \text{Var } z \text{ SUBS Var } y$ 
  by (rule cut3) (auto intro: EQ_imp_SUBS [THEN cut1] Subset_D)
  have 2:  $\{\text{Var } z \text{ IN Var } y, \text{Var } y \text{ IN SUCC } k, \text{OrdP } k\} \vdash \text{Var } z \text{ SUBS Var } y$ 
  by (metis rotate2 Mem_SUCC_E in_case eq_case)
  show ?thesis
  apply (rule OrdP_I [OF 1 2])
  using atoms apply auto
  done
qed

lemma OrdP_SUCC_I:  $H \vdash \text{OrdP } k \implies H \vdash \text{OrdP (SUCC } k)$ 
by (metis OrdP_SUCC_I0 cut1)

lemma Zero_In_OrdP:  $\{\text{OrdP } x\} \vdash x \text{ EQ Zero OR Zero IN } x$ 
proof -
  obtain  $i::\text{name}$  and  $j::\text{name}$ 
  where  $i: \text{atom } i \# x$  and  $j: \text{atom } j \# (x,i)$ 
  by (metis obtain_fresh)
  show ?thesis
  apply (rule cut_thin [where  $HB = \{\text{OrdP } x\}$ , 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  $A = \text{Var } i \text{ EQ Zero}$ ])
  prefer 2 apply (blast intro: Iff_MP_same [OF Mem_cong [OF Assume_Ref1]])
  apply (auto intro!: Eq_Zero_I [where  $i=j$ ] Ex_I [where  $x=\text{Var } i$ ])
  apply (blast intro: Disjoint_E Subset_D)
  done
qed

```

lemma *OrdP_HPPairE*: insert (OrdP (HPair x y)) H ⊢ A
proof –
 have { OrdP (HPair x y) } ⊢ A
 by (rule cut_same [OF Zero_In_OrdP]) (auto simp: HPair_def)
 thus ?thesis
 by (metis Assume cut1)
qed

lemmas *OrdP_HPPairEH* = OrdP_HPPairE OrdP_HPPairE [THEN rotate2] OrdP_HPPairE [THEN rotate3] OrdP_HPPairE [THEN rotate4] OrdP_HPPairE [THEN rotate5]
 OrdP_HPPairE [THEN rotate6] OrdP_HPPairE [THEN rotate7] OrdP_HPPairE [THEN rotate8] OrdP_HPPairE [THEN rotate9] OrdP_HPPairE [THEN rotate10]
declare *OrdP_HPPairEH* [intro!]

lemma *Zero_Eq_HPPairE*: insert (Zero EQ HPair x y) H ⊢ A
 by (metis Eats_EQ_Zero_E2 HPair_def)

lemmas *Zero_Eq_HPPairEH* = Zero_Eq_HPPairE Zero_Eq_HPPairE [THEN rotate2] Zero_Eq_HPPairE [THEN rotate3] Zero_Eq_HPPairE [THEN rotate4] Zero_Eq_HPPairE [THEN rotate5]
 Zero_Eq_HPPairE [THEN rotate6] Zero_Eq_HPPairE [THEN rotate7] Zero_Eq_HPPairE [THEN rotate8] Zero_Eq_HPPairE [THEN rotate9] Zero_Eq_HPPairE [THEN rotate10]
declare *Zero_Eq_HPPairEH* [intro!]

lemma *HPair_Eq_ZeroE*: insert (HPair x y EQ Zero) H ⊢ A
 by (metis Sym_L Zero_Eq_HPPairE)

lemmas *HPair_Eq_ZeroEH* = HPair_Eq_ZeroE HPair_Eq_ZeroE [THEN rotate2] HPair_Eq_ZeroE [THEN rotate3] HPair_Eq_ZeroE [THEN rotate4] HPair_Eq_ZeroE [THEN rotate5]
 HPair_Eq_ZeroE [THEN rotate6] HPair_Eq_ZeroE [THEN rotate7] HPair_Eq_ZeroE [THEN rotate8] HPair_Eq_ZeroE [THEN rotate9] HPair_Eq_ZeroE [THEN rotate10]
declare *HPair_Eq_ZeroEH* [intro!]

3.6 Induction on Ordinals

lemma *OrdInd_lemma*:
 assumes *j*: atom (*j*::name) ‡ (*i*,A)
 shows { OrdP (Var *i*) } ⊢ (All *i* (OrdP (Var *i*) IMP ((All2 *j* (Var *i*) (A(i::= Var *j*))) IMP A))) IMP A
proof –
 obtain *l*::name and *k*::name
 where *l*: atom *l* ‡ (*i*,*j*,A) and *k*: atom *k* ‡ (*i*,*j*,*l*,A)
 by (metis obtain_fresh)
 have { (All *i* (OrdP (Var *i*) IMP ((All2 *j* (Var *i*) (A(i::= Var *j*))) IMP A))) }
 ⊢ (All2 *l* (Var *i*) (OrdP (Var *l*) IMP A(i::= Var *l*)))
 apply (rule Ind [of *k*])
 using *j k l* **apply** auto
 apply (rule All_E [where *x*=Var *l*, THEN rotate5], auto)
 apply (metis Assume Disj_I1 anti_deduction thin1)
 apply (rule Ex_I [where *x*=Var *l*], auto)
 apply (rule All_E [where *x*=Var *j*, THEN rotate6], auto)
 apply (blast intro: ContraProve Iff_MP_same [OF Mem_cong [OF Ref]])
 apply (metis Assume Ord_IN_Ord0 ContraProve insert_commute)
 apply (metis Assume Neg_D thin1)+
 done
 hence { (All *i* (OrdP (Var *i*) IMP ((All2 *j* (Var *i*) (A(i::= Var *j*))) IMP A))) }
 ⊢ (All2 *l* (Var *i*) (OrdP (Var *l*) IMP A(i::= Var *l*)))(i::= Eats Zero (Var *i*))
 by (rule Subst, auto)
 hence *indlem*: { All *i* (OrdP (Var *i*) IMP ((All2 *j* (Var *i*) (A(i::= Var *j*))) IMP A)) }
 ⊢ All2 *l* (Eats Zero (Var *i*) (OrdP (Var *l*) IMP A(i::= Var *l*)))


```

    using j l by simp
  show ?thesis
    apply (rule Imp_I)
    apply (rule cut_thin [OF indlem, where HB = {OrdP (Var i)}])
    apply (rule All2_Eats_E) using j l
    apply auto
  done
qed

```

```

lemma OrdInd:
  assumes j: atom (j::name) # (i,A)
  and x: H ⊢ OrdP (Var i) and step: H ⊢ All i (OrdP (Var i) IMP (All2 j (Var i) (A(i::= Var j)) IMP
  A))
  shows H ⊢ A
  apply (rule cut_thin [OF x, where HB=H])
  apply (rule MP_thin [OF OrdInd_lemma step])
  apply (auto simp: j)
  done

```

```

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

```

3.7 Linearity of Ordinals

```

lemma OrdP_linear_lemma:
  assumes j: atom j # i
  shows { OrdP (Var i) } ⊢ All j (OrdP (Var j) IMP (Var i IN Var j OR Var i EQ Var j OR Var j IN
  Var i))
  (is _ ⊢ ?scheme)
proof -
  obtain k::name and l::name and m::name
  where k: atom k # (i,j) and l: atom l # (i,j,k) and m: atom m # (i,j)
  by (metis obtain_fresh)
  show ?thesis
  proof (rule OrdIndH [where i=i and j=k])
    show atom k # (i, ?scheme)
    using k by (force simp add: fresh_Pair)
  next
    show {} ⊢ All i (OrdP (Var i) IMP (All2 k (Var i) (?scheme(i::= Var k)) IMP ?scheme))
    using j k
    apply simp
    apply (rule All_I Imp_I)+
    defer 1
    apply auto [2]
    apply (rule OrdIndH [where i=j and j=l]) using l
    — nested induction
    apply (force simp add: fresh_Pair)
    apply simp
    apply (rule All_I Imp_I)+
    prefer 2 apply force
    apply (rule Disj_3I)
    apply (rule Equality_I)
    — Now the opposite inclusion, Var j SUBS Var i
    apply (rule Subset_I [where i=m])
    apply (rule All2_E [THEN rotate4]) using l m
  qed

```

```

apply auto
apply (blast intro: ContraProve [THEN rotate3] OrdP_Trans)
apply (blast intro: ContraProve [THEN rotate3] Mem_cong [OF Hyp Refl, THEN Iff_MP2_same])
— Now the opposite inclusion, Var i SUBS Var j
apply (rule Subset_I [where i=m])
apply (rule All2_E [THEN rotate6], auto)
apply (rule All_E [where x = Var j], auto)
apply (blast intro: ContraProve [THEN rotate4] Mem_cong [OF Hyp Refl, THEN Iff_MP_same])
apply (blast intro: ContraProve [THEN rotate4] OrdP_Trans)
done
qed
qed

```

```

lemma OrdP_linear_imp: {} ⊢ OrdP x IMP OrdP y IMP x IN y OR x EQ y OR y IN x
proof —
obtain i::name and j::name
  where atoms: atom i # (x,y) atom j # (x,y,i)
  by (metis obtain_fresh)
have { OrdP (Var i) } ⊢ (OrdP (Var j) IMP (Var i IN Var j OR Var i EQ Var j OR Var j IN Var
i))(j::=y)
  using atoms by (metis All_D OrdP_linear_lemma fresh_Pair)
hence {} ⊢ OrdP (Var i) IMP OrdP y IMP (Var i IN y OR Var i EQ y OR y IN Var i)
  using atoms by auto
hence {} ⊢ (OrdP (Var i) IMP OrdP y IMP (Var i IN y OR Var i EQ y OR y IN Var i))(i::=x)
  by (metis Subst empty_iff)
thus ?thesis
  using atoms by auto
qed

```

```

lemma OrdP_linear:
assumes H ⊢ OrdP x H ⊢ OrdP y
  insert (x IN y) H ⊢ A insert (x EQ y) H ⊢ A insert (y IN x) H ⊢ A
shows H ⊢ A
proof —
have { OrdP x, OrdP y } ⊢ x IN y OR x EQ y OR y IN x
  by (metis OrdP_linear_imp Imp_Imp_commute anti_deduction)
thus ?thesis
  using assms by (metis cut2 Disj_E cut_same)
qed

```

```

lemma Zero_In_SUCC: {OrdP k} ⊢ Zero IN SUCC k
by (rule OrdP_linear [OF OrdP_Zero OrdP_SUCC_I]) (force simp: SUCC_def)+

```

3.8 The predicate *OrdNotEqP*

```

nominal_function OrdNotEqP :: tm ⇒ tm ⇒ fm (infixr NEQ 150)
where OrdNotEqP x y = OrdP x AND OrdP y AND (x IN y OR y IN x)
by (auto simp: eqvt_def OrdNotEqP_graph_aux_def)

```

```

nominal_termination (eqvt)
by lexicographic_order

```

```

lemma OrdNotEqP_fresh_iff [simp]: a # OrdNotEqP x y ⟷ a # x ∧ a # y
by auto

```

```

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

```

lemma *OrdNotEqP_cong*: $H \vdash x \text{ EQ } x' \implies H \vdash y \text{ EQ } y' \implies H \vdash \text{OrdNotEqP } x \ y \text{ IFF } \text{OrdNotEqP } x' \ y'$
by (rule *P2_cong*) *auto*

lemma *OrdNotEqP_self_contra*: $\{x \text{ NEQ } x\} \vdash \text{Fls}$
by *auto*

lemma *OrdNotEqP_OrdP_E*: $\text{insert } (\text{OrdP } x) (\text{insert } (\text{OrdP } y) H) \vdash A \implies \text{insert } (x \text{ NEQ } y) H \vdash A$
by (*auto intro: thin1 rotate2*)

lemma *OrdNotEqP_I*: $\text{insert } (x \text{ EQ } y) H \vdash \text{Fls} \implies H \vdash \text{OrdP } x \implies H \vdash \text{OrdP } y \implies H \vdash x \text{ NEQ } y$
by (rule *OrdP_linear* [of $_ x \ y$]) (*auto intro: ExFalso thin1 Disj_I1 Disj_I2*)

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

lemma *OrdNotEqP_imp_Neg_Eq*: $\{x \text{ NEQ } y\} \vdash \text{Neg } (x \text{ EQ } y)$
by (*blast intro: OrdNotEqP_cong [THEN Iff_MP2_same] OrdNotEqP_self_contra [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 ::*name*) $\#$ i **and** k : *atom* (k ::*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 ::*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 { OrdP  $x$ , Neg ( $x \text{ EQ } \text{Zero}$ ) }  $\vdash$   $\text{Ex } j$  ( $\text{Var } j \text{ IN } x \text{ AND } (\text{All2 } k \ x \ (\text{Var } k \ \text{SUBS } \text{Var } j))$ )
proof –
  obtain  $i$ ::name where  $i$ : atom  $i \# (x,j,k)$ 
  by (metis obtain_fresh)
  have {}  $\vdash$  ((Neg ( $\text{Var } i \text{ EQ } \text{Zero}$ ) AND (All2  $j$  ( $\text{Var } i$ ) (OrdP ( $\text{Var } j$ )))) IMP
    ( $\text{Ex } j$  ( $\text{Var } j \text{ IN } \text{Var } i \text{ AND } (\text{All2 } k \ (\text{Var } i) \ (\text{Var } k \ \text{SUBS } \text{Var } j))$ )))( $i::=x$ )
  apply (rule Subst [OF OrdP_set_max_lemma])
  using  $i$   $k$  apply auto
  done
  hence { Neg ( $x \text{ EQ } \text{Zero}$ ) AND (All2  $j$   $x$  (OrdP ( $\text{Var } j$ ))) }
     $\vdash$   $\text{Ex } j$  ( $\text{Var } j \text{ IN } x \text{ AND } (\text{All2 } k \ x \ (\text{Var } k \ \text{SUBS } \text{Var } j))$ )
  using  $i$   $j$   $k$  by simp (metis anti_deduction)
  hence { All2  $j$   $x$  (OrdP ( $\text{Var } j$ )), Neg ( $x \text{ EQ } \text{Zero}$ ) }
     $\vdash$   $\text{Ex } j$  ( $\text{Var } j \text{ IN } x \text{ AND } (\text{All2 } k \ x \ (\text{Var } k \ \text{SUBS } \text{Var } j))$ )
  by (rule cut1) (metis Assume Conj_I thin1)
  moreover have { OrdP  $x$  }  $\vdash$  All2  $j$   $x$  (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 { OrdP  $x$ , Neg ( $x \text{ EQ } \text{Zero}$ ) }  $\vdash$   $\text{Ex } p$  (OrdP ( $\text{Var } p$ ) AND  $x \text{ EQ } \text{SUCC}$  ( $\text{Var } p$ ))
proof –
  obtain  $j$ ::name and  $k$ ::name where  $j$ : atom  $j \# (x,p)$  and  $k$ : atom  $k \# (x,j,p)$ 
  by (metis obtain_fresh)
  show ?thesis
  apply (rule cut_same [OF OrdP_max_imp [of j x k]])
  using  $p$   $j$   $k$  apply auto
  apply (rule Ex_I [where x=Var j], auto)
  apply (metis Assume Ord_IN_Ord thin1)
  apply (rule cut_same [where A = OrdP (SUCC (Var j))])
  apply (metis Assume Ord_IN_Ord0 OrdP_SUCC_I rotate2 thin1)
  apply (rule OrdP_linear [where x = x, OF_ Assume], auto intro!: Mem_SUCC_EH)
  apply (metis Mem_not_sym rotate3)
  apply (rule Mem_non_refl, blast intro: Mem_cong [OF Assume Refl, THEN Iff_MP2_same])
  apply (force intro: thin1 All2_E [where x = SUCC (Var j), THEN rotate4])
  done
qed

```

```

lemma OrdP_cases_disj:

```

assumes p : atom $p \# x$
shows $\text{insert } (\text{OrdP } x) H \vdash x \text{ EQ Zero OR } \text{Ex } p (\text{OrdP } (\text{Var } p) \text{ AND } x \text{ EQ SUCC } (\text{Var } p))$
by (metis Disj_CI Assume cut2 [OF OrdP_cases_lemma [OF p]] Swap thin1)

lemma OrdP_cases_E:
 $\llbracket \text{insert } (x \text{ EQ Zero}) H \vdash A;$
 $\text{insert } (x \text{ EQ SUCC } (\text{Var } k)) (\text{insert } (\text{OrdP } (\text{Var } k)) H) \vdash A;$
 $\text{atom } k \# (x, A); \quad \forall C \in H. \text{atom } k \# C \rrbracket$
 $\implies \text{insert } (\text{OrdP } x) H \vdash A$
by (rule cut_same [OF OrdP_cases_disj [of k]]) (auto simp: insert_commute intro: thin1)

lemma OrdInd2_lemma:
 $\{ \text{OrdP } (\text{Var } i), A(i::= \text{Zero}), (\text{All } i (\text{OrdP } (\text{Var } i) \text{ IMP } A \text{ IMP } (A(i::= \text{SUCC } (\text{Var } i)))) \} \vdash A$
proof –

obtain $j::\text{name}$ **and** $k::\text{name}$ **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 $x=\text{Var } k$], simp)
apply (rule Neg_Imp_I, blast)
apply (rule cut_same [where $A = A(i::= \text{Var } k)$])
apply (rule All2_E [where $x = \text{Var } k$, THEN rotate5])
apply (auto intro: Mem_SUCC_I2 Mem_cong [OF Refl, THEN Iff_MP2_same])
apply (rule ContraProve [THEN rotate5])
by (metis Assume Iff_MP_left' Var_Eq_subst_Iff thin1)
qed

lemma OrdInd2:
assumes $H \vdash \text{OrdP } (\text{Var } i)$
and $H \vdash A(i::= \text{Zero})$
and $H \vdash \text{All } i (\text{OrdP } (\text{Var } i) \text{ IMP } A \text{ IMP } (A(i::= \text{SUCC } (\text{Var } i))))$
shows $H \vdash A$
by (metis cut3 [OF OrdInd2_lemma] assms)

lemma OrdInd2H:
assumes $H \vdash A(i::= \text{Zero})$
and $H \vdash \text{All } i (\text{OrdP } (\text{Var } i) \text{ IMP } A \text{ IMP } (A(i::= \text{SUCC } (\text{Var } i))))$
shows $\text{insert } (\text{OrdP } (\text{Var } i)) H \vdash A$
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 :: $\text{tm} \Rightarrow \text{fm}$

where $\llbracket \text{atom } z \# (r, z', x, y, x', y'); \text{atom } z' \# (r, x, y, x', y');$
 $\text{atom } x \# (r, y, x', y'); \text{atom } y \# (r, x', y'); \text{atom } x' \# (r, y'); \text{atom } y' \# (r) \rrbracket \implies$
 $\text{HFun_Sigma } r =$
 $\text{All2 } z \ r \ (\text{All2 } z' \ r \ (\text{Ex } x \ (\text{Ex } y \ (\text{Ex } x' \ (\text{Ex } y' \$
 $(\text{Var } z \ \text{EQ} \ \text{HPair} \ (\text{Var } x) \ (\text{Var } y) \ \text{AND} \ \text{Var } z' \ \text{EQ} \ \text{HPair} \ (\text{Var } x') \ (\text{Var } y')$
 $\ \text{AND} \ \text{OrdP} \ (\text{Var } x) \ \text{AND} \ \text{OrdP} \ (\text{Var } x') \ \text{AND}$
 $((\text{Var } x \ \text{EQ} \ \text{Var } x') \ \text{IMP} \ (\text{Var } y \ \text{EQ} \ \text{Var } y'))))))))$
by $(\text{auto simp: eqvt_def HFun_Sigma_graph_aux_def flip_fresh_fresh}) \ (\text{metis obtain_fresh})$

nominal_termination (eqvt)
by $\text{lexicographic_order}$

lemma
shows $\text{HFun_Sigma_fresh_iff} \ [\text{simp}]: a \# \text{HFun_Sigma } r \longleftrightarrow a \# r \ (\text{is } ?\text{thesis1})$
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 $(\text{metis obtain_fresh})$
thus $?\text{thesis1}$
by $(\text{auto simp: HBall_def hfun_sigma_ord_def})$
qed

lemma $\text{HFun_Sigma_subst} \ [\text{simp}]: (\text{HFun_Sigma } r)(i::=t) = \text{HFun_Sigma} \ (\text{subst } i \ t \ r)$
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 $\text{atom } z \# (r, t, i, z', x, y, x', y') \ \text{atom } z' \# (r, t, i, x, y, x', y')$
 $\text{atom } x \# (r, t, i, y, x', y') \ \text{atom } y \# (r, t, i, x', y')$
 $\text{atom } x' \# (r, t, i, y') \ \text{atom } y' \# (r, t, i)$
by $(\text{metis obtain_fresh})$
thus $?\text{thesis}$
by $(\text{auto simp: HFun_Sigma.simps [of } z _ z' \ x \ y \ x' \ y'])$
qed

lemma $\text{HFun_Sigma_Zero}: H \vdash \text{HFun_Sigma } \text{Zero}$
proof $-$
obtain $x::\text{name}$ **and** $y::\text{name}$ **and** $z::\text{name}$ **and** $x'::\text{name}$ **and** $y'::\text{name}$ **and** $z'::\text{name}$ **and** $z''::\text{name}$
where $\text{atom } z'' \# (z, z', x, y, x', y')$ $\text{atom } z \# (z', x, y, x', y')$ $\text{atom } z' \# (x, y, x', y')$
 $\text{atom } x \# (y, x', y')$ $\text{atom } y \# (x', y')$ $\text{atom } x' \# y'$
by $(\text{metis obtain_fresh})$
hence $\{\} \vdash \text{HFun_Sigma } \text{Zero}$
by $(\text{auto simp: HFun_Sigma.simps [of } z _ z' \ x \ y \ x' \ y'])$
thus $?\text{thesis}$
by (metis thin0)
qed

lemma $\text{Subset_HFun_Sigma}: \{\text{HFun_Sigma } s, s' \ \text{SUBS } s\} \vdash \text{HFun_Sigma } s'$
proof $-$
obtain $x::\text{name}$ **and** $y::\text{name}$ **and** $z::\text{name}$ **and** $x'::\text{name}$ **and** $y'::\text{name}$ **and** $z'::\text{name}$ **and** $z''::\text{name}$
where $\text{atom } z'' \# (z, z', x, y, x', y', s, s')$
 $\text{atom } z \# (z', x, y, x', y', s, s')$ $\text{atom } z' \# (x, y, x', y', s, s')$
 $\text{atom } x \# (y, x', y', s, s')$ $\text{atom } y \# (x', y', s, s')$
 $\text{atom } x' \# (y', s, s')$ $\text{atom } y' \# (s, s')$
by $(\text{metis obtain_fresh})$
thus $?\text{thesis}$
apply $(\text{auto simp: HFun_Sigma.simps [of } z _ z' \ x \ y \ x' \ y'])$
apply $(\text{rule Ex_I [where } x = \text{Var } z], \text{ auto})$

```

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} ⊢ ?concl using xy
  apply (auto simp: HFun_Sigma.simps [of z r z' x y x' y'])
  apply (rule All_E [where x=a], auto)
  apply (rule All_E [where x=a], simp)
  apply (rule Imp_E, blast)
  apply (rule Ex_EH Conj_EH)+
  apply simp_all
  apply (rule Ex_I [where x=Var x], simp)
  apply (rule Ex_I [where x=Var y], auto)
  done
  thus ?thesis
    by (rule cut2) (rule assms)+
qed

```

3.12 The predicate $HDomain_Incl$

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

```

nominal_function HDomain_Incl :: tm ⇒ tm ⇒ fm
  where [[atom x # (r,d,y,z); atom y # (r,d,z); atom z # (r,d)] ⇒⇒
    HDomain_Incl r d = All2 x d (Ex y (Ex z (Var z IN r AND Var z EQ HPair (Var x) (Var y))))
  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 ⇔ a # r ∧ a # d (is ?thesis1)
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
    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*: $H \vdash HDomain_Incl\ r\ d \implies H \vdash HDomain_Incl\ (Eats\ r\ (HPair\ d\ d'))\ (SUCC\ d)$
by (*metis HDomain_Incl_Eats cut1*)

3.13 *HPair* is Provably Injective

lemma *Doubleton_E*:
assumes *insert (a EQ c) (insert (b EQ d) H) \vdash A*


```

      insert (a EQ d) (insert (b EQ c) H) ⊢ 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 ⊢ HPair a b IN r
  and b': H ⊢ HPair a b' IN r
  shows H ⊢ b EQ b'

```

proof –

```

  obtain x::name and y::name and z::name and x'::name and y'::name and z'::name
  where atoms: atom z # (r,a,b,b',z',x,y,x',y') atom z' # (r,a,b,b',x,y,x',y')
    atom x # (r,a,b,b',y,x',y') atom y # (r,a,b,b',x',y')
    atom x' # (r,a,b,b',y') atom y' # (r,a,b,b')
  by (metis obtain_fresh)
  hence d1: H ⊢ All2 z r (All2 z' r (Ex x (Ex y (Ex x' (Ex y'
    (Var z EQ HPair (Var x) (Var y) AND Var z' EQ HPair (Var x') (Var y')
    AND OrdP (Var x) AND OrdP (Var x') AND ((Var x EQ Var x') IMP (Var y EQ Var
    y'))))))))
  using r HFun_Sigma.simps [of z r z' x y x' y']
  by simp
  have d2: H ⊢ All2 z' r (Ex x (Ex y (Ex x' (Ex y'
    (HPair a b EQ HPair (Var x) (Var y) AND Var z' EQ HPair (Var x') (Var y')
    AND OrdP (Var x) AND OrdP (Var x') AND ((Var x EQ Var x') IMP (Var y EQ Var
    y'))))))))
  using All_D [where x = HPair a b, OF d1] atoms
  by simp (metis MP_same b)
  have d4: H ⊢ Ex x (Ex y (Ex x' (Ex y'
    (HPair a b EQ HPair (Var x) (Var y) AND HPair a b' EQ HPair (Var x') (Var y')
    AND OrdP (Var x) AND OrdP (Var x') AND ((Var x EQ Var x') IMP (Var y EQ Var

```

```

y')))))))
  using All_D [where x = HPair a b', OF d2] atoms
  by simp (metis MP_same b')
  have d': { Ex x (Ex y (Ex x' (Ex y'
    (HPair a b EQ HPair (Var x) (Var y) AND HPair a b' EQ HPair (Var x') (Var y')
    AND OrdP (Var x) AND OrdP (Var x') AND ((Var x EQ Var x') IMP (Var y EQ Var y')))))))
} ⊢ b EQ b'
  using atoms
  by (auto intro: ContraProve Trans Sym)
  thus ?thesis
  by (rule cut_thin [OF d4], auto)
qed

```

3.14 SUCC is Provably Injective

```

lemma SUCC_SUBS_lemma: {SUCC x SUBS SUCC y} ⊢ x SUBS y
  apply (rule obtain_fresh [where x=(x,y)])
  apply (auto simp: SUCC_def)
  prefer 2 apply (metis Assume Conj_E1 Extensionality Iff_MP_same)
  apply (auto intro!: Subset_I)
  apply (blast intro: Set_MP cut_same [OF Mem_cong [OF Refl Assume, THEN Iff_MP2_same]]
    Mem_not_sym thin2)
done

```

```

lemma SUCC_SUBS: insert (SUCC x SUBS SUCC y) H ⊢ 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*: $H \vdash \text{OrdP } x \implies H \vdash \text{SUCC } y \text{ IN } \text{SUCC } x \implies H \vdash y \text{ IN } x$
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*: $\text{hfun_sigma_ord } s \longleftrightarrow \text{OrdDom } s \wedge \text{hfun_sigma } s$
by (*auto simp: hfun_sigma_ord_def OrdDom_def hfun_sigma_def HBall_def, metis+*)

lemma *hfun_sigma_iff*: $\text{hfun_sigma } r \longleftrightarrow \text{hfunction } r \wedge \text{hrelation } r$
by (*auto simp add: HBall_def hfun_sigma_def hfunction_def hrelation_def is_hpair_def, metis+*)

lemma *Seq_iff*: $\text{Seq } r \ d \longleftrightarrow d \leq \text{hdomain } r \wedge \text{hfun_sigma } r$
by (*auto simp: Seq_def hfun_sigma_iff*)

lemma *LstSeq_iff*: $\text{LstSeq } s \ k \ y \longleftrightarrow \text{succ } k \leq \text{hdomain } s \wedge \langle k, y \rangle \in s \wedge \text{hfun_sigma_ord } s$
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*]:

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

proof –

show *?thesis1*

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* (*SUCC* *k*))

(*insert* (*OrdP* *k*) (*insert* (*HFun_Sigma* *s*))

(*insert* (*HPair* *k* *y* *IN* *s*) *H*)) $\vdash B$

shows *insert* (*LstSeqP* *s* *k* *y*) *H* $\vdash B$

using *assms* **by** (*auto simp: insert_commute*)

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

lemma *LstSeqP_cong*:

assumes $H \vdash s \text{ EQ } s' \ H \vdash k \text{ EQ } k' \ H \vdash y \text{ EQ } y'$

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_fm_aux t S  $\wedge$  ground_fm_aux u S)
  ground_fm_aux (t EQ u) S = (ground_fm_aux t S  $\wedge$  ground_fm_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_fm_aux (A IFF B) S = (ground_fm_aux A S  $\wedge$  ground_fm_aux B S)
  ground_fm_aux (Neg A) S = (ground_fm_aux A S)
  ground_fm_aux (Ex x A) S = (ground_fm_aux A (S  $\cup$  {atom x}))
by (auto simp: ground_fm_aux_def ground_aux_def supp_conv_fresh)
```

```
lemma ground_fresh[simp]:
  ground t  $\Longrightarrow$  atom i  $\#$  t
  ground_fm A  $\Longrightarrow$  atom i  $\#$  A
unfolding ground_aux_def ground_fm_aux_def fresh_def
by simp_all
```

4.2 Sigma Formulas

Section 2 material

4.2.1 Strict Sigma Formulas

Definition 2.1

```
inductive ss_fm :: fm  $\Rightarrow$  bool where
  MemI: ss_fm (Var i IN Var j)
| DisjI: ss_fm A  $\Longrightarrow$  ss_fm B  $\Longrightarrow$  ss_fm (A OR B)
| ConjI: ss_fm A  $\Longrightarrow$  ss_fm B  $\Longrightarrow$  ss_fm (A AND B)
| ExI: ss_fm A  $\Longrightarrow$  ss_fm (Ex i A)
| All2I: ss_fm A  $\Longrightarrow$  atom j  $\#$  (i,A)  $\Longrightarrow$  ss_fm (All2 i (Var j) A)
```

equivariance *ss_fm*

nominal_inductive *ss_fm*

avoids *ExI*: *i* | *All2I*: *i*
by (*simp_all* *add*: *fresh_star_def*)

declare *ss_fm.intros* [*intro*]

definition *Sigma_fm* :: *fm* \Rightarrow *bool*

where *Sigma_fm* *A* \longleftrightarrow (\exists *B*. *ss_fm* *B* \wedge *supp* *B* \subseteq *supp* *A* \wedge $\{\}$ \vdash *A* IFF *B*)

lemma *Sigma_fm_Iff*: [$\{\}$ \vdash *B* IFF *A*; *supp* *A* \subseteq *supp* *B*; *Sigma_fm* *A*] \Longrightarrow *Sigma_fm* *B*
by (*metis* *Sigma_fm_def* *Iff_trans* *order_trans*)

lemma *ss_fm_imp_Sigma_fm* [*intro*]: *ss_fm* *A* \Longrightarrow *Sigma_fm* *A*
by (*metis* *Iff_refl* *Sigma_fm_def* *order_refl*)

lemma *Sigma_fm_Fls* [*iff*]: *Sigma_fm* *Fls*
by (*rule* *Sigma_fm_Iff* [*of* _ *Ex* *i* (Var *i* IN Var *i*)] *auto*)

4.2.2 Closure properties for Sigma-formulas

lemma

assumes *Sigma_fm* *A* *Sigma_fm* *B*
shows *Sigma_fm_AND* [*intro!*]: *Sigma_fm* (*A* AND *B*)
and *Sigma_fm_OR* [*intro!*]: *Sigma_fm* (*A* OR *B*)
and *Sigma_fm_Ex* [*intro!*]: *Sigma_fm* (Ex *i* *A*)

proof –

obtain *SA* *SB* **where** *ss_fm* *SA* $\{\}$ \vdash *A* IFF *SA* *supp* *SA* \subseteq *supp* *A*
and *ss_fm* *SB* $\{\}$ \vdash *B* IFF *SB* *supp* *SB* \subseteq *supp* *B*

using *assms* **by** (*auto simp* *add*: *Sigma_fm_def*)

then show *Sigma_fm* (*A* AND *B*) *Sigma_fm* (*A* OR *B*) *Sigma_fm* (Ex *i* *A*)

apply (*auto simp*: *Sigma_fm_def*)

apply (*metis* *ss_fm.ConjI* *Conj_cong* *Un_mono* *supp_Conj*)

apply (*metis* *ss_fm.DisjI* *Disj_cong* *Un_mono* *fm.supp*(3))

apply (*rule* *exI* [**where** *x* = Ex *i* *SA*])

apply (*auto intro!*: *Ex_cong*)

done

qed

lemma *Sigma_fm_All2_Var*:

assumes *H0*: *Sigma_fm* *A* **and** *ij*: *atom* *j* $\#$ (*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 {} ⊢ z EQ Eats x y IFF (All2 i z (Var i IN x OR Var i EQ y)) AND x SUBS z AND y IN z
proof (rule Iff_I, auto)
  have {Var i IN z, z EQ Eats x y} ⊢ 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} ⊢ Var i IN x OR Var i EQ y
  by (metis Iff_MP_same Mem_Eats_Iff)
next
  show {z EQ Eats x y} ⊢ 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  $\bigwedge j::name. \text{Sigma\_fm } (\text{Var } j \text{ IN } t)$ 
    and  $\bigwedge k::name. \text{Sigma\_fm } (\text{Var } k \text{ EQ } u)$ 
  shows  $\text{Sigma\_fm } (\text{Var } i \text{ SUBS Eats } t \ u)$ 
proof -
  obtain k::name where k: atom k # (t,u,Var i)
    by (metis obtain_fresh)
  hence  $\{\} \vdash \text{Var } i \text{ SUBS Eats } t \ u \text{ IFF } \text{All2 } k \ (\text{Var } i) \ (\text{Var } k \text{ IN } t \ \text{OR } \text{Var } k \text{ EQ } u)$ 
    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  $\bigwedge j::name. \text{Sigma\_fm } (\text{Var } j \text{ EQ } t)$ 
    and  $\bigwedge k::name. \text{Sigma\_fm } (\text{Var } k \text{ EQ } u)$ 
  shows  $\text{Sigma\_fm } (\text{Var } i \text{ EQ Eats } t \ u)$ 
proof -
  obtain j::name and k::name and l::name
    where atoms: atom j # (t,u,i) atom k # (t,u,i,j) atom l # (t,u,i,j,k)
    by (metis obtain_fresh)
  hence  $\{\} \vdash \text{Var } i \text{ EQ Eats } t \ u \text{ IFF}$ 
     $\text{Ex } j \ (\text{Ex } k \ (\text{Var } i \text{ EQ Eats } (\text{Var } j) \ (\text{Var } k) \ \text{AND } \text{Var } j \text{ EQ } t \ \text{AND } \text{Var } k \text{ EQ } u))$ 
    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  $\bigwedge j::name. \text{Sigma\_fm } (\text{Var } j \text{ EQ } t)$ 
    and  $\bigwedge k::name. \text{Sigma\_fm } (\text{Var } k \text{ EQ } u)$ 
  shows  $\text{Sigma\_fm } (\text{Eats } t \ u \text{ IN } \text{Var } i)$ 
proof -
  obtain j::name where j: atom j # (t,u,Var i)
    by (metis obtain_fresh)
  hence  $\{\} \vdash \text{Eats } t \ u \text{ IN } \text{Var } i \text{ IFF}$ 
     $\text{Ex } j \ (\text{Var } j \text{ IN } \text{Var } i \ \text{AND } \text{Var } j \text{ EQ Eats } t \ u)$ 
    apply (auto simp: fresh_Pair intro: Ex_I [where x=Eats t u])

```

```

apply (metis Assume Mem_cong [OF _ Refl, THEN Iff_MP_same] rotate2)
done
thus ?thesis
by (rule Sigma_fm_Iff) (auto simp: assms supp_conv_fresh Eq_Eats_sf)
qed

lemma Subset_Mem_sf_lemma:
  size t + size u < n  $\implies$  Sigma_fm (t SUBS u)  $\wedge$  Sigma_fm (t IN u)
proof (induction n arbitrary: t u rule: less_induct)
case (less n t u)
show ?case
proof
show Sigma_fm (t SUBS u)
proof (cases t rule: tm.exhaust)
case Zero thus ?thesis
by (auto intro: theorem_sf)
next
case (Var i) thus ?thesis using less.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 Sigma_fm (t IN u)
proof (cases u rule: tm.exhaust)
case Zero show ?thesis
by (rule Sigma_fm_Iff [where A=Fls]) (auto simp: supp_conv_fresh Zero)
next
case (Var i) show ?thesis
proof (cases t rule: tm.exhaust)
case Zero thus ?thesis using ⟨u = Var i⟩
by (auto intro: Zero_Mem_sf)
next
case (Var j)
thus ?thesis using ⟨u = Var i⟩
by auto
next
case (Eats t1 t2) thus ?thesis using ⟨u = Var i⟩ less.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
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 \implies atom j # (i,t,A) \implies
 $\{\} \vdash (All2\ i\ t\ A)\ IFF\ Ex\ j\ (Var\ j\ EQ\ t\ AND\ All2\ i\ (Var\ j)\ A)$*

apply *auto*

apply (*rule Ex_I [where x=t], auto*)

apply (*rule Ex_I [where x=Var i]*)

apply (*auto intro: ContraProve Mem_cong [THEN Iff_MP2_same]*)

done

lemma *Sigma_fm_All2 [intro!]:*
assumes *Sigma_fm A atom i # t*
shows *Sigma_fm (All2 i t A)*

proof –

obtain *j::name where j: atom j # (i,t,A)*

by (*metis obtain_fresh*)

show *Sigma_fm (All2 i t A)*

apply (*rule Sigma_fm_Iff [OF All2_term_Iff [of i t j]]*)

using *assms j*

apply (*auto simp: supp_conv_fresh Sigma_fm_All2_Var*)

done

qed

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

lemma *OrdP_sf [iff]: Sigma_fm (OrdP t)*

proof –

obtain *z::name and y::name where atom z # t atom y # (t, z)*

by (*metis obtain_fresh*)

thus *?thesis*

by (*auto simp: OrdP.simps*)

qed

lemma *OrdNotEqP_sf [iff]: Sigma_fm (OrdNotEqP t u)*

by (*auto simp: OrdNotEqP.simps*)

lemma *HDomain_Incl_sf [iff]: Sigma_fm (HDomain_Incl t u)*

proof –

obtain *x::name and y::name and z::name*

where *atom x # (t,u,y,z) atom y # (t,u,z) atom z # (t,u)*

by (*metis obtain_fresh*)

thus *?thesis*

by *auto*

qed

lemma *HFun_Sigma_Iff:*

assumes *atom z # (r,z',x,y,x',y') atom z' # (r,x,y,x',y')*

atom x # (r,y,x',y') atom y # (r,x',y')

atom x' # (r,y') atom y' # (r)

shows

```

{} ⊢ HFun_Sigma r IFF
  All2 z r (All2 z' r (Ex x (Ex y (Ex x' (Ex y'
    (Var z EQ HPair (Var x) (Var y) AND Var z' EQ HPair (Var x') (Var y')
    AND OrdP (Var x) AND OrdP (Var x') AND
    ((Var x NEQ Var x') OR (Var y EQ Var y'))))))))
apply (simp add: HFun_Sigma.simps [OF assms])
apply (rule Iff_refl All_cong Imp_cong Ex_cong)+
apply (rule Conj_cong [OF Iff_refl])
apply (rule Conj_cong [OF Iff_refl], auto)
apply (blast intro: Disj_I1 Neg_D OrdNotEqP_I)
apply (blast intro: Disj_I2)
apply (blast intro: OrdNotEqP_E rotate2)
done

lemma HFun_Sigma_sf [iff]: Sigma_fm (HFun_Sigma t)
proof –
  obtain x::name and y::name and z::name and x'::name and y'::name and z'::name
  where atoms: atom z # (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)

end

```

Chapter 5

Predicates for Terms, Formulas and Substitution

```
theory Coding_Predicates
imports Coding Sigma
begin
```

```
declare succ_iff [simp del]
```

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

5.1 Predicates for atomic terms

5.1.1 Free Variables

```
definition VarP :: tm  $\Rightarrow$  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 VarP_sf [iff]: Sigma_fm (VarP x)
by (auto simp: VarP_def)
```

```
lemma VarP_subst [simp]: (VarP x)(i::=t) = VarP (subst i t x)
by (simp add: VarP_def)
```

```
lemma VarP_cong: H  $\vdash$  x EQ x'  $\implies$  H  $\vdash$  VarP x IFF VarP x'
by (rule P1_cong) auto
```

```
lemma VarP_HPairE [intro!]: insert (VarP (HPair x y)) H  $\vdash$  A
by (auto simp: VarP_def)
```

5.1.2 De Bruijn Indexes

```
abbreviation Q_Ind :: tm  $\Rightarrow$  tm
where Q_Ind k  $\equiv$  HPair (HTuple 6) k
```

```
nominal_function IndP :: tm  $\Rightarrow$  fm
where atom m  $\#$  x  $\implies$ 
```

$IndP\ x = Ex\ m\ (OrdP\ (Var\ m)\ AND\ x\ EQ\ HPair\ (HTuple\ 6)\ (Var\ m))$
by (*auto simp: eqvt_def IndP_graph_aux_def flip_fresh_fresh*) (*metis obtain_fresh*)

nominal_termination (*eqvt*)

by *lexicographic_order*

lemma

shows $IndP_fresh_iff\ [simp]:\ a\ \# \ IndP\ x \longleftrightarrow a\ \# \ x$ (**is** *?thesis1*)
and $IndP_sf\ [iff]:\ \Sigma_{fm}\ (IndP\ x)$ (**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 ?thsf ?thqind*

by (*auto 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*

5.1.3 Various syntactic lemmas

5.2 The predicate $SeqCTermP$, for Terms and Constants

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 $SeqCTermP_sf\ [iff]:$

$\Sigma_{fm}\ (SeqCTermP\ vf\ s\ k\ t)$ (**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 } ⊢ 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)
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 SeqTermP :: tm ⇒ tm ⇒ tm ⇒ fm
  where SeqTermP ≡ SeqCTermP True

```

```

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

```

```

lemma SeqConstP_imp_SeqTermP: {SeqConstP s k t} ⊢ SeqTermP s k t

```

```

proof –
  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

```

nominal_function CTermP :: bool ⇒ tm ⇒ fm

```

where $\llbracket \text{atom } k \# (s,t); \text{atom } s \# t \rrbracket \implies$
 $C\text{TermP } vf \ t = \text{Ex } s \ (\text{Ex } k \ (\text{SeqCTermP } vf \ (\text{Var } s) \ (\text{Var } k) \ t))$
by (auto simp: eqvt_def CTermP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)

nominal_termination (eqvt)
by lexicographic_order

lemma
shows $C\text{TermP_fresh_iff}$ [simp]: $a \# C\text{TermP } vf \ t \longleftrightarrow a \# t$ (is ?thesis1)
and $C\text{TermP_sf}$ [iff]: $\text{Sigma_fm} \ (C\text{TermP } vf \ t)$ (is ?thsf)

proof –
obtain $k::\text{name}$ **and** $s::\text{name}$ **where** $\text{atom } k \# (s,t)$ $\text{atom } s \# t$
by (metis obtain_fresh)
thus ?thesis1 ?thsf
by auto
qed

lemma $C\text{TermP_subst}$ [simp]: $(C\text{TermP } vf \ i)(j::=w) = C\text{TermP } vf \ (\text{subst } j \ w \ i)$
proof –
obtain $k::\text{name}$ **and** $s::\text{name}$ **where** $\text{atom } k \# (s,i,j,w)$ $\text{atom } s \# (i,j,w)$
by (metis obtain_fresh)
thus ?thesis
by (simp add: CTermP.simps [of k s])
qed

abbreviation $\text{TermP} :: \text{tm} \Rightarrow \text{fm}$
where $\text{TermP} \equiv C\text{TermP } \text{True}$

abbreviation $\text{ConstP} :: \text{tm} \Rightarrow \text{fm}$
where $\text{ConstP} \equiv C\text{TermP } \text{False}$

5.3.2 Correctness properties for constants

lemma ConstP_imp_TermP : $\{\text{ConstP } t\} \vdash \text{TermP } t$
proof –
obtain $k::\text{name}$ **and** $s::\text{name}$ **where** $\text{atom } k \# (s,t)$ $\text{atom } s \# t$
by (metis obtain_fresh)
thus ?thesis
apply auto
apply (rule Ex_I [where $x = \text{Var } s$], simp)
apply (rule Ex_I [where $x = \text{Var } k$], auto intro: SeqConstP_imp_SeqTermP [THEN cut1])
done
qed

5.4 Abstraction over terms

nominal_function $\text{SeqStTermP} :: \text{tm} \Rightarrow \text{tm} \Rightarrow \text{tm} \Rightarrow \text{tm} \Rightarrow \text{tm} \Rightarrow \text{tm} \Rightarrow \text{fm}$
where $\llbracket \text{atom } l \# (s,k,v,i,sl,sl',m,n,sm,sm',sn,sn');$
 $\text{atom } sl \# (s,v,i,sl',m,n,sm,sm',sn,sn'); \text{atom } sl' \# (s,v,i,m,n,sm,sm',sn,sn');$
 $\text{atom } m \# (s,n,sm,sm',sn,sn'); \text{atom } n \# (s,sm,sm',sn,sn');$
 $\text{atom } sm \# (s,sm',sn,sn'); \text{atom } sm' \# (s,sn,sn');$
 $\text{atom } sn \# (s,sn'); \text{atom } sn' \# s \rrbracket \implies$
 $\text{SeqStTermP } v \ i \ t \ u \ s \ k =$
 $\text{VarP } v \ \text{AND } \text{LstSeqP } s \ k \ (\text{HPair } t \ u) \ \text{AND}$
 $\text{All2 } l \ (\text{SUCC } k) \ (\text{Ex } sl \ (\text{Ex } sl' \ (\text{HPair } (\text{Var } l) \ (\text{HPair } (\text{Var } sl) \ (\text{Var } sl')) \ \text{IN } s \ \text{AND}$
 $((\text{Var } sl \ \text{EQ } v \ \text{AND } \text{Var } sl' \ \text{EQ } i) \ \text{OR}$
 $((\text{IndP } (\text{Var } sl) \ \text{OR } \text{Var } sl \ \text{NEQ } v) \ \text{AND } \text{Var } sl' \ \text{EQ } \text{Var } sl)) \ \text{OR}$
 $\text{Ex } m \ (\text{Ex } n \ (\text{Ex } sm \ (\text{Ex } sm' \ (\text{Ex } sn \ (\text{Ex } sn' \ (\text{Var } m \ \text{IN } \text{Var } l \ \text{AND } \text{Var } n \ \text{IN } \text{Var } l \ \text{AND}$


```

      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 # 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 SeqStTermP_sf [iff]:
  Sigma_fm (SeqStTermP v i t u s k) (is ?thsf)
and SeqStTermP_imp_OrdP:
  { SeqStTermP v i t u s k }  $\vdash$  OrdP k (is ?thord)
and SeqStTermP_imp_VarP:
  { SeqStTermP v i t u s k }  $\vdash$  VarP v (is ?thvar)
and SeqStTermP_imp_LstSeqP:
  { SeqStTermP v i t u s k }  $\vdash$  LstSeqP s k (HPair t u) (is ?thlstseq)

```

proof –

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

where atoms:

```

atom l # (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)

```

by (metis obtain_fresh)

thus ?thesis1 ?thsf ?thord ?thvar ?thlstseq

by (auto intro: LstSeqP_OrdP)

qed

lemma SeqStTermP_subst [simp]:

```

(SeqStTermP v i t u s k)(j::=w) =
  SeqStTermP (subst j w v) (subst j w i) (subst j w t) (subst j w u) (subst j w s) (subst j w k)

```

proof –

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

where atom l # (s,k,v,i,w,j,sl,sl',m,n,sm,sm',sn,sn')

atom sl # (s,v,i,w,j,sl',m,n,sm,sm',sn,sn')

atom sl' # (s,v,i,w,j,m,n,sm,sm',sn,sn')

atom m # (s,w,j,n,sm,sm',sn,sn') atom n # (s,w,j,sm,sm',sn,sn')

atom sm # (s,w,j,sm',sn,sn') atom sm' # (s,w,j,sn,sn')

atom sn # (s,w,j,sn') atom sn' # (s,w,j)

by (metis obtain_fresh)

thus ?thesis

by (force simp add: SeqStTermP.simps [of l _ _ _ _ sl sl' m n sm sm' sn sn'])

qed

lemma SeqStTermP_cong:

```

[[H  $\vdash$  t EQ t'; H  $\vdash$  u EQ u'; H  $\vdash$  s EQ s'; H  $\vdash$  k EQ k']]

```

```

 $\implies$  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]

5.4.1 Defining the syntax: main predicate

nominal_function *AbstTermP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $\llbracket atom\ s \ \# \ (v,i,t,u,k); \ atom\ k \ \# \ (v,i,t,u) \rrbracket \Longrightarrow$
 $AbstTermP\ v\ i\ t\ u =$
 $OrdP\ i\ AND\ Ex\ s\ (Ex\ k\ (SeqStTermP\ v\ (Q_Ind\ i)\ t\ u\ (Var\ s)\ (Var\ k)))$
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*]:
 $a \ \# \ AbstTermP\ v\ i\ t\ u \longleftrightarrow a \ \# \ v \wedge a \ \# \ i \wedge a \ \# \ t \wedge a \ \# \ u$ (**is** *?thesis1*)
and *AbstTermP_sf* [*iff*]:
 $Sigma_fm\ (AbstTermP\ v\ i\ t\ u)$ (**is** *?thsf*)
and *AbstTermP_imp_VarP*:
 $\{ AbstTermP\ v\ i\ t\ u \} \vdash VarP\ v$ (**is** *?thvar*)
and *AbstTermP_imp_OrdP*:
 $\{ AbstTermP\ v\ i\ t\ u \} \vdash OrdP\ i$ (**is** *?thord*)

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* *?thsf* *?thvar* *?thord*
by (*auto intro: SeqStTermP_imp_VarP thin2*)

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*]

5.5 Substitution over terms

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 *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 } ⊢ 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 ?thsf ?thterm ?thvar
  by (auto intro: SeqStTermP_imp_VarP thin2)
qed

```

```

lemma SubstTermP_subst [simp]:
  (SubstTermP v i t u)(j::=w) = SubstTermP (subst j w v) (subst j w i) (subst j w t) (subst j w u)
proof -
  obtain s::name and k::name
  where atom s # (v,i,t,u,w,j,k) atom k # (v,i,t,u,w,j)
  by (metis obtain_fresh)
  thus ?thesis
  by (simp add: SubstTermP.simps [of s _ _ _ _ k])
qed

```

```

lemma SubstTermP_cong:
  [[H ⊢ v EQ v'; H ⊢ i EQ i'; H ⊢ t EQ t'; H ⊢ u EQ u']]
  ⇒ H ⊢ SubstTermP v i t u IFF SubstTermP v' i' t' u'
  by (rule P4_cong) auto

```

```

declare SubstTermP.simps [simp del]

```

5.6 Abstraction over formulas

5.6.1 The predicate *AbstAtomicP*

```

nominal_function AbstAtomicP :: tm ⇒ tm ⇒ tm ⇒ tm ⇒ fm
  where [[atom t # (v,i,y,y',t',u,u'); atom t' # (v,i,y,y',u,u');
        atom u # (v,i,y,y',u'); atom u' # (v,i,y,y')]] ⇒
  AbstAtomicP v i y y' =
  Ex t (Ex u (Ex t' (Ex u'
    (AbstTermP v i (Var t) (Var t') AND AbstTermP v i (Var u) (Var u') AND
      ((y EQ Q_Eq (Var t) (Var u) AND y' EQ Q_Eq (Var t') (Var u')) OR
      (y EQ Q_Mem (Var t) (Var u) AND y' EQ Q_Mem (Var t') (Var u'))))))))
  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 # AbstAtomicP v i y y' ⇔ a # v ∧ a # i ∧ a # y ∧ a # y' (is ?thesis1)
  and AbstAtomicP_sf [iff]: Sigma_fm (AbstAtomicP v i y y') (is ?thsf)

```

```

proof -
  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 ?thsf
  by auto
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*

```

nominal_function SeqAbstFormP :: tm ⇒ tm ⇒ tm ⇒ tm ⇒ tm ⇒ tm ⇒ fm
  where [[atom l # (s,k,v,sl,sl',m,n,smi,sm,sm',sni,sn,sn');
        atom sli # (s,v,sl,sl',m,n,smi,sm,sm',sni,sn,sn');
        atom sl # (s,v,sl',m,n,smi,sm,sm',sni,sn,sn');
        atom sl' # (s,v,m,n,smi,sm,sm',sni,sn,sn');
        atom m # (s,n,smi,sm,sm',sni,sn,sn');
        atom n # (s,smi,sm,sm',sni,sn,sn'); atom smi # (s,sm,sm',sni,sn,sn');
        atom sm # (s,sm',sni,sn,sn'); atom sm' # (s,sni,sn,sn');
        atom sni # (s,sn,sn'); atom sn # (s,sn'); atom sn' # (s)]] ⇒⇒
  SeqAbstFormP v i x x' s k =
    LstSeqP s k (HPair i (HPair x x')) AND
    All2 l (SUCC k) (Ex sli (Ex sl (Ex sl' (HPair (Var l) (HPair (Var sli) (HPair (Var sl) (Var sl'))))
  IN s AND
    (AbstAtomicP v (Var sli) (Var sl) (Var sl') OR
     OrdP (Var sli) AND
     Ex m (Ex n (Ex smi (Ex sm (Ex sm' (Ex sni (Ex sn (Ex sn'
       (Var m IN Var l AND Var n IN Var l AND
       HPair (Var m) (HPair (Var smi) (HPair (Var sm) (Var sm')))) IN s AND
       HPair (Var n) (HPair (Var sni) (HPair (Var sn) (Var sn')))) IN s AND
       ((Var sli EQ Var smi AND Var sli EQ Var sni AND
        Var sl EQ Q_Disj (Var sm) (Var sn) AND
        Var sl' EQ Q_Disj (Var sm') (Var sn')) OR
        (Var sli EQ Var smi AND
         Var sl EQ Q_Neg (Var sm) AND Var sl' EQ Q_Neg (Var sm')) OR
        (SUCC (Var sli) EQ Var smi AND
         Var sl EQ Q_Ex (Var sm) AND Var sl' EQ Q_Ex (Var sm'))))))))))))))))
  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 ⇔ a # v ∧ a # i ∧ a # x ∧ a # x' ∧ a # s ∧ a # k (is ?thesis1)
  and SeqAbstFormP_sf [iff]:
    Sigma_fm (SeqAbstFormP v i x x' s k) (is ?thsf)
  and SeqAbstFormP_imp_OrdP:
    { SeqAbstFormP v u x x' s k } ⊢ OrdP k (is ?thOrd)
  and SeqAbstFormP_imp_LstSeqP:
    { SeqAbstFormP v u x x' s k } ⊢ LstSeqP s k (HPair u (HPair x x')) (is ?thLstSeq)

```

```

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,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 ?thOrd ?thLstSeq
by (auto intro: LstSeqP_OrdP)
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,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

```

nominal_function AbstFormP :: tm ⇒ tm ⇒ tm ⇒ tm ⇒ fm
  where [[atom s # (v,i,x,x',k);
         atom k # (v,i,x,x')]] ⇒
  AbstFormP v i x x' = VarP v AND OrdP i AND Ex s (Ex k (SeqAbstFormP v i x x' (Var s) (Var k)))
  by (auto simp: eqvt_def AbstFormP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)

nominal_termination (eqvt)
  by lexicographic_order

lemma
  shows AbstFormP_fresh_iff [simp]:
    a # AbstFormP v i x x' ↔ a # v ∧ a # i ∧ a # x ∧ a # x' (is ?thesis1)
  and AbstFormP_sf [iff]:
    Sigma_fm (AbstFormP v i x x') (is ?thsf)
proof -
  obtain s::name and k::name where atom s # (v,i,x,x',k) atom k # (v,i,x,x')
  by (metis obtain_fresh)
  thus ?thesis1 ?thsf
  by auto
qed

```

```

lemma AbstFormP_subst [simp]:
  (AbstFormP v i x x')(j::=t) = AbstFormP (subst j t v) (subst j t i) (subst j t x) (subst j t x')
proof –
  obtain s::name and k::name where atom s # (v,i,x,x',t,j,k) 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]

```

5.7 Substitution over formulas

5.7.1 The predicate *SubstAtomicP*

```

nominal_function SubstAtomicP :: tm ⇒ tm ⇒ tm ⇒ tm ⇒ fm
where [atom t # (v,tm,y,y',t',u,u');
  atom t' # (v,tm,y,y',u,u');
  atom u # (v,tm,y,y',u');
  atom u' # (v,tm,y,y')] ⇒⇒
  SubstAtomicP v tm y y' =
  Ex t (Ex u (Ex t' (Ex u'
    (SubstTermP v tm (Var t) (Var t') AND SubstTermP v tm (Var u) (Var u') AND
    ((y EQ Q_Eq (Var t) (Var u) AND y' EQ Q_Eq (Var t') (Var u')) OR
    (y EQ Q_Mem (Var t) (Var u) AND y' EQ Q_Mem (Var t') (Var u')))))))
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' ↔ a # v ∧ a # tm ∧ a # y ∧ a # y' (is ?thesis1)
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 ?thsf
by auto
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:
  [H ⊢ v EQ v'; H ⊢ tm EQ tm'; H ⊢ x EQ x'; H ⊢ y EQ y']

```

$\Rightarrow H \vdash \text{SubstAtomicP } v \text{ tm } x \text{ y}$ IFF $\text{SubstAtomicP } v' \text{ tm}' x' y'$
 by (rule $P4_cong$) auto

5.7.2 The predicate SubstMakeForm

nominal_function $\text{SeqSubstFormP} :: \text{tm} \Rightarrow \text{tm} \Rightarrow \text{tm} \Rightarrow \text{tm} \Rightarrow \text{tm} \Rightarrow \text{tm} \Rightarrow \text{tm} \Rightarrow \text{fm}$
where $\llbracket \text{atom } l \# (s, k, v, u, sl, sl', m, n, sm, sm', sn, sn') \rrbracket$;
 $\text{atom } sl \# (s, v, u, sl', m, n, sm, sm', sn, sn')$;
 $\text{atom } sl' \# (s, v, u, m, n, sm, sm', sn, sn')$;
 $\text{atom } m \# (s, n, sm, sm', sn, sn')$; $\text{atom } n \# (s, sm, sm', sn, sn')$;
 $\text{atom } sm \# (s, sm', sn, sn')$; $\text{atom } sm' \# (s, sn, sn')$;
 $\text{atom } sn \# (s, sn')$; $\text{atom } sn' \# s \rrbracket \Rightarrow$
 $\text{SeqSubstFormP } v \text{ u } x \text{ x}' \text{ s } k =$
 $\text{LstSeqP } s \text{ k } (\text{HPair } x \text{ x}') \text{ AND}$
 $\text{All2 } l (\text{SUCC } k) (\text{Ex } sl (\text{Ex } sl' (\text{HPair } (\text{Var } l) (\text{HPair } (\text{Var } sl) (\text{Var } sl')) \text{ IN } s \text{ AND}$
 $(\text{SubstAtomicP } v \text{ u } (\text{Var } sl) (\text{Var } sl')) \text{ OR}$
 $\text{Ex } m (\text{Ex } n (\text{Ex } sm (\text{Ex } sm' (\text{Ex } sn (\text{Ex } sn' (\text{Var } m \text{ IN } \text{Var } l \text{ AND } \text{Var } n \text{ IN } \text{Var } l \text{ AND}$
 $\text{HPair } (\text{Var } m) (\text{HPair } (\text{Var } sm) (\text{Var } sm')) \text{ IN } s \text{ AND}$
 $\text{HPair } (\text{Var } n) (\text{HPair } (\text{Var } sn) (\text{Var } sn')) \text{ IN } s \text{ AND}$
 $((\text{Var } sl \text{ EQ } Q_Disj (\text{Var } sm) (\text{Var } sn)) \text{ AND}$
 $\text{Var } sl' \text{ EQ } Q_Disj (\text{Var } sm') (\text{Var } sn')) \text{ OR}$
 $(\text{Var } sl \text{ EQ } Q_Neg (\text{Var } sm) \text{ AND } \text{Var } sl' \text{ EQ } Q_Neg (\text{Var } sm')) \text{ OR}$
 $(\text{Var } sl \text{ EQ } Q_Ex (\text{Var } sm) \text{ AND } \text{Var } sl' \text{ EQ } Q_Ex (\text{Var } sm'))))))))))))$
apply (simp_all add: $\text{eqvt_def SeqSubstFormP_graph_aux_def flip_fresh_fresh}$)
by auto ($\text{metis obtain_fresh}$)

nominal_termination (eqvt)
 by $\text{lexicographic_order}$

lemma

shows $\text{SeqSubstFormP_fresh_iff}$ [simp]:
 $a \# \text{SeqSubstFormP } v \text{ u } x \text{ x}' \text{ s } k \longleftrightarrow a \# v \wedge a \# u \wedge a \# x \wedge a \# x' \wedge a \# s \wedge a \# k$ (**is** $?thesis1$)
and SeqSubstFormP_sf [iff]:
 $\text{Sigma_fm } (\text{SeqSubstFormP } v \text{ u } x \text{ x}' \text{ s } k)$ (**is** $?thsf$)
and $\text{SeqSubstFormP_imp_OrdP}$:
 $\{ \text{SeqSubstFormP } v \text{ u } x \text{ x}' \text{ s } k \} \vdash \text{OrdP } k$ (**is** $?thOrd$)
and $\text{SeqSubstFormP_imp_LstSeqP}$:
 $\{ \text{SeqSubstFormP } v \text{ u } x \text{ x}' \text{ s } k \} \vdash \text{LstSeqP } s \text{ k } (\text{HPair } x \text{ x}')$ (**is** $?thLstSeq$)

proof –

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 atoms :

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

by ($\text{metis obtain_fresh}$)

thus $?thesis1$ $?thsf$ $?thOrd$ $?thLstSeq$

by ($\text{auto intro: LstSeqP_OrdP}$)

qed

lemma $\text{SeqSubstFormP_subst}$ [simp]:

$(\text{SeqSubstFormP } v \text{ u } x \text{ x}' \text{ s } k)(i::t) =$
 $\text{SeqSubstFormP } (\text{subst } i \text{ t } v) (\text{subst } i \text{ t } u) (\text{subst } i \text{ t } x) (\text{subst } i \text{ t } x') (\text{subst } i \text{ t } s) (\text{subst } i \text{ t } k)$

proof –

obtain $l::\text{name}$ **and** $sl::\text{name}$ **and** $sl'::\text{name}$ **and** $m::\text{name}$ **and** $n::\text{name}$ **and**

```

    sm::name and sm'::name and sn::name and sn'::name
  where atom l # (s,k,v,u,t,i,sl,sl',m,n,sm,sm',sn,sn')
        atom sl # (s,v,u,t,i,sl',m,n,sm,sm',sn,sn')
        atom sl' # (s,v,u,t,i,m,n,sm,sm',sn,sn')
        atom m # (s,t,i,n,sm,sm',sn,sn') atom n # (s,t,i,sm,sm',sn,sn')
        atom sm # (s,t,i,sm',sn,sn') atom sm' # (s,t,i,sn,sn')
        atom sn # (s,t,i,sn) atom sn' # (s,t,i)
  by (metis obtain_fresh)
  thus ?thesis
  by (force simp add: SeqSubstFormP.simps [of l _ _ _ _ sl sl' m n sm sm' sn sn'])
qed

```

```

lemma SeqSubstFormP_cong:
  [[H ⊢ t EQ t'; H ⊢ u EQ u'; H ⊢ s EQ s'; H ⊢ k EQ k']
  ⇒ H ⊢ SeqSubstFormP v i t u s k IFF SeqSubstFormP v i t' u' s' k'
  by (rule P4_cong [where tms=[v,i]]) (auto simp: fresh_Cons)

```

```

declare SeqSubstFormP.simps [simp del]

```

5.7.3 Defining the syntax: the main SubstForm predicate

```

nominal_function SubstFormP :: tm ⇒ tm ⇒ tm ⇒ tm ⇒ fm
  where [[atom s # (v,i,x,x',k); atom k # (v,i,x,x')] ⇒
        SubstFormP v i x x' =
        VarP v AND TermP i AND Ex s (Ex k (SeqSubstFormP v i x x' (Var s) (Var k)))
  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 # SubstFormP v i x x' ⇔ a # v ∧ a # i ∧ a # x ∧ a # x' (is ?thesis1)
  and SubstFormP_sf [iff]:
    Sigma_fm (SubstFormP v i x x') (is ?thsf)

```

```

proof -
  obtain s::name and k::name
  where atom s # (v,i,x,x',k) atom k # (v,i,x,x')
  by (metis obtain_fresh)
  thus ?thesis1 ?thsf
  by auto
qed

```

```

lemma SubstFormP_subst [simp]:
  (SubstFormP v i x x')(j::=t) = SubstFormP (subst j t v) (subst j t i) (subst j t x) (subst j t x')
proof -
  obtain s::name and k::name where atom s # (v,i,x,x',t,j,k) 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:
  [[H ⊢ v EQ v'; H ⊢ i EQ i'; H ⊢ t EQ t'; H ⊢ u EQ u']
  ⇒ H ⊢ SubstFormP v i t u IFF SubstFormP v' i' t' u'
  by (rule P4_cong) auto

```

```

lemma ground_SubstFormP [simp]: ground_fm (SubstFormP v y x x') ⇔ ground v ∧ ground y ∧ ground

```


$x \wedge \text{ground } x'$
by (*auto simp: ground_aux_def ground_fm_aux_def supp_conv_fresh*)

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

5.8 The predicate *AtomicP*

nominal_function *AtomicP* :: $tm \Rightarrow fm$
where $\llbracket \text{atom } t \# (u,y); \text{atom } u \# y \rrbracket \Longrightarrow$
 $\text{AtomicP } y = \text{Ex } t (\text{Ex } u (\text{TermP } (\text{Var } t) \text{ AND } \text{TermP } (\text{Var } u) \text{ AND}$
 $(y \text{ EQ } Q_Eq (\text{Var } t) (\text{Var } u) \text{ OR}$
 $y \text{ EQ } Q_Mem (\text{Var } t) (\text{Var } u))))$
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*]: $a \# \text{AtomicP } y \longleftrightarrow a \# y$ (**is** *?thesis1*)
and *AtomicP_sf* [*iff*]: $\text{Sigma_fm } (\text{AtomicP } y)$ (**is** *?thsf*)

proof –
obtain *t::name* **and** *u::name* **where** $\text{atom } t \# (u,y)$ $\text{atom } u \# y$
by (*metis obtain_fresh*)
thus *?thesis1* *?thsf*
by *auto*

qed

lemma *AtomicP_subst* [*simp*]: $(\text{AtomicP } t)(j::=w) = \text{AtomicP } (\text{subst } j \ w \ t)$

proof –
obtain $x \ y :: \text{name}$ **where** $\text{atom } x \# (j,w,t,y)$ $\text{atom } y \# (j,w,t)$
by (*metis obtain_fresh*)
thus *?thesis*
by (*auto simp: AtomicP.simps* [*of x y*])

qed

5.9 The predicate *MakeForm*

nominal_function *MakeFormP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $\llbracket \text{atom } v \# (y,u,w,au); \text{atom } au \# (y,u,w) \rrbracket \Longrightarrow$
 $\text{MakeFormP } y \ u \ w =$
 $y \text{ EQ } Q_Disj \ u \ w \ \text{OR } y \text{ EQ } Q_Neg \ u \ \text{OR}$
 $\text{Ex } v (\text{Ex } au (\text{AbstFormP } (\text{Var } v) \ \text{Zero } u \ (\text{Var } au) \ \text{AND } y \text{ EQ } Q_Ex \ (\text{Var } au)))$
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*]:
 $a \# \text{MakeFormP } y \ u \ w \longleftrightarrow a \# y \wedge a \# u \wedge a \# w$ (**is** *?thesis1*)
and *MakeFormP_sf* [*iff*]:
 $\text{Sigma_fm } (\text{MakeFormP } y \ u \ w)$ (**is** *?thsf*)

proof –
obtain *v::name* **and** *au::name* **where** $\text{atom } v \# (y,u,w,au)$ $\text{atom } au \# (y,u,w)$
by (*metis obtain_fresh*)
thus *?thesis1* *?thsf*
by *auto*

qed

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

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

proof –

obtain *a b* :: *name* where *atom a* # (*j,w,y,u,t,b*) *atom b* # (*j,w,y,u,t*)

by (*metis obtain_fresh*)

thus ?*thesis*

by (*auto simp: MakeFormP.simps* [*of a _ _ _ b*])

qed

5.10 The predicate *SeqFormP*

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*) \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 # *SeqFormP s k t* \longleftrightarrow *a* # *s* \wedge *a* # *k* \wedge *a* # *t* (*is ?thesis1*)

and *SeqFormP_sf* [*iff*]: *Sigma_fm* (*SeqFormP s k t*) (*is ?thsf*)

and *SeqFormP_imp_OrdP*:

{ *SeqFormP s k t* } \vdash *OrdP k* (*is ?thOrd*)

and *SeqFormP_imp_LstSeqP*:

{ *SeqFormP s k t* } \vdash *LstSeqP s k t* (*is ?thLstSeq*)

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* ?*thOrd* ?*thLstSeq*

by (*auto intro: LstSeqP_OrdP*)

qed

lemma *SeqFormP_subst* [*simp*]:

(*SeqFormP s k t*)(*j::=w*) = *SeqFormP* (*subst j w s*) (*subst j w k*) (*subst j w t*)

proof –

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

where *atom l* # (*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

```
nominal_function FormP :: tm  $\Rightarrow$  fm
where  $\llbracket atom\ k\ \#\ (s,y); atom\ s\ \#\ y \rrbracket \Longrightarrow$ 
  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 FormP_sf [iff]: Sigma_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 ?thsf
by auto
```

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

```
nominal_function VarNonOccFormP :: tm  $\Rightarrow$  tm  $\Rightarrow$  fm
where VarNonOccFormP v x = FormP x AND SubstFormP v Zero x x
by (auto simp: eqvt_def VarNonOccFormP_graph_aux_def)
```

```
nominal_termination (eqvt)
by lexicographic_order
```

lemma

```
shows VarNonOccFormP_fresh_iff [simp]: a  $\#\ VarNonOccFormP\ v\ y \longleftrightarrow a\ \#\ v \wedge a\ \#\ y$  (is ?thesis1)
and VarNonOccFormP_sf [iff]: Sigma_fm (VarNonOccFormP v y) (is ?thsf)
```

proof –

```
show ?thesis1 ?thsf
by auto
```

qed

```
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

```
nominal_function SentP :: tm ⇒ fm
where [|atom y # (z,w,x); atom z # (w,x); atom w # x|] ⇒
  SentP x = Ex y (Ex z (Ex w (FormP (Var y) AND FormP (Var z) AND FormP (Var w) AND
    ( (x EQ Q_Imp (Var y) (Var y)) OR
      (x EQ Q_Imp (Var y) (Q_Disj (Var y) (Var z)) OR
        (x EQ Q_Imp (Q_Disj (Var y) (Var y)) (Var y)) OR
          (x EQ Q_Imp (Q_Disj (Var y) (Q_Disj (Var z) (Var w)))
            (Q_Disj (Q_Disj (Var y) (Var z)) (Var w)) OR
              (x EQ Q_Imp (Q_Disj (Var y) (Var z))
                (Q_Imp (Q_Disj (Q_Neg (Var y)) (Var w)) (Q_Disj (Var z) (Var w))))))))))
  by (auto simp: eqt_def SentP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)

nominal_termination (eqt)
by lexicographic_order
```

lemma

```
shows SentP_fresh_iff [simp]: a # SentP x ↔ a # x (is ?thesis1)
and SentP_sf [iff]: Sigma_fm (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 ?thsf
by auto
```

qed

6.1.2 The predicate *Equality_axP*, for the Equality Axioms

```
function Equality_axP :: tm ⇒ fm
where Equality_axP x =
  x EQ «refl_ax» OR x EQ «eq_cong_ax» OR x EQ «mem_cong_ax» OR x EQ «eats_cong_ax»
by auto
```

termination

```
by lexicographic_order
```

6.1.3 The predicate HF_axP , for the HF Axioms

function $HF_axP :: tm \Rightarrow fm$
where $HF_axP\ x = x\ EQ\ \langle HF1 \rangle\ OR\ x\ EQ\ \langle HF2 \rangle$
by *auto*

termination
by *lexicographic_order*

lemma $HF_axP_sf\ [iff]:\ Sigma_fm\ (HF_axP\ t)$
by *auto*

6.1.4 The specialisation axioms

Defining the syntax

nominal_function $Special_axP :: tm \Rightarrow fm$ **where**
 $\llbracket atom\ v\ \# (p, sx, y, ax, x); atom\ x\ \# (p, sx, y, ax);$
 $atom\ ax\ \# (p, sx, y); atom\ y\ \# (p, sx); atom\ sx\ \# p \rrbracket \implies$
 $Special_axP\ p = Ex\ v\ (Ex\ x\ (Ex\ ax\ (Ex\ y\ (Ex\ sx$
 $(FormP\ (Var\ x)\ AND\ VarP\ (Var\ v)\ AND\ TermP\ (Var\ y)\ AND$
 $AbstFormP\ (Var\ v)\ Zero\ (Var\ x)\ (Var\ ax)\ AND$
 $SubstFormP\ (Var\ v)\ (Var\ y)\ (Var\ x)\ (Var\ sx)\ AND$
 $p\ EQ\ Q_Imp\ (Var\ sx)\ (Q_Ex\ (Var\ ax))))))$
by (*auto simp: eqvt_def Special_axP_graph_aux_def flip_fresh_fresh*) (*metis obtain_fresh*)

nominal_termination (*eqvt*)
by *lexicographic_order*

lemma
shows $Special_axP_fresh_iff\ [simp]:\ a\ \# Special_axP\ p \longleftrightarrow a\ \# p$ (*is ?thesis1*)
and $Special_axP_sf\ [iff]:\ Sigma_fm\ (Special_axP\ p)$ (*is ?thesis3*)
proof –
obtain *v::name and x::name and ax::name and y::name and sx::name*
where $atom\ v\ \# (p, sx, y, ax, x)\ atom\ x\ \# (p, sx, y, ax)$
 $atom\ ax\ \# (p, sx, y)\ atom\ y\ \# (p, sx)\ atom\ sx\ \# p$
by (*metis obtain_fresh*)
thus *?thesis1 ?thesis3*
by *auto*
qed

6.1.5 The induction axioms

Defining the syntax

nominal_function $Induction_axP :: tm \Rightarrow fm$ **where**
 $\llbracket atom\ ax\ \# (p, v, w, x, x0, xv, xevw, allw, allvw);$
 $atom\ allvw\ \# (p, v, w, x, x0, xv, xevw, allw); atom\ allw\ \# (p, v, w, x, x0, xv, xevw);$
 $atom\ xevw\ \# (p, v, w, x, x0, xv); atom\ xv\ \# (p, v, w, x, x0);$
 $atom\ x0\ \# (p, v, w, x); atom\ x\ \# (p, v, w);$
 $atom\ w\ \# (p, v); atom\ v\ \# p \rrbracket \implies$
 $Induction_axP\ p = Ex\ v\ (Ex\ w\ (Ex\ x\ (Ex\ x0\ (Ex\ xv\ (Ex\ xevw\ (Ex\ allw\ (Ex\ allvw\ (Ex\ ax$
 $((Var\ v\ NEQ\ Var\ w)\ AND\ VarNonOccFormP\ (Var\ w)\ (Var\ x)\ AND$
 $SubstFormP\ (Var\ v)\ Zero\ (Var\ x)\ (Var\ x0)\ AND$
 $SubstFormP\ (Var\ v)\ (Var\ w)\ (Var\ x)\ (Var\ xv)\ AND$
 $SubstFormP\ (Var\ v)\ (Q_Eats\ (Var\ v)\ (Var\ w))\ (Var\ x)\ (Var\ xevw)\ AND$
 $AbstFormP\ (Var\ w)\ Zero\ (Q_Imp\ (Var\ x)\ (Q_Imp\ (Var\ xv)\ (Var\ xevw)))\ (Var\ allw)\ AND$
 $AbstFormP\ (Var\ v)\ Zero\ (Q_All\ (Var\ allw))\ (Var\ allvw)\ AND$
 $AbstFormP\ (Var\ v)\ Zero\ (Var\ x)\ (Var\ ax)\ AND$
 $p\ EQ\ Q_Imp\ (Var\ x0)\ (Q_Imp\ (Q_All\ (Var\ allvw))\ (Q_All\ (Var\ ax))))))))))$

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]: $a \# \text{Induction_axP } p \longleftrightarrow a \# p$ (is ?thesis1)

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

qed

6.1.6 The predicate *AxiomP*, for any Axioms

definition *AxiomP* :: *tm* \Rightarrow *fm*

where *AxiomP* *x* $\equiv x \text{ EQ } \langle\langle \text{extra_axiom} \rangle\rangle \text{ OR SentP } x \text{ OR Equality_axP } x \text{ OR}$
HF_axP *x* *OR Special_axP* *x* *OR Induction_axP* *x*

lemma *AxiomP_I*:

$\{\} \vdash \text{AxiomP } \langle\langle \text{extra_axiom} \rangle\rangle$
 $\{\} \vdash \text{SentP } x \implies \{\} \vdash \text{AxiomP } x$
 $\{\} \vdash \text{Equality_axP } x \implies \{\} \vdash \text{AxiomP } x$
 $\{\} \vdash \text{HF_axP } x \implies \{\} \vdash \text{AxiomP } x$
 $\{\} \vdash \text{Special_axP } x \implies \{\} \vdash \text{AxiomP } x$
 $\{\} \vdash \text{Induction_axP } x \implies \{\} \vdash \text{AxiomP } x$

unfolding *AxiomP_def*

by (rule *Disj_I1*, rule *Refl*,
rule *Disj_I2*, rule *Disj_I1*, *assumption*,
rule *Disj_I2*, rule *Disj_I2*, rule *Disj_I1*, *assumption*,
rule *Disj_I2*, rule *Disj_I2*, rule *Disj_I2*, rule *Disj_I1*, *assumption*,
rule *Disj_I2*, rule *Disj_I2*, rule *Disj_I2*, rule *Disj_I2*, rule *Disj_I1*, *assumption*,
rule *Disj_I2*, rule *Disj_I2*, rule *Disj_I2*, rule *Disj_I2*, rule *Disj_I2*, *assumption*)

lemma *AxiomP_eqvt* [eqvt]: $(p \cdot \text{AxiomP } x) = \text{AxiomP } (p \cdot x)$

by (simp add: *AxiomP_def*)

lemma *AxiomP_fresh_iff* [simp]: $a \# \text{AxiomP } x \longleftrightarrow a \# x$

by (auto simp: *AxiomP_def*)

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 *ModPonP* :: *tm* \Rightarrow *tm* \Rightarrow *tm* \Rightarrow *fm*

where *ModPonP* *x y z* = $(y \text{ EQ } Q_Imp \ x \ z)$

lemma *ModPonP_eqvt* [eqvt]: $(p \cdot \text{ModPonP } x \ y \ z) = \text{ModPonP } (p \cdot x) \ (p \cdot y) \ (p \cdot z)$

by (simp add: *ModPonP_def*)

lemma *ModPonP_fresh_iff* [simp]: $a \# \text{ModPonP } x \ y \ z \longleftrightarrow a \# x \wedge a \# y \wedge a \# z$

by (auto simp: ModPonP_def)

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

nominal_function *ExistsP* :: *tm* \Rightarrow *tm* \Rightarrow *fm* where
 \llbracket atom $x \# (p, q, v, y, x')$; atom $x' \# (p, q, v, y)$;
 atom $y \# (p, q, v)$; atom $v \# (p, q)$ $\rrbracket \Longrightarrow$
 $ExistsP\ p\ q = Ex\ x\ (Ex\ x'\ (Ex\ y\ (Ex\ v\ (FormP\ (Var\ x)\ AND$
 $VarNonOccFormP\ (Var\ v)\ (Var\ y)\ AND$
 $AbstFormP\ (Var\ v)\ Zero\ (Var\ x)\ (Var\ x')\ AND$
 $p\ EQ\ Q_Imp\ (Var\ x)\ (Var\ y)\ AND$
 $q\ EQ\ Q_Imp\ (Q_Ex\ (Var\ x'))\ (Var\ y))))))$
 by (auto simp: eqvt_def ExistsP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)

nominal_termination (eqvt)
by lexicographic_order

lemma
shows *ExistsP_fresh_iff* [simp]: $a \# ExistsP\ p\ q \longleftrightarrow a \# p \wedge a \# q$ (is ?thesis1)
and *ExistsP_sf* [iff]: *Sigma_fm* (*ExistsP* p q) (is ?thesis3)

proof –

obtain $x::name$ **and** $x'::name$ **and** $y::name$ **and** $v::name$
where atom $x \# (p, q, v, y, x')$ atom $x' \# (p, q, v, y)$ atom $y \# (p, q, v)$ atom $v \# (p, q)$
by (metis obtain_fresh)
thus ?thesis1 ?thesis3
by auto

qed

lemma *ExistsP_subst* [simp]: (*ExistsP* p q)($j::=w$) = *ExistsP* (subst j w p) (subst j w q)

proof –

obtain $x::name$ **and** $x'::name$ **and** $y::name$ **and** $v::name$
where atom $x \# (j, w, p, q, v, y, x')$ atom $x' \# (j, w, p, q, v, y)$
 atom $y \# (j, w, p, q, v)$ atom $v \# (j, w, p, q)$
by (metis obtain_fresh)
thus ?thesis
by (auto simp: ExistsP_simps [of x _ _ x' y v])

qed

6.1.9 The predicate *SubstP*, for the substitution rule

Although the substitution rule is derivable in the calculus, the derivation is too complicated to reproduce within the proof function. It is much easier to provide it as an immediate inference step, justifying its soundness in terms of other inference rules.

Definition

nominal_function *SubstP* :: *tm* \Rightarrow *tm* \Rightarrow *fm* where
 \llbracket atom $u \# (p, q, v)$; atom $v \# (p, q)$ $\rrbracket \Longrightarrow$
 $SubstP\ p\ q = Ex\ v\ (Ex\ u\ (SubstFormP\ (Var\ v)\ (Var\ u)\ p\ q))$

by (auto simp: eqvt_def SubstP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)

nominal_termination (eqvt)
by lexicographic_order

lemma

shows $SubstP_fresh_iff$ [simp]: $a \# SubstP\ p\ q \longleftrightarrow a \# p \wedge a \# q$ (is ?thesis1)
and $SubstP_sf$ [iff]: $Sigma_fm\ (SubstP\ p\ q)$ (is ?thesis3)

proof –

obtain $u::name$ and $v::name$ where $atom\ u \# (p,q,v)$ $atom\ v \# (p,q)$

by (metis obtain_fresh)

thus ?thesis1 ?thesis3

by auto

qed

lemma $SubstP_subst$ [simp]: $(SubstP\ p\ q)(j::=w) = SubstP\ (subst\ j\ w\ p)\ (subst\ j\ w\ q)$

proof –

obtain $u::name$ and $v::name$ where $atom\ u \# (j,w,p,q,v)$ $atom\ v \# (j,w,p,q)$

by (metis obtain_fresh)

thus ?thesis

by (simp add: SubstP.simps [of u _ _ v])

qed

6.1.10 The predicate $PrfP$

nominal_function $PrfP$:: $tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$

where $\llbracket atom\ l \# (s,sl,m,n,sm,sn); atom\ sl \# (s,m,n,sm,sn);$

$atom\ m \# (s,n,sm,sn); atom\ n \# (s,k,sm,sn);$

$atom\ sm \# (s,sn); atom\ sn \# (s) \rrbracket \Longrightarrow$

$PrfP\ s\ k\ t =$

$LstSeqP\ s\ k\ t\ AND$

$All2\ n\ (SUCC\ k)\ (Ex\ sn\ (HPair\ (Var\ n)\ (Var\ sn)\ IN\ s\ AND\ (AxiomP\ (Var\ sn)\ OR$

$Ex\ m\ (Ex\ l\ (Ex\ sm\ (Ex\ sl\ (Var\ m\ IN\ Var\ n\ AND\ Var\ l\ IN\ Var\ n\ AND$

$HPair\ (Var\ m)\ (Var\ sm)\ IN\ s\ AND\ HPair\ (Var\ l)\ (Var\ sl)\ IN\ s\ AND$

$(ModPonP\ (Var\ sm)\ (Var\ sl)\ (Var\ sn)\ OR$

$ExistsP\ (Var\ sm)\ (Var\ sn)\ OR$

$SubstP\ (Var\ sm)\ (Var\ sn))))))$

by (auto simp: eqvt_def PrfP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)

nominal_termination (eqvt)
by lexicographic_order

lemma

shows $PrfP_fresh_iff$ [simp]: $a \# PrfP\ s\ k\ t \longleftrightarrow a \# s \wedge a \# k \wedge a \# t$ (is ?thesis1)

and $PrfP_imp_OrdP$ [simp]: $\{PrfP\ s\ k\ t\} \vdash OrdP\ k$ (is ?thord)

and $PrfP_imp_LstSeqP$ [simp]: $\{PrfP\ s\ k\ t\} \vdash LstSeqP\ s\ k\ t$ (is ?thlstseq)

and $PrfP_sf$ [iff]: $Sigma_fm\ (PrfP\ s\ k\ t)$ (is ?thsf)

proof –

obtain $l::name$ and $sl::name$ and $m::name$ and $n::name$ and $sm::name$ and $sn::name$

where $atoms: atom\ l \# (s,sl,m,n,sm,sn)$ $atom\ sl \# (s,m,n,sm,sn)$

$atom\ m \# (s,n,sm,sn)$ $atom\ n \# (s,k,sm,sn)$

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

by (metis obtain_fresh)

thus ?thesis1 ?thord ?thlstseq ?thsf

by (auto intro: LstSeqP_OrdP)

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

```

nominal_function PfP :: tm  $\Rightarrow$  fm
where  $\llbracket$ atom k # (s,y); 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 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 ?thsf
  by auto
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)

```

end

Chapter 7

Syntactic Preliminaries for the Second Incompleteness Theorem

```
theory II_Prelims
imports Pf_Predicates
begin

declare IndP.simps [simp del]

lemma OrdP_ORD_OF [intro]:  $H \vdash \text{OrdP } (\text{ORD\_OF } n)$ 
proof -
  have {}  $\vdash \text{OrdP } (\text{ORD\_OF } n)$ 
  by (induct n) (auto simp: OrdP_SUCC_I)
  thus ?thesis
  by (rule thin0)
qed

lemma VarP_Var [intro]:  $H \vdash \text{VarP } \langle \text{Var } i \rangle$ 
  unfolding VarP_def
  by (auto simp: quot_Var OrdP_ORD_OF intro!: OrdP_SUCC_I cut1[OF Zero_In_SUCC])

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 Mem_HFun_Sigma_OrdP:  $\{\text{HPair } t \ u \ \text{IN } f, \text{HFun\_Sigma } f\} \vdash \text{OrdP } t$ 
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  $\text{atom } z \# (f, t, u, z', x, y, x', y')$ 
  and  $\text{atom } x \# (f, t, u, y, x', y')$ 
  and  $\text{atom } y \# (f, t, u, x', y')$ 
  and  $\text{atom } x' \# (f, t, u, y')$ 
  and  $\text{atom } y' \# (f, t, u)$ 
  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  $x = \text{HPair } t \ u$ , THEN rotate2], auto)

```

```

  apply (rule All2_E [where x=HPair t u], auto intro: OrdP_cong [THEN Iff_MP2_same])
done
qed

```

7.1 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  $\vdash$  NotInDom t Zero
proof -
  obtain z::name where atom z  $\#$  t
  by (metis obtain_fresh)
  hence {}  $\vdash$  NotInDom t Zero
  by (auto simp: fresh_Pair)
  thus ?thesis
  by (rule thin0)
qed

```

```

lemma NotInDom_Fls: {HPair d d' IN r, NotInDom d r}  $\vdash$  A
proof -
  obtain z::name where atom z  $\#$  (d,r)
  by (metis obtain_fresh)
  hence {HPair d d' IN r, NotInDom d r}  $\vdash$  Fls
  by (auto intro!: Ex_I [where x=d'])
  thus ?thesis
  by (metis ExFalso)
qed

```

```

lemma NotInDom_Contra: H  $\vdash$  NotInDom d r  $\Longrightarrow$  H  $\vdash$  HPair x y IN r  $\Longrightarrow$  insert (x EQ d) H  $\vdash$  A
by (rule NotInDom_Fls [THEN cut2, THEN ExFalso])
  (auto intro: thin1 NotInDom_cong [OF Assume Refl, THEN Iff_MP2_same])

```

7.2 Restriction of a Sequence to a Domain

nominal_function *RestrictedP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $\llbracket atom\ x \# (y,f,k,g); atom\ y \# (f,k,g) \rrbracket \implies$
 $RestrictedP\ f\ k\ g =$
 $g\ SUBS\ f\ AND$
 $All\ x\ (All\ y\ (HPair\ (Var\ x)\ (Var\ y)\ IN\ g\ IFF$
 $(Var\ x)\ IN\ k\ AND\ HPair\ (Var\ x)\ (Var\ y)\ IN\ f))$
by (*auto simp: eqvt_def RestrictedP_graph_aux_def flip_fresh_fresh*) (*metis obtain_fresh*)

nominal_termination (*eqvt*)
by *lexicographic_order*

lemma *RestrictedP_fresh_iff* [*simp*]: $a \# RestrictedP\ f\ k\ g \longleftrightarrow a \# f \wedge a \# k \wedge a \# g$
proof –
obtain $x::name$ **and** $y::name$ **where** $atom\ x \# (y,f,k,g)$ $atom\ y \# (f,k,g)$
by (*metis obtain_fresh*)
thus *?thesis*
by *auto*
qed

lemma *subst_fm_RestrictedP* [*simp*]:
 $(RestrictedP\ f\ k\ g)(i::=u) = RestrictedP\ (subst\ i\ u\ f)\ (subst\ i\ u\ k)\ (subst\ i\ u\ g)$
proof –
obtain $x::name$ **and** $y::name$ **where** $atom\ x \# (y,f,k,g,i,u)$ $atom\ y \# (f,k,g,i,u)$
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  $\vdash$  RestrictedP s k t  $\implies$  H  $\vdash$  a IN t  $\implies$  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  $\vdash$  Ex s (RestrictedP f k (Var s))

```

```

lemma cut_RestrictedP:
  assumes s: atom s # (f,k,A) and  $\forall C \in H. atom s \# C$ 
  shows insert (RestrictedP f k (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]

```

7.3 Applications to LstSeqP

```

lemma HFun_Sigma_Eats:
  assumes H  $\vdash$  HFun_Sigma r H  $\vdash$  NotInDom d r H  $\vdash$  OrdP d
  shows H  $\vdash$  HFun_Sigma (Eats r (HPair d d'))

```

```

lemma HFun_Sigma_single [iff]: H  $\vdash$  OrdP d  $\implies$  H  $\vdash$  HFun_Sigma (Eats Zero (HPair d d'))
  by (metis HFun_Sigma_Eats HFun_Sigma_Zero NotInDom_Zero)

```

```

lemma LstSeqP_single [iff]: H  $\vdash$  LstSeqP (Eats Zero (HPair Zero x)) Zero x
  by (auto simp: LstSeqP.simps intro!: OrdP_SUCC_I HDomain_Incl_Eats_I Mem_Eats_I2)

```

lemma *NotInDom_LstSeqP_Eats*:
 $\{ \text{NotInDom } (\text{SUCC } k) \ s, \text{LstSeqP } s \ k \ y \} \vdash \text{LstSeqP } (\text{Eats } s \ (\text{HPair } (\text{SUCC } k) \ z)) \ (\text{SUCC } k) \ z$
by (*auto simp: LstSeqP.simps intro: HDomain_Incl_Eats_I Mem_Eats_I2 OrdP_SUCC_I HFun_Sigma_Eats*)

lemma *RestrictedP_HDomain_Incl*: $\{ \text{HDomain_Incl } s \ k, \text{RestrictedP } s \ k \ s' \} \vdash \text{HDomain_Incl } s' \ k$
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*: $\{ \text{HFun_Sigma } s, \text{RestrictedP } s \ k \ s' \} \vdash \text{HFun_Sigma } s'$
by (*metis Assume RestrictedP_imp_Subset Subset_HFun_Sigma rcut2*)

lemma *RestrictedP_LstSeqP*:
 $\{ \text{RestrictedP } s \ (\text{SUCC } k) \ s', \text{LstSeqP } s \ k \ y \} \vdash \text{LstSeqP } s' \ k \ y$
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*:
 $\{ \text{RestrictedP } s \ (\text{SUCC } k) \ s', \text{LstSeqP } s \ k \ y \}$
 $\vdash \text{LstSeqP } (\text{Eats } s' \ (\text{HPair } (\text{SUCC } k) \ z)) \ (\text{SUCC } k) \ z$
by (*blast intro: Mem_Neg_refl cut2 [OF NotInDom_LstSeqP_Eats]*
cut2 [OF RestrictedP_NotInDom] cut2 [OF RestrictedP_LstSeqP])

7.4 Ordinal Addition

7.4.1 Predicate form, defined on sequences

nominal_function *SeqHaddP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$
where $\llbracket \text{atom } l \# (sl,s,k,j); \text{atom } sl \# (s,j) \rrbracket \Longrightarrow$
 $\text{SeqHaddP } s \ j \ k \ y = \text{LstSeqP } s \ k \ y \text{ AND}$
 $\text{HPair Zero } j \text{ IN } s \text{ AND}$
 $\text{All2 } l \ k \ (\text{Ex } sl \ (\text{HPair } (\text{Var } l) \ (\text{Var } sl) \text{ IN } s \text{ AND}$
 $\text{HPair } (\text{SUCC } (\text{Var } l)) \ (\text{SUCC } (\text{Var } sl)) \text{ IN } s))$
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** *atom l # (sl,s,k,j)* *atom sl # (s,j)*
by (*metis obtain_fresh*)

thus *?thesis*
by *force*
qed

lemma *SeqHaddP_subst [simp]*:
 $(SeqHaddP\ s\ j\ k\ y)(i::=t) = SeqHaddP\ (subst\ i\ t\ s)\ (subst\ i\ t\ j)\ (subst\ i\ t\ k)\ (subst\ i\ t\ y)$
proof –
obtain *l::name and sl::name where atom l # (s,k,j,sl,t,i) atom sl # (s,k,j,t,i)*
by (*metis obtain_fresh*)
thus *?thesis*
by (*auto simp: SeqHaddP.simps [where l=l and sl=sl]*)
qed

declare *SeqHaddP.simps [simp del]*

nominal_function *HaddP :: tm ⇒ tm ⇒ tm ⇒ fm*
where $\llbracket atom\ s\ \#(x,y,z) \rrbracket \implies$
 $HaddP\ x\ y\ z = Ex\ s\ (SeqHaddP\ (Var\ s)\ x\ y\ z)$
by (*auto simp: eqvt_def HaddP_graph_aux_def flip_fresh_fresh (metis obtain_fresh)*)

nominal_termination (*eqvt*)
by *lexicographic_order*

lemma *HaddP_fresh_iff [simp]*: $a\ \# HaddP\ x\ y\ z \longleftrightarrow a\ \# x \wedge a\ \# y \wedge a\ \# z$
proof –
obtain *s::name where atom s # (x,y,z)*
by (*metis obtain_fresh*)
thus *?thesis*
by *force*
qed

lemma *HaddP_subst [simp]*: $(HaddP\ x\ y\ z)(i::=t) = HaddP\ (subst\ i\ t\ x)\ (subst\ i\ t\ y)\ (subst\ i\ t\ z)$
proof –
obtain *s::name where atom s # (x,y,z,t,i)*
by (*metis obtain_fresh*)
thus *?thesis*
by (*auto simp: HaddP.simps [of s]*)
qed

lemma *HaddP_cong*: $\llbracket H \vdash t\ EQ\ t'; H \vdash u\ EQ\ u'; H \vdash v\ EQ\ v' \rrbracket \implies H \vdash HaddP\ t\ u\ v\ IFF\ HaddP\ t'\ u'\ v'$
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} ⊢ HaddP x (SUCC y) (SUCC z)

```

7.4.2 Proving that these relations are functions

```

lemma SeqHaddP_Zero_E: {SeqHaddP s w Zero z} ⊢ w EQ z

```

proof –

```

  obtain l::name and sl::name where atom l # (s,w,z,sl) atom sl # (s,w)
  by (metis obtain_fresh)
thus ?thesis
  by (auto simp: SeqHaddP.simps [of l sl] LstSeqP.simps intro: HFun_Sigma_E)
qed

```

```

lemma SeqHaddP_SUCC_lemma:

```

```

  assumes y': atom y' # (s,j,k,y)
  shows {SeqHaddP s j (SUCC k) y} ⊢ Ex y' (SeqHaddP s j k (Var y') AND y EQ SUCC (Var y'))

```

proof –

```

  obtain l::name and sl::name where atom l # (s,j,k,y,y',sl) atom sl # (s,j,k,y,y')
  by (metis obtain_fresh)
thus ?thesis using y'
  apply (auto simp: SeqHaddP.simps [where s=s and l=l and sl=sl])
  apply (rule All2_SUCC_E' [where t=k, THEN rotate2], auto)
  apply (auto intro!: Ex_I [where x=Var sl])
  apply (blast intro: LstSeqP_SUCC) — showing SeqHaddP s j k (Var sl)
  apply (blast intro: LstSeqP_EQ)
  done

```

qed

```

lemma SeqHaddP_SUCC:

```

```

  assumes H ⊢ SeqHaddP s j (SUCC k) y atom y' # (s,j,k,y)
  shows H ⊢ Ex y' (SeqHaddP s j k (Var y') AND y EQ SUCC (Var y'))
  by (metis SeqHaddP_SUCC_lemma [THEN cut1] assms)

```

```

lemma SeqHaddP_unique: {OrdP x, SeqHaddP s w x y, SeqHaddP s' w x y'} ⊢ y' EQ y

```

```

lemma HaddP_unique: {HaddP w x y, HaddP w x y'} ⊢ y' EQ y

```

proof –

```

  obtain s::name and s'::name where atom s # (w,x,y,y') atom s' # (w,x,y,y',s)
  by (metis obtain_fresh)
hence {OrdP x, HaddP w x y, HaddP w x y'} ⊢ y' EQ y
  by (auto simp: HaddP.simps [of s _ _ y] HaddP.simps [of s' _ _ y']
    intro: SeqHaddP_unique [THEN cut3])
thus ?thesis
  by (metis HaddP_imp_OrdP cut_same thin1)
qed

```

```

lemma HaddP_Zero1: assumes H ⊢ OrdP x shows H ⊢ HaddP Zero x x

```

proof –

```

  fix k::name
  have {OrdP (Var k)} ⊢ HaddP Zero (Var k) (Var k)
  by (rule OrdInd2H [where i=k] (auto intro: HaddP_Zero2 HaddP_SUCC2 [THEN cut1]))
hence {} ⊢ OrdP (Var k) IMP HaddP Zero (Var k) (Var k)
  by (metis Imp_I)

```



```

hence {} ⊢ (OrdP (Var k) IMP HaddP Zero (Var k) (Var k))(k::=x)
  by (rule Subst) auto
hence {} ⊢ OrdP x IMP HaddP Zero x x
  by simp
thus ?thesis using assms
  by (metis MP_same thin0)
qed

lemma HaddP_Zero_D1: insert (HaddP Zero x y) H ⊢ x EQ y
  by (metis Assume HaddP_imp_OrdP HaddP_Zero1 HaddP_unique [THEN cut2] rcut1)

lemma HaddP_Zero_D2: insert (HaddP x Zero y) H ⊢ x EQ y
  by (metis Assume HaddP_Zero2 HaddP_unique [THEN cut2])

lemma HaddP_SUCC_Ex2:
  assumes H ⊢ HaddP x (SUCC y) z atom z' ‡ (x,y,z)
  shows H ⊢ Ex z' (HaddP x y (Var z') AND z EQ SUCC (Var z'))
proof –
  obtain s::name and s'::name where atom s ‡ (x,y,z,z') atom s' ‡ (x,y,z,z',s)
  by (metis obtain_fresh)
  hence { HaddP x (SUCC y) z } ⊢ Ex z' (HaddP x y (Var z') AND z EQ SUCC (Var z'))
  using assms
  apply (auto simp: HaddP.simps [of s _ _] HaddP.simps [of s' _ _])
  apply (rule cut_same [OF SeqHaddP_SUCC_lemma [of z']], auto)
  apply (rule Ex_I, auto)+
  done
  thus ?thesis
  by (metis assms(1) cut1)
qed

lemma HaddP_SUCC1: { HaddP x y z } ⊢ HaddP (SUCC x) y (SUCC z)
lemma HaddP_commute: {HaddP x y z, OrdP x} ⊢ HaddP y x z
lemma HaddP_SUCC_Ex1:
  assumes atom i ‡ (x,y,z)
  shows insert (HaddP (SUCC x) y z) (insert (OrdP x) H)
    ⊢ Ex i (HaddP x y (Var i) AND z EQ SUCC (Var i))
proof –
  have { HaddP (SUCC x) y z, OrdP x } ⊢ Ex i (HaddP x y (Var i) AND z EQ SUCC (Var i))
  apply (rule cut_same [OF HaddP_commute [THEN cut2]])
  apply (blast intro: OrdP_SUCC_I)+
  apply (rule cut_same [OF HaddP_SUCC_Ex2 [where z'=i]], blast)
  using assms apply auto
  apply (auto intro!: Ex_I [where x=Var i])
  by (metis AssumeH(2) HaddP_commute [THEN cut2] HaddP_imp_OrdP rotate2 thin1)
  thus ?thesis
  by (metis Assume AssumeH(2) cut2)
qed

lemma HaddP_inv2: {HaddP x y z, HaddP x y' z, OrdP x} ⊢ y' EQ y
lemma Mem_imp_subtract:
lemma HaddP_OrdP:
  assumes H ⊢ HaddP x y z H ⊢ OrdP x shows H ⊢ OrdP z
lemma HaddP_Mem_cancel_left:
  assumes H ⊢ HaddP x y' z' H ⊢ HaddP x y z H ⊢ OrdP x
  shows H ⊢ z' IN z IFF y' IN y
lemma HaddP_Mem_cancel_right_Mem:
  assumes H ⊢ HaddP x' y z' H ⊢ HaddP x y z H ⊢ x' IN x H ⊢ OrdP x
  shows H ⊢ z' 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 $i::\text{name} \ \# \ (k1, k2, k, x, A)$ **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 *atoms*: $\text{atom } i \ \# \ (x, y, z) \ \text{atom } j \ \# \ (i, x, y, z) \ \text{atom } k \ \# \ (i, j, x, y, z)$
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 $_ \vdash ?\text{scheme}$ **)**
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* $k \ \text{Var } i \ y \ \text{Var } j$, *THEN* *cut2*]], *auto*)
apply (*rule* *Ex_I* [*where* $x = \text{Var } k$], *auto*)
apply (*blast* *intro*: *OrdP_IN_SUCC_D* *Mem_cong* [*OF* $_ \text{Refl}$, *THEN* *Iff_MP_same*])
done
qed
hence $\{\text{OrdP } (\text{Var } i)\} \vdash (\text{HaddP } (\text{Var } i) \ y \ (\text{Var } j) \ \text{IMP } \text{Neg } ((\text{Var } j) \text{ IN } (\text{Var } i)))(j::=z)$
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 $\{\} \vdash (\text{OrdP } (\text{Var } i) \ \text{IMP } \text{HaddP } (\text{Var } i) \ y \ z \ \text{IMP } \text{Neg } (z \text{ IN } (\text{Var } i)))(i::=x)$
by (*metis* *Subst_emptyE*)
thus *?thesis*
using *atoms* **by** *simp* (*metis* *MP_same* *MP_null* *Neg_D* *assms*)
qed

lemma *exists_HaddP*:

assumes $H \vdash \text{OrdP } y \ \text{atom } j \ \# \ (x, y)$
shows $H \vdash \text{Ex } j \ (\text{HaddP } x \ y \ (\text{Var } j))$

proof –

obtain $i::\text{name}$
where *atoms*: $\text{atom } i \ \# \ (j, x, y)$
by (*metis* *obtain_fresh*)
have $\{\text{OrdP } (\text{Var } i)\} \vdash \text{Ex } j \ (\text{HaddP } x \ (\text{Var } i) \ (\text{Var } j))$
(is $_ \vdash ?\text{scheme}$ **)**
proof (*rule* *OrdInd2H*)
show $\{\} \vdash ?\text{scheme}(i::=\text{Zero})$
using *atoms* *assms*
by (*force* *intro*!: *Ex_I* [*where* $x = x$] *HaddP_Zero2*)

```

next
  show {} ⊢ All i (OrdP (Var i) IMP ?scheme IMP ?scheme(i:=SUCC (Var i)))
  using atoms assms
  apply auto
  apply (auto intro!: Ex_I [where x=SUCC (Var j)] HaddP_SUCC2)
  apply (metis HaddP_SUCC2 insert_commute thin1)
  done
qed
hence {} ⊢ OrdP (Var i) IMP Ex j (HaddP x (Var i) (Var j))
by (metis Imp_I)
hence {} ⊢ (OrdP (Var i) IMP Ex j (HaddP x (Var i) (Var j)))(i:=y)
using atoms by (force intro!: Subst)
thus ?thesis
using atoms assms by simp (metis MP_null assms(1))
qed

```

```

lemma HaddP_Mem_I:
  assumes H ⊢ HaddP x y z H ⊢ OrdP x shows H ⊢ x IN SUCC z
proof -
  have {HaddP x y z, OrdP x} ⊢ x IN SUCC z
  apply (rule OrdP_linear [of _ x SUCC z])
  apply (auto intro: OrdP_SUCC_I HaddP_OrdP)
  apply (rule HaddP_Mem_contra, blast)
  apply (metis Assume Mem_SUCC_I2 OrdP_IN_SUCC_D Sym_L thin1 thin2, blast)
  apply (blast intro: HaddP_Mem_contra Mem_SUCC_Refl OrdP_Trans)
  done
  thus ?thesis
  by (rule cut2) (auto intro: assms)
qed

```

7.5 A Shifted Sequence

```

nominal_function ShiftP :: tm ⇒ tm ⇒ tm ⇒ tm ⇒ fm
where [[atom x # (x',y,z,f,del,k); atom x' # (y,z,f,del,k); atom y # (z,f,del,k); atom z # (f,del,g,k)]] ⇒⇒
  ShiftP f k del g =
  All z (Var z IN g IFF
    (Ex x (Ex x' (Ex y ((Var z) EQ HPair (Var x') (Var y) AND
      HaddP del (Var x) (Var x') AND
      HPair (Var x) (Var y) IN f AND Var x IN k))))))
by (auto simp: eqvt_def ShiftP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)

```

```

nominal_termination (eqvt)
by lexicographic_order

```

```

lemma ShiftP_fresh_iff [simp]: a # ShiftP f k del g ⇔ a # f ∧ a # k ∧ a # del ∧ a # g
proof -
  obtain x::name and x'::name and y::name and z::name
  where atom x # (x',y,z,f,del,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: {} ⊢ ShiftP Zero k d Zero
proof –
obtain x::name and x':name and y::name and z::name
where atom x # (x',y,z,k,d) atom x' # (y,z,k,d) atom y # (z,k,d) atom z # (k,d)
by (metis obtain_fresh)
thus ?thesis
by (auto simp: ShiftP.simps [of x x' y z])
qed

lemma ShiftP_Mem1:
  {ShiftP f k del g, HPair a b IN f, HaddP del a a', a IN k} ⊢ HPair a' b IN g
proof –
obtain x::name and x':name and y::name and z::name
where atom x # (x',y,z,f,del,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} ⊢ Ex u ((Var u) IN k AND HaddP del (Var u) a AND HPair
(Var u) b IN f)
proof –
obtain x::name and x':name and y::name and z::name
where atoms: atom x # (x',y,z,f,del,g,k,a,u,b) atom x' # (y,z,f,del,g,k,a,u,b)
      atom y # (z,f,del,g,k,a,u,b) atom z # (f,del,g,k,a,u,b)
by (metis obtain_fresh)
thus ?thesis using assms
apply (auto simp: ShiftP.simps [of x x' y z])
apply (rule All_E [where x=HPair a b])
apply (auto intro!: Iff_E1 [OF Assume])
apply (rule Ex_I [where x=Var x])
apply (auto intro: Mem_cong [OF HPair_cong Refl, THEN Iff_MP2_same])
apply (blast intro: HaddP_cong [OF Refl Refl, THEN Iff_MP2_same])
done
qed

lemma ShiftP_Mem_D:
assumes H ⊢ ShiftP f k del g H ⊢ a IN g
      atom x # (x',y,a,f,del,k) atom x' # (y,a,f,del,k) atom y # (a,f,del,k)
shows H ⊢ (Ex x (Ex x' (Ex y (a EQ HPair (Var x') (Var y) AND
HaddP del (Var x) (Var x') AND
HPair (Var x) (Var y) IN f AND Var x IN k))))
(is _ ⊢ ?concl)

```

proof –

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 $H \vdash Ex\ t\ (ShiftP\ s\ k\ del\ (Var\ t))$

7.6 Union of Two Sets

nominal_function $UnionP :: tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$

where $atom\ i \# (x,y,z) \Longrightarrow UnionP\ x\ y\ z = All\ i\ (Var\ i\ IN\ z\ IFF\ (Var\ i\ IN\ x\ OR\ Var\ i\ IN\ y))$

by $(auto\ simp:\ eqvt_def\ UnionP_graph_aux_def\ flip_fresh_fresh)\ (metis\ obtain_fresh)$

nominal_termination $(eqvt)$

by $lexicographic_order$

lemma $UnionP_fresh_iff\ [simp]: a \# UnionP\ x\ y\ z \longleftrightarrow a \# x \wedge a \# y \wedge a \# z$

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 $\{\} \vdash UnionP\ Zero\ x\ x$
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 atom j # ( y,z,i )
    by (metis obtain_fresh)
  show {} ⊢ Ex z (UnionP (Var i) y (Var z))
    apply (rule Ind [of j i]) using j z i
    apply simp_all
    apply (rule Ex_I [where x=y], simp add: Union_Zero1)
    apply (auto del: Ex_EH)
    apply (rule Ex_E)
    apply (rule NegNeg_E)
    apply (rule Ex_E)
    apply (auto del: Ex_EH)
    apply (rule thin1, force intro: Ex_I [where x=Eats (Var z) (Var j)] Union_Eats)
  done
qed

```

```

lemma exists_UnionP:
  assumes z: atom z # ( x,y ) shows H ⊢ Ex z (UnionP x y (Var z))
proof –
  obtain i::name where i: atom i # ( y,z )
    by (metis obtain_fresh)
  hence {} ⊢ Ex z (UnionP (Var i) y (Var z))
    by (metis exists_Union_lemma fresh_Pair fresh_at_base(2) z)
  hence {} ⊢ (Ex z (UnionP (Var i) y (Var z)))(i:=x)
    by (metis Subst empty_iff)
  thus ?thesis using i z
    by (simp add: thin0)
qed

```

```

lemma UnionP_Mem1: { UnionP x y z, a IN x } ⊢ a IN z
proof –
  obtain i::name where atom i # ( x,y,z,a )
    by (metis obtain_fresh)
  thus ?thesis
    by (force simp: UnionP.simps [of i] intro: All_E [where x=a] Disj_I1 Iff_E2)
qed

```

```

lemma UnionP_Mem2: { UnionP x y z, a IN y } ⊢ a IN z
proof –
  obtain i::name where atom i # ( x,y,z,a )
    by (metis obtain_fresh)
  thus ?thesis

```

by (force simp: UnionP.simps [of i] intro: All_E [where x=a] Disj_I2 Iff_E2)
qed

lemma UnionP_Mem: { UnionP x y z, a IN z } ⊢ a IN x OR a IN y

proof –

obtain i::name **where** atom i # (x,y,z,a)

by (metis obtain_fresh)

thus ?thesis

by (force simp: UnionP.simps [of i] intro: All_E [where x=a] Iff_E1)

qed

lemma UnionP_Mem_E:

assumes H ⊢ UnionP x y z

and insert (a IN x) H ⊢ A

and insert (a IN y) H ⊢ A

shows insert (a IN z) H ⊢ A

using assms

by (blast intro: rotate2 cut_same [OF UnionP_Mem [THEN cut2]] thin1)

7.7 Append on Sequences

nominal_function SeqAppendP :: tm ⇒ tm ⇒ tm ⇒ tm ⇒ tm ⇒ fm

where [atom g1 # (g2,f1,k1,f2,k2,g); atom g2 # (f1,k1,f2,k2,g)] ⇒

SeqAppendP f1 k1 f2 k2 g =

(Ex g1 (Ex g2 (RestrictedP f1 k1 (Var g1) AND

ShiftP f2 k2 k1 (Var g2) AND

UnionP (Var g1) (Var g2) g)))

by (auto simp: eqvt_def SeqAppendP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)

nominal_termination (eqvt)

by lexicographic_order

lemma SeqAppendP_fresh_iff [simp]:

a # SeqAppendP f1 k1 f2 k2 g ↔ a # f1 ∧ a # k1 ∧ a # f2 ∧ a # k2 ∧ a # g

proof –

obtain g1::name **and** g2::name

where atom g1 # (g2,f1,k1,f2,k2,g) atom g2 # (f1,k1,f2,k2,g)

by (metis obtain_fresh)

thus ?thesis

by auto

qed

lemma subst_fm_SeqAppendP [simp]:

(SeqAppendP f1 k1 f2 k2 g)(i::=u) =

SeqAppendP (subst i u f1) (subst i u k1) (subst i u f2) (subst i u k2) (subst i u g)

proof –

obtain g1::name **and** g2::name

where atom g1 # (g2,f1,k1,f2,k2,g,i,u) atom g2 # (f1,k1,f2,k2,g,i,u)

by (metis obtain_fresh)

thus ?thesis

by (auto simp: SeqAppendP.simps [of g1 g2])

qed

lemma exists_SeqAppendP:

assumes atom g # (f1,k1,f2,k2)

shows H ⊢ Ex g (SeqAppendP f1 k1 f2 k2 (Var g))

proof –

obtain g1::name **and** g2::name

```

where atoms: atom g1 # (g2,f1,k1,f2,k2,g) atom g2 # (f1,k1,f2,k2,g)
  by (metis obtain_fresh)
hence {} ⊢ Ex g (SeqAppendP f1 k1 f2 k2 (Var g))
  using assms
  apply (auto simp: SeqAppendP.simps [of g1 g2])
  apply (rule cut_same [OF exists_RestrictedP [of g1 f1 k1]], auto)
  apply (rule cut_same [OF exists_ShiftP [of g2 f2 k2 k1]], auto)
  apply (rule cut_same [OF exists_UnionP [of g Var g1 Var g2]], auto)
  apply (rule Ex_I [where x=Var g], simp)
  apply (rule Ex_I [where x=Var g1], simp)
  apply (rule Ex_I [where x=Var g2], auto)
  done
thus ?thesis using assms
  by (metis thin0)
qed

```

```

lemma SeqAppendP_Mem1: {SeqAppendP f1 k1 f2 k2 g, HPair x y IN f1, x IN k1} ⊢ HPair x y IN g
proof –
  obtain g1::name and g2::name
    where atom g1 # (g2,f1,k1,f2,k2,g,x,y) atom g2 # (f1,k1,f2,k2,g,x,y)
    by (metis obtain_fresh)
  thus ?thesis
    by (auto simp: SeqAppendP.simps [of g1 g2] intro: UnionP_Mem1 [THEN cut2] RestrictedP_Mem
[THEN cut3])
qed

```

```

lemma SeqAppendP_Mem2: {SeqAppendP f1 k1 f2 k2 g, HaddP k1 x x', x IN k2, HPair x y IN f2} ⊢
HPair x' y IN g
proof –
  obtain g1::name and g2::name
    where atom g1 # (g2,f1,k1,f2,k2,g,x,x',y) atom g2 # (f1,k1,f2,k2,g,x,x',y)
    by (metis obtain_fresh)
  thus ?thesis
    by (auto simp: SeqAppendP.simps [of g1 g2] intro: UnionP_Mem2 [THEN cut2] ShiftP_Mem1 [THEN
cut4])
qed

```

```

lemma SeqAppendP_Mem_E:
  assumes H ⊢ SeqAppendP f1 k1 f2 k2 g
    and insert (HPair x y IN f1) (insert (x IN k1) H) ⊢ A
    and insert (HPair (Var u) y IN f2) (insert (HaddP k1 (Var u) x) (insert (Var u IN k2) H)) ⊢ A
    and u: atom u # (f1,k1,f2,k2,x,y,g,A) ∀ C ∈ H. atom u # C
  shows insert (HPair x y IN g) H ⊢ A

```

7.8 LstSeqP and SeqAppendP

```

lemma HDomain_Incl_SeqAppendP: — The And eliminates the need to prove cut5
  {SeqAppendP f1 k1 f2 k2 g, HDomain_Incl f1 k1 AND HDomain_Incl f2 k2,
  HaddP k1 k2 k, OrdP k1} ⊢ HDomain_Incl g k
declare SeqAppendP.simps [simp del]

```

```

lemma HFun_Sigma_SeqAppendP:
  {SeqAppendP f1 k1 f2 k2 g, HFun_Sigma f1, HFun_Sigma f2, OrdP k1} ⊢ HFun_Sigma g
lemma LstSeqP_SeqAppendP:
  assumes H ⊢ SeqAppendP f1 (SUCC k1) f2 (SUCC k2) g
    H ⊢ LstSeqP f1 k1 y1 H ⊢ LstSeqP f2 k2 y2 H ⊢ HaddP k1 k2 k
  shows H ⊢ LstSeqP g (SUCC k) y2
proof –

```



```

have {SeqAppendP f1 (SUCC k1) f2 (SUCC k2) g, LstSeqP f1 k1 y1, LstSeqP f2 k2 y2, HaddP k1 k2 k}
  ⊢ LstSeqP g (SUCC k) y2
  apply (auto simp: LstSeqP.simps intro: HaddP_OrdP OrdP_SUCC_I)
  apply (rule HDomain_Incl_SeqAppendP [THEN cut4])
  apply (rule AssumeH Conj_I)+
  apply (blast intro: HaddP_SUCC1 [THEN cut1] HaddP_SUCC2 [THEN cut1])
  apply (blast intro: HaddP_OrdP OrdP_SUCC_I)
  apply (rule HFun_Sigma_SeqAppendP [THEN cut4])
  apply (auto intro: HaddP_OrdP OrdP_SUCC_I)
  apply (blast intro: Mem_SUCC_Refl HaddP_SUCC1 [THEN cut1] HaddP_SUCC2 [THEN cut1]
    SeqAppendP_Mem2 [THEN cut4])
done
thus ?thesis using assms
  by (rule cut4)
qed

```

```

lemma SeqAppendP_NotInDom: {SeqAppendP f1 k1 f2 k2 g, HaddP k1 k2 k, OrdP k1} ⊢ NotInDom k g
proof -
  obtain x::name and z::name
    where atom x ‡ (z,f1,k1,f2,k2,g,k) atom z ‡ (f1,k1,f2,k2,g,k)
  by (metis obtain_fresh)
  thus ?thesis
    apply (auto simp: NotInDom.simps [of z])
    apply (rule SeqAppendP_Mem_E [where u=x])
    apply (rule AssumeH)+
    apply (blast intro: HaddP_Mem_contra, simp_all)
    apply (rule cut_same [where A=(Var x) EQ k2])
    apply (blast intro: HaddP_inv2 [THEN cut3])
    apply (blast intro: Mem_non_refl [where x=k2] Mem_cong [OF _ Refl, THEN Iff_MP_same])
  done
qed

```

```

lemma LstSeqP_SeqAppendP_Eats:
  assumes H ⊢ SeqAppendP f1 (SUCC k1) f2 (SUCC k2) g
    H ⊢ LstSeqP f1 k1 y1 H ⊢ LstSeqP f2 k2 y2 H ⊢ HaddP k1 k2 k
  shows H ⊢ LstSeqP (Eats g (HPair (SUCC (SUCC k)) z)) (SUCC (SUCC k)) z
proof -
  have {SeqAppendP f1 (SUCC k1) f2 (SUCC k2) g, LstSeqP f1 k1 y1, LstSeqP f2 k2 y2, HaddP k1 k2 k}
    ⊢ LstSeqP (Eats g (HPair (SUCC (SUCC k)) z)) (SUCC (SUCC k)) z
    apply (rule cut2 [OF NotInDom_LstSeqP_Eats])
    apply (rule SeqAppendP_NotInDom [THEN cut3])
    apply (rule AssumeH)
    apply (metis HaddP_SUCC1 HaddP_SUCC2 cut1 thin1)
    apply (metis Assume LstSeqP_OrdP OrdP_SUCC_I insert_commute)
    apply (blast intro: LstSeqP_SeqAppendP)
  done
  thus ?thesis using assms
    by (rule cut4)
qed

```

7.9 Substitution and Abstraction on Terms

7.9.1 Atomic cases

```

lemma SeqStTermP_Var_same:
  assumes atom s ‡ (k,v,i) atom k ‡ (v,i)
  shows {VarP v} ⊢ Ex s (Ex k (SeqStTermP v i v i (Var s) (Var k)))
proof -

```

```

obtain l::name and sl::name and sl'::name and m::name and sm::name and sm'::name
and n::name and sn::name and sn'::name
where atom l # (v,i,s,k,sl,sl',m,n,sm,sm',sn,sn')
      atom sl # (v,i,s,k,sl',m,n,sm,sm',sn,sn')
      atom sl' # (v,i,s,k,m,n,sm,sm',sn,sn')
      atom m # (v,i,s,k,n,sm,sm',sn,sn') atom n # (v,i,s,k,sm,sm',sn,sn')
      atom sm # (v,i,s,k,sm',sn,sn') atom sm' # (v,i,s,k,sn,sn')
      atom sn # (v,i,s,k,sn') atom sn' # (v,i,s,k)
by (metis obtain_fresh)
thus ?thesis using assms
apply (simp add: SeqStTermP.simps [of l ___ v i sl sl' m n sm sm' sn sn'])
apply (rule Ex_I [where x = Eats Zero (HPair Zero (HPair v i))], simp)
apply (rule Ex_I [where x = Zero], auto intro!: Mem_SUCC_EH)
apply (rule Ex_I [where x = v], simp)
apply (rule Ex_I [where x = i], auto intro: Disj_I1 Mem_Eats_I2 HPair_cong)
done
qed

lemma SeqStTermP_Var_diff:
assumes atom s # (k,v,w,i) atom k # (v,w,i)
shows { VarP v, VarP w, Neg (v EQ w) } ⊢ Ex s (Ex k (SeqStTermP v i w w (Var s) (Var k)))
proof –
obtain l::name and sl::name and sl'::name and m::name and sm::name and sm'::name
and n::name and sn::name and sn'::name
where atom l # (v,w,i,s,k,sl,sl',m,n,sm,sm',sn,sn')
      atom sl # (v,w,i,s,k,sl',m,n,sm,sm',sn,sn')
      atom sl' # (v,w,i,s,k,m,n,sm,sm',sn,sn')
      atom m # (v,w,i,s,k,n,sm,sm',sn,sn') atom n # (v,w,i,s,k,sm,sm',sn,sn')
      atom sm # (v,w,i,s,k,sm',sn,sn') atom sm' # (v,w,i,s,k,sn,sn')
      atom sn # (v,w,i,s,k,sn') atom sn' # (v,w,i,s,k)
by (metis obtain_fresh)
thus ?thesis using assms
apply (simp add: SeqStTermP.simps [of l ___ v i sl sl' m n sm sm' sn sn'])
apply (rule Ex_I [where x = Eats Zero (HPair Zero (HPair w w))], simp)
apply (rule Ex_I [where x = Zero], auto intro!: Mem_SUCC_EH)
apply (rule rotate2 [OF Swap])
apply (rule Ex_I [where x = w], simp)
apply (rule Ex_I [where x = w], auto simp: VarP_def)
apply (blast intro: HPair_cong Mem_Eats_I2)
apply (blast intro: Sym OrdNotEqP_I Disj_I1 Disj_I2)
done
qed

lemma SeqStTermP_Zero:
assumes atom s # (k,v,i) atom k # (v,i)
shows { VarP v } ⊢ Ex s (Ex k (SeqStTermP v i Zero Zero (Var s) (Var k)))
corollary SubstTermP_Zero: { TermP t } ⊢ SubstTermP «Var v» t Zero Zero
proof –
obtain s::name and k::name where atom s # (v,t,k) atom k # (v,t)
by (metis obtain_fresh)
thus ?thesis
by (auto simp: SubstTermP.simps [of s ___ ___ k] intro: SeqStTermP_Zero [THEN cut1])
qed

corollary SubstTermP_Var_same: { VarP v, TermP t } ⊢ SubstTermP v t v t
proof –
obtain s::name and k::name where atom s # (v,t,k) atom k # (v,t)
by (metis obtain_fresh)

```

thus *?thesis*
by (*auto simp: SubstTermP.simps [of s _ _ _ k] intro: SeqStTermP_Var_same [THEN cut1]*)
qed

corollary *SubstTermP_Var_diff*: $\{ \text{VarP } v, \text{VarP } w, \text{Neg } (v \text{ EQ } w), \text{TermP } t \} \vdash \text{SubstTermP } v \ t \ w \ w$
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 $\{ \text{VarP } v, \text{IndP } t \} \vdash \text{Ex } s \ (\text{Ex } k \ (\text{SeqStTermP } v \ i \ t \ t \ (\text{Var } s) \ (\text{Var } k)))$
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*: $\{ \text{VarP } v, \text{IndP } w, \text{TermP } t \} \vdash \text{SubstTermP } v \ t \ w \ w$
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

7.9.2 Non-atomic cases

lemma *SeqStTermP_Eats*:
assumes *sk: atom s # (k,s1,s2,k1,k2,t1,t2,u1,u2,v,i)*
atom k # (t1,t2,u1,u2,v,i)
shows $\{ \text{SeqStTermP } v \ i \ t1 \ u1 \ s1 \ k1, \text{SeqStTermP } v \ i \ t2 \ u2 \ s2 \ k2 \}$
 $\vdash \text{Ex } s \ (\text{Ex } k \ (\text{SeqStTermP } v \ i \ (Q_Eats \ t1 \ t2) \ (Q_Eats \ u1 \ u2) \ (\text{Var } s) \ (\text{Var } k)))$
theorem *SubstTermP_Eats*:
 $\{ \text{SubstTermP } v \ i \ t1 \ u1, \text{SubstTermP } v \ i \ t2 \ u2 \} \vdash \text{SubstTermP } v \ i \ (Q_Eats \ t1 \ t2) \ (Q_Eats \ u1 \ u2)$
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

7.9.3 Substitution over a constant

lemma SeqConstP_lemma:

```

  assumes atom m # (s,k,c,n,sm,sn)  atom n # (s,k,c,sm,sn)
         atom sm # (s,k,c,sn)      atom sn # (s,k,c)
  shows { SeqConstP s k c }
        ⊢ c EQ Zero OR
         Ex m (Ex n (Ex sm (Ex sn (Var m IN k AND Var n IN k AND
                               SeqConstP s (Var m) (Var sm) AND
                               SeqConstP s (Var n) (Var sn) AND
                               c EQ Q_Eats (Var sm) (Var sn))))))

```

lemma SeqConstP_imp_SubstTermP: {SeqConstP s kk c, TermP t} ⊢ SubstTermP «Var w» t c c

theorem SubstTermP_Const: {ConstP c, TermP t} ⊢ SubstTermP «Var w» t c c

proof –

```

  obtain s::name and k::name where atom s # (c,t,w,k) atom k # (c,t,w)
  by (metis obtain_fresh)
  thus ?thesis
    by (auto simp: CTermP.simps [of k s c] SeqConstP_imp_SubstTermP)

```

qed

7.10 Substitution on Formulas

7.10.1 Membership

lemma SubstAtomicP_Mem:

```

  {SubstTermP v i x x', SubstTermP v i y y'} ⊢ SubstAtomicP v i (Q_Mem x y) (Q_Mem x' y')

```

proof –

```

  obtain t::name and u::name and t'::name and u'::name
  where atom t # (v,i,x,x',y,y',t',u,u') atom t' # (v,i,x,x',y,y',u,u')
        atom u # (v,i,x,x',y,y',u') atom u' # (v,i,x,x',y,y')
  by (metis obtain_fresh)
  thus ?thesis
    apply (simp add: SubstAtomicP.simps [of t _ _ _ _ t' u u'])
    apply (rule Ex_I [where x = x], simp)
    apply (rule Ex_I [where x = y], simp)
    apply (rule Ex_I [where x = x'], simp)
    apply (rule Ex_I [where x = y'], auto intro: Disj_I2)
  done

```

qed

lemma SeqSubstFormP_Mem:

```

  assumes atom s # (k,x,y,x',y',v,i) atom k # (x,y,x',y',v,i)
  shows {SubstTermP v i x x', SubstTermP v i y y'}
        ⊢ Ex s (Ex k (SeqSubstFormP v i (Q_Mem x y) (Q_Mem x' y') (Var s) (Var k)))

```

proof –

```

  let ?vs = (s,k,x,y,x',y',v,i)
  obtain l::name and sl::name and sl'::name and m::name and n::name and sm::name and sm'::name
  and sn::name and sn'::name
  where atom l # (?vs,sl,sl',m,n,sm,sm',sn,sn')

```

```

    atom sl # (?vs,sl',m,n,sm,sm',sn,sn') atom sl' # (?vs,m,n,sm,sm',sn,sn')
    atom m # (?vs,n,sm,sm',sn,sn') atom n # (?vs,sm,sm',sn,sn')
    atom sm # (?vs,sm',sn,sn') atom sm' # (?vs,sn,sn')
    atom sn # (?vs,sn') atom sn' # ?vs
  by (metis obtain_fresh)
thus ?thesis
using assms
apply (auto simp: SeqSubstFormP.simps [of l Var s ___ sl sl' m n sm sm' sn sn'])
apply (rule Ex_I [where x = Eats Zero (HPair Zero (HPair (Q_Mem x y) (Q_Mem x' y')))], simp)
apply (rule Ex_I [where x = Zero], auto intro: Mem_SUCC_EH)
apply (rule Ex_I [where x = Q_Mem x y], simp)
apply (rule Ex_I [where x = Q_Mem x' y'], auto intro: Mem_Eats_I2 HPair_cong)
apply (blast intro: SubstAtomicP_Mem [THEN cut2] Disj_I1)
done
qed

```

lemma *SubstFormP_Mem*:

```

{SubstTermP v i x', SubstTermP v i y y'} ⊢ SubstFormP v i (Q_Mem x y) (Q_Mem x' y')
proof -
  obtain k1::name and s1::name and k2::name and s2::name and k::name and s::name
  where atom s1 # (v,i,x,y,x',y') atom k1 # (v,i,x,y,x',y',s1)
        atom s2 # (v,i,x,y,x',y',k1,s1) atom k2 # (v,i,x,y,x',y',s2,k1,s1)
        atom s # (v,i,x,y,x',y',k2,s2,k1,s1) atom k # (v,i,x,y,x',y',s,k2,s2,k1,s1)
  by (metis obtain_fresh)
  thus ?thesis
  by (auto simp: SubstFormP.simps [of s v i (Q_Mem x y) _ k]
        SubstFormP.simps [of s1 v i x x' k1]
        SubstFormP.simps [of s2 v i y y' k2]
        intro: SubstTermP_imp_TermP SubstTermP_imp_VarP SeqSubstFormP_Mem thin1)
qed

```

7.10.2 Equality

lemma *SubstAtomicP_Eq*:

```

{SubstTermP v i x x', SubstTermP v i y y'} ⊢ SubstAtomicP v i (Q_Eq x y) (Q_Eq x' y')
proof -
  obtain t::name and u::name and t'::name and u'::name
  where atom t # (v,i,x,x',y,y',t',u,u') atom t' # (v,i,x,x',y,y',u,u')
        atom u # (v,i,x,x',y,y',u') atom u' # (v,i,x,x',y,y')
  by (metis obtain_fresh)
  thus ?thesis
  apply (simp add: SubstAtomicP.simps [of t ___ t' u u'])
  apply (rule Ex_I [where x = x], simp)
  apply (rule Ex_I [where x = y], simp)
  apply (rule Ex_I [where x = x'], simp)
  apply (rule Ex_I [where x = y'], auto intro: Disj_I1)
  done
qed

```

lemma *SeqSubstFormP_Eq*:

```

assumes sk: atom s # (k,x,y,x',y',v,i) atom k # (x,y,x',y',v,i)
shows {SubstTermP v i x x', SubstTermP v i y y'}
      ⊢ Ex s (Ex k (SeqSubstFormP v i (Q_Eq x y) (Q_Eq x' y') (Var s) (Var k)))
proof -
  let ?vs = (s,k,x,y,x',y',v,i)
  obtain l::name and sl::name and sl'::name and m::name and n::name and sm::name and sm'::name
  and sn::name and sn'::name
  where atom l # (?vs,sl,sl',m,n,sm,sm',sn,sn')

```

```

    atom sl # (?vs,sl',m,n,sm,sm',sn,sn') atom sl' # (?vs,m,n,sm,sm',sn,sn')
    atom m # (?vs,n,sm,sm',sn,sn') atom n # (?vs,sm,sm',sn,sn')
    atom sm # (?vs,sm',sn,sn') atom sm' # (?vs,sn,sn')
    atom sn # (?vs,sn') atom sn' # ?vs
  by (metis obtain_fresh)
thus ?thesis
using sk
apply (auto simp: SeqSubstFormP.simps [of l Var s ___ sl sl' m n sm sm' sn sn'])
apply (rule Ex_I [where x = Eats Zero (HPair Zero (HPair (Q_Eq x y) (Q_Eq x' y')))], simp)
apply (rule Ex_I [where x = Zero], auto intro!: Mem_SUCC_EH)
apply (rule Ex_I [where x = Q_Eq x y], simp)
apply (rule Ex_I [where x = Q_Eq x' y], auto)
  apply (metis Mem_Eats_I2 Assume HPair_cong Refl)
  apply (blast intro: SubstAtomicP_Eq [THEN cut2] Disj_I1)
done
qed

```

lemma *SubstFormP_Eq*:
 $\{SubstTermP\ v\ i\ x',\ SubstTermP\ v\ i\ y'\} \vdash SubstFormP\ v\ i\ (Q_Eq\ x\ y)\ (Q_Eq\ x'\ y')$
proof –
obtain *k1::name and s1::name and k2::name and s2::name and k::name and s::name*
where *atom s1 # (v,i,x,y,x',y') atom k1 # (v,i,x,y,x',y',s1)*
atom s2 # (v,i,x,y,x',y',k1,s1) atom k2 # (v,i,x,y,x',y',s2,k1,s1)
atom s # (v,i,x,y,x',y',k2,s2,k1,s1) atom k # (v,i,x,y,x',y',s,k2,s2,k1,s1)
by (metis obtain_fresh)
thus ?thesis
by (auto simp: SubstFormP.simps [of s v i (Q_Eq x y) _ k]
SubstFormP.simps [of s1 v i x x' k1]
SubstFormP.simps [of s2 v i y y' k2]
intro: SeqSubstFormP_Eq SubstTermP_imp_TermP SubstTermP_imp_VarP thin1)
qed

7.10.3 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

7.10.4 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

```

7.10.5 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

7.11 Constant Terms

lemma *ConstP_Zero:* $\{\} \vdash ConstP\ Zero$

proof –

obtain *s::name and k::name and l::name and sl::name and m::name and n::name and sm::name*
and *sn::name*

where *atoms:*

atom s # (k,l,sl,m,n,sm,sn)

atom k # (l,sl,m,n,sm,sn)

atom l # (sl,m,n,sm,sn)

atom sl # (m,n,sm,sn)

atom m # (n,sm,sn)

atom n # (sm,sn)

atom sm # sn

by (*metis obtain_fresh*)

then show *?thesis*

apply (*subst CTermP.simps[of k s]; auto?*)

apply (*rule Ex_I[of _ _ _ Eats Zero (HPair Zero Zero)]; auto?*)

apply (*rule Ex_I[of _ _ _ Zero]; auto?*)

apply (*subst SeqCTermP.simps[of l _ _ sl m n sm sn]; auto?*)

apply (*rule Ex_I[of _ _ _ Zero]; auto?*)

apply (*rule Mem_SUCC_E[OF Mem_Zero_E]*)

apply (*rule Mem_Eats_I2*)

apply (*rule HPair_cong[OF Assume Refl]*)

apply (*rule Disj_II[OF Refl]*)

done

qed

lemma *SeqConstP_Eats:*

```

assumes atom s # (k,s1,s2,k1,k2,t1,t2) atom k # (s1,s2,k1,k2,t1,t2)
shows {SeqConstP s1 k1 t1, SeqConstP s2 k2 t2}
  ⊢ Ex s (Ex k (SeqConstP (Var s) (Var k) (Q_Eats t1 t2)))
theorem ConstP_Eats: {ConstP t1, ConstP t2} ⊢ ConstP (Q_Eats t1 t2)
proof –
  obtain k1::name and s1::name and k2::name and s2::name and k::name and s::name
  where atom s1 # (t1,t2) atom k1 # (t1,t2,s1)
    atom s2 # (t1,t2,k1,s1) atom k2 # (t1,t2,s2,k1,s1)
    atom s # (t1,t2,k2,s2,k1,s1) atom k # (t1,t2,s,k2,s2,k1,s1)
  by (metis obtain_fresh)
  thus ?thesis
  by (auto simp: CTermP.simps [of k s (Q_Eats t1 t2)]
    CTermP.simps [of k1 s1 t1] CTermP.simps [of k2 s2 t2]
    intro!: SeqConstP_Eats [THEN cut2])
qed

lemma TermP_Zero: {} ⊢ TermP Zero
proof –
  obtain s::name and k::name and l::name and sl::name and m::name and n::name and sm::name
and sn::name
  where atoms:
    atom s # (k,l,sl,m,n,sm,sn)
    atom k # (l,sl,m,n,sm,sn)
    atom l # (sl,m,n,sm,sn)
    atom sl # (m,n,sm,sn)
    atom m # (n,sm,sn)
    atom n # (sm,sn)
    atom sm # sn
  by (metis obtain_fresh)
  then show ?thesis
  apply (subst CTermP.simps[of k s]; auto?)
  apply (rule Ex_I[of _ _ _ Eats Zero (HPair Zero Zero)]; auto?)
  apply (rule Ex_I[of _ _ _ Zero]; auto?)
  apply (subst SeqCTermP.simps[of l _ _ sl m n sm sn]; auto?)
  apply (rule Ex_I[of _ _ _ Zero]; auto?)
  apply (rule Mem_SUCC_E[OF Mem_Zero_E])
  apply (rule Mem_Eats_I2)
  apply (rule HPair_cong[OF Assume Refl])
  apply (rule Disj_I1[OF Refl])
  done
qed

lemma TermP_Var: {} ⊢ TermP «Var x»
proof –
  obtain s::name and k::name and l::name and sl::name and m::name and n::name and sm::name
and sn::name
  where atoms:
    atom s # (k,l,sl,m,n,sm,sn,x)
    atom k # (l,sl,m,n,sm,sn,x)
    atom l # (sl,m,n,sm,sn,x)
    atom sl # (m,n,sm,sn,x)
    atom m # (n,sm,sn,x)
    atom n # (sm,sn,x)
    atom sm # (sn,x)
    atom sn # x
  by (metis obtain_fresh)
  then show ?thesis
  apply (subst CTermP.simps[of k s]; auto?)

```



```

apply (rule Ex_I[of _ _ _ Eats Zero (HPair Zero «Var x»)]; auto?)
apply (rule Ex_I[of _ _ _ Zero]; auto?)
apply (subst SeqCTermP.simps[of l _ _ sl m n sm sn]; auto?)
apply (rule Ex_I[of _ _ _ «Var x»]; auto?)
apply (rule Mem_SUCC_E[OF Mem_Zero_E])
apply (rule Mem_Eats_I2)
apply (rule HPair_cong[OF Assume Refl])
apply (rule Disj_I2[OF Disj_I1])
apply (auto simp: VarP_Var)
done
qed

```

```

lemma SeqTermP_Eats:
  assumes atom s # (k,s1,s2,k1,k2,t1,t2) atom k # (s1,s2,k1,k2,t1,t2)
  shows {SeqTermP s1 k1 t1, SeqTermP s2 k2 t2}
    ⊢ Ex s (Ex k (SeqTermP (Var s) (Var k) (Q_Eats t1 t2)))
theorem TermP_Eats: {TermP t1, TermP t2} ⊢ TermP (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!: SeqTermP_Eats [THEN cut2])

```

qed

7.12 Proofs

```

lemma PrfP_inference:
  assumes atom s # (k,s1,s2,k1,k2,α1,α2,β) atom k # (s1,s2,k1,k2,α1,α2,β)
  shows {PrfP s1 k1 α1, PrfP s2 k2 α2, ModPonP α1 α2 β OR ExistsP α1 β OR SubstP α1 β}
    ⊢ Ex k (Ex s (PrfP (Var s) (Var k) β))
corollary Pfp_inference: {Pfp α1, Pfp α2, ModPonP α1 α2 β OR ExistsP α1 β OR SubstP α1 β} ⊢
  Pfp β
proof -
  obtain k1::name and s1::name and k2::name and s2::name and k::name and s::name
  where atom s1 # (α1,α2,β) atom k1 # (α1,α2,β,s1)
    atom s2 # (α1,α2,β,k1,s1) atom k2 # (α1,α2,β,s2,k1,s1)
    atom s # (α1,α2,β,k2,s2,k1,s1)
    atom k # (α1,α2,β,s,k2,s2,k1,s1)
  by (metis obtain_fresh)
  thus ?thesis
  apply (simp add: Pfp.simps [of k s β] Pfp.simps [of k1 s1 α1] Pfp.simps [of k2 s2 α2])
  apply (auto intro!: PrfP_inference [of s k Var s1 Var s2, THEN cut3] del: Disj_EH)
  done

```

qed

```

theorem Pfp_implies_SubstForm_Pfp:
  assumes H ⊢ Pfp y H ⊢ SubstFormP x t y z
  shows H ⊢ Pfp z
proof -
  obtain u::name and v::name
  where atoms: atom u # (t,x,y,z,v) atom v # (t,x,y,z)
  by (metis obtain_fresh)
  show ?thesis

```

```

apply (rule Pfp_inference [of y, THEN cut3])
apply (rule assms)+
using atoms
apply (auto simp: SubstP.simps [of u _ _ v] intro!: Disj_I2)
apply (rule Ex_I [where x=x], simp)
apply (rule Ex_I [where x=t], simp add: assms)
done
qed

theorem Pfp_implies_ModPon_Pfp:  $\llbracket H \vdash Pfp (Q\_Imp\ x\ y); H \vdash Pfp\ x \rrbracket \implies H \vdash Pfp\ y$ 
  by (force intro: Pfp_inference [of x, THEN cut3] Disj_I1 simp add: ModPonP_def)

corollary Pfp_implies_ModPon_Pfp_quot:  $\llbracket H \vdash Pfp\ \langle\langle \alpha\ IMP\ \beta \rangle\rangle; H \vdash Pfp\ \langle\langle \alpha \rangle\rangle \rrbracket \implies H \vdash Pfp\ \langle\langle \beta \rangle\rangle$ 
  by (auto simp: quot_fm_def intro: Pfp_implies_ModPon_Pfp)

lemma TermP_quot:
  fixes  $\alpha :: tm$ 
  shows  $\{\} \vdash TermP\ \langle\langle \alpha \rangle\rangle$ 
  by (induct  $\alpha$  rule: tm.induct)
    (auto simp: quot_Eats intro: TermP_Zero TermP_Var TermP_Eats[THEN cut2])

lemma TermP_quot_dbtm:
  fixes  $\alpha :: tm$ 
  assumes wf_dbtm u
  shows  $\{\} \vdash TermP\ (quot\_dbtm\ u)$ 
  using assms
  by (induct u rule: dbtm.induct)
    (auto simp: quot_Eats intro: TermP_Zero
      TermP_Var[unfolded quot_tm_def, simplified] TermP_Eats[THEN cut2])

```

7.13 Formulas

7.14 Abstraction on Formulas

7.14.1 Membership

```

lemma AbstAtomicP_Mem:
   $\{AbstTermP\ v\ i\ x\ x', AbstTermP\ v\ i\ y\ y'\} \vdash AbstAtomicP\ v\ i\ (Q\_Mem\ x\ y)\ (Q\_Mem\ x'\ y')$ 
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: AbstAtomicP.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 SeqAbstFormP_Mem:
  assumes  $atom\ s \# (k, x, y, x', y', v, i)$   $atom\ k \# (x, y, x', y', v, i)$ 
  shows  $\{AbstTermP\ v\ i\ x\ x', AbstTermP\ v\ i\ y\ y'\}$ 
     $\vdash Ex\ s\ (Ex\ k\ (SeqAbstFormP\ v\ i\ (Q\_Mem\ x\ y)\ (Q\_Mem\ x'\ y')\ (Var\ s)\ (Var\ k)))$ 

```

```

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
  and sli smi sni :: name
  where
    atom sni # (?vs,sl,sl',m,n,sm,sm',sn,sn',l,sli,smi)
    atom smi # (?vs,sl,sl',m,n,sm,sm',sn,sn',l,sli)
    atom sli # (?vs,sl,sl',m,n,sm,sm',sn,sn',l)
    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: SeqAbstFormP.simps [of l Var s __ sli sl sl' m n smi sm sm' sni sn sn'])
  apply (rule Ex_I [where x = Eats Zero (HPair Zero (HPair i (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 = i], simp)
  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: AbstAtomicP_Mem [THEN cut2] Disj_I1)
  done
qed

```

```

lemma AbstFormP_Mem:
  {AbstTermP v i x x', AbstTermP v i y y'} ⊢ AbstFormP v i (Q_Mem x y) (Q_Mem x' y')
proof –
  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: AbstFormP.simps [of s v i (Q_Mem x y) __ k]
    AbstFormP.simps [of s1 v i x x' k1]
    AbstFormP.simps [of s2 v i y y' k2]
    intro: AbstTermP_imp_VarP AbstTermP_imp_OrdP SeqAbstFormP_Mem thin1)
qed

```

7.14.2 Equality

```

lemma AbstAtomicP_Eq:
  {AbstTermP v i x x', AbstTermP v i y y'} ⊢ AbstAtomicP v i (Q_Eq x y) (Q_Eq x' y')
proof –
  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: AbstAtomicP.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 *SeqAbstFormP_Eq*:

assumes $sk: \text{atom } s \# (k, x, y, x', y', v, i) \text{ atom } k \# (x, y, x', y', v, i)$
 shows $\{\text{AbstTermP } v \ i \ x \ x', \text{ AbstTermP } v \ i \ y \ y'\}$
 $\vdash \text{Ex } s \ (\text{Ex } k \ (\text{SeqAbstFormP } v \ i \ (Q_Eq \ x \ y) \ (Q_Eq \ x' \ y') \ (\text{Var } s) \ (\text{Var } k)))$

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** $sn'::\text{name}$

and $sli \ smi \ sni :: \text{name}$

where

$\text{atom } sni \# (?vs, sl, sl', m, n, sm, sm', sn, sn', l, sli, smi)$
 $\text{atom } smi \# (?vs, sl, sl', m, n, sm, sm', sn, sn', l, sli)$
 $\text{atom } sli \# (?vs, sl, sl', m, n, sm, sm', sn, sn', l)$
 $\text{atom } l \# (?vs, sl, sl', m, n, sm, sm', sn, sn')$
 $\text{atom } sl \# (?vs, sl', m, n, sm, sm', sn, sn') \text{ atom } sl' \# (?vs, m, n, sm, sm', sn, sn')$
 $\text{atom } m \# (?vs, n, sm, sm', sn, sn') \text{ atom } n \# (?vs, sm, sm', sn, sn')$
 $\text{atom } sm \# (?vs, sm', sn, sn') \text{ atom } sm' \# (?vs, sn, sn')$
 $\text{atom } sn \# (?vs, sn') \text{ atom } sn' \# ?vs$

by (*metis obtain_fresh*)

thus *?thesis*

using *sk*

apply (*auto simp: SeqAbstFormP.simps [of l Var s _ _ sli sl sl' m n smi sm sm' sni sn sn']*)

apply (*rule Ex_I [where x = Eats Zero (HPair Zero (HPair i (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 = i], simp*)

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: AbstAtomicP_Eq [THEN cut2] Disj_I1*)

done

qed

lemma *AbstFormP_Eq*:

$\{\text{AbstTermP } v \ i \ x \ x', \text{ AbstTermP } v \ i \ y \ y'\} \vdash \text{AbstFormP } v \ i \ (Q_Eq \ x \ y) \ (Q_Eq \ x' \ y')$

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: AbstFormP.simps [of s v i (Q_Eq x y) _ k]*)

AbstFormP.simps [of s1 v i x x' k1]

AbstFormP.simps [of s2 v i y y' k2]

intro: SeqAbstFormP_Eq AbstTermP_imp_OrdP AbstTermP_imp_VarP thin1)

qed

7.14.3 Negation

lemma *SeqAbstFormP_Neg*:

assumes $\text{atom } s \# (k, s1, k1, x, x', v, i) \text{ atom } k \# (s1, k1, x, x', v, i)$

shows $\{\text{SeqAbstFormP } v \ i \ x \ x' \ s1 \ k1, \text{ OrdP } i, \text{ VarP } v\}$

$\vdash \text{Ex } s \ (\text{Ex } k \ (\text{SeqAbstFormP } v \ i \ (Q_Neg \ x) \ (Q_Neg \ x') \ (\text{Var } s) \ (\text{Var } k)))$

theorem *AbstFormP_Neg*: $\{\text{AbstFormP } v \ i \ x \ x'\} \vdash \text{AbstFormP } v \ i \ (Q_Neg \ x) \ (Q_Neg \ x')$

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: AbstFormP.simps [of s v i Q_Neg x _ k] AbstFormP.simps [of s1 v i x x' k1]*
intro: SeqAbstFormP_Neg [THEN cut3])
qed

7.14.4 Disjunction

lemma *SeqAbstFormP_Disj*:
assumes $atom\ s \# (k,s1,s2,k1,k2,x,y,x',y',v,i)$ $atom\ k \# (s1,s2,k1,k2,x,y,x',y',v,i)$
shows $\{SeqAbstFormP\ v\ i\ x\ x'\ s1\ k1,$
 $SeqAbstFormP\ v\ i\ y\ y'\ s2\ k2,$ *OrdP* $i,$ *VarP* $v\}$
 $\vdash\ Ex\ s\ (Ex\ k\ (SeqAbstFormP\ v\ i\ (Q_Disj\ x\ y)\ (Q_Disj\ x'\ y')\ (Var\ s)\ (Var\ k)))$
theorem *AbstFormP_Disj*:
 $\{AbstFormP\ v\ i\ x\ x',\ AbstFormP\ v\ i\ y\ y'\} \vdash AbstFormP\ v\ i\ (Q_Disj\ x\ y)\ (Q_Disj\ x'\ y')$
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: AbstFormP.simps [of s v i Q_Disj x y _ k]*
AbstFormP.simps [of s1 v i x x' k1]
AbstFormP.simps [of s2 v i y y' k2]
intro: SeqAbstFormP_Disj [THEN cut4])
qed

7.14.5 Existential

lemma *SeqAbstFormP_Ex*:
assumes $atom\ s \# (k,s1,k1,x,x',v,i)$ $atom\ k \# (s1,k1,x,x',v,i)$
shows $\{SeqAbstFormP\ v\ (SUCC\ i)\ x\ x'\ s1\ k1,$ *OrdP* $i,$ *VarP* $v\}$
 $\vdash\ Ex\ s\ (Ex\ k\ (SeqAbstFormP\ v\ i\ (Q_Ex\ x)\ (Q_Ex\ x')\ (Var\ s)\ (Var\ k)))$
theorem *AbstFormP_Ex*: $\{AbstFormP\ v\ (SUCC\ i)\ x\ x'\} \vdash AbstFormP\ v\ i\ (Q_Ex\ x)\ (Q_Ex\ x')$
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 (*auto simp: AbstFormP.simps [of s v i Q_Ex x _ k] AbstFormP.simps [of s1 v SUCC i x x' k1]*
intro!: SeqAbstFormP_Ex [THEN cut3] Ord_IN_Ord[OF Mem_SUCC_I2[OF Refl], of _ i])
qed

corollary *AbstTermP_Zero*: $\{OrdP\ t\} \vdash AbstTermP\ \llbracket Var\ v \rrbracket\ t\ Zero\ Zero$

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: AbstTermP.simps [of s _ _ _ k] intro: SeqStTermP_Zero [THEN cut1]*)
qed

corollary *AbstTermP_Var_same*: $\{VarP\ v,\ OrdP\ t\} \vdash AbstTermP\ v\ t\ v\ (Q_Ind\ t)$

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: AbstTermP.simps [of s _ _ _ k] intro: SeqStTermP_Var_same [THEN cut1]*)
qed

corollary $AbstTermP_Var_diff: \{VarP\ v, VarP\ w, Neg\ (v\ EQ\ w), OrdP\ t\} \vdash AbstTermP\ v\ t\ w\ w$
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: AbstTermP.simps [of s _ _ _ k] intro: SeqStTermP_Var_diff [THEN cut3]*)
qed

theorem $AbstTermP_Eats:$
 $\{AbstTermP\ v\ i\ t1\ u1, AbstTermP\ v\ i\ t2\ u2\} \vdash AbstTermP\ v\ i\ (Q_Eats\ t1\ t2)\ (Q_Eats\ u1\ u2)$
proof –
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]*
 $simp: AbstTermP.simps [of s _ _ _ (Q_Eats\ u1\ u2)\ k]$
 $AbstTermP.simps [of s1\ v\ i\ t1\ u1\ k1]$
 $AbstTermP.simps [of s2\ v\ i\ t2\ u2\ k2]$)
qed

corollary $AbstTermP_Ind: \{VarP\ v, IndP\ w, OrdP\ t\} \vdash AbstTermP\ v\ t\ w\ w$
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: AbstTermP.simps [of s _ _ _ k]*
 $intro: SeqStTermP_Ind [THEN cut2]$)
qed

lemma $ORD_OF_EQ_diff: x \neq y \implies \{ORD_OF\ x\ EQ\ ORD_OF\ y\} \vdash Fls$
proof (*induct x arbitrary: y*)
case ($Suc\ x$)
then show ?case **using** $SUCC_inject_E$
by (*cases y*) (*auto simp: gr0_conv_Suc Eats_EQ_Zero_E SUCC_def*)
qed (*auto simp: gr0_conv_Suc SUCC_def*)

lemma $quot_Var_EQ_diff: i \neq x \implies \{\llbracket Var\ i \rrbracket\ EQ\ \llbracket Var\ x \rrbracket\} \vdash Fls$
by (*auto simp: quot_Var ORD_OF_EQ_diff*)

lemma $AbstTermP_dbtm: \{\} \vdash AbstTermP\ \llbracket Var\ i \rrbracket\ (ORD_OF\ n)\ (quot_dbtm\ u)\ (quot_dbtm\ (abst_dbtm\ i\ n\ u))$
proof (*induct u rule: dbtm.induct*)
case ($DBVar\ x$)
then show ?case
by (*auto simp: quot_Var[symmetric] quot_Var_EQ_diff*
 $intro!: AbstTermP_Var_same [THEN cut2]$ $AbstTermP_Var_diff [THEN cut4]$ $TermP_Zero$)
qed (*auto intro!: AbstTermP_Zero [THEN cut1]* $AbstTermP_Eats [THEN cut2]$ $AbstTermP_Ind [THEN cut3]$ $IndP_Q_Ind$)

lemma *AbstFormP_dbfm*: $\{\} \vdash \text{AbstFormP} \llbracket \text{Var } i \rrbracket \text{ (ORD_OF } n) \text{ (quot_dbfm } db) \text{ (quot_dbfm (abst_dbfm } i \text{ } n \text{ } db))}$

by (*induction db arbitrary: n rule: dbfm.induct*)
(auto intro!: AbstTermP_dbtm AbstFormP_Mem[THEN cut2] AbstFormP_Eq[THEN cut2]
AbstFormP_Disj[THEN cut2] AbstFormP_Neg[THEN cut1] AbstFormP_Ex[THEN cut1]
dest: meta_spec[of _ Suc _])

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

lemma *SubstTermP_trivial_dbtm*:

atom i # u \implies $\{\} \vdash \text{SubstTermP} \llbracket \text{Var } i \rrbracket \text{ Zero (quot_dbtm } u) \text{ (quot_dbtm } u)$

proof (*induct u rule: dbtm.induct*)

case (*DBVar x*)

then show *?case*

by (*auto simp: quot_Var[symmetric] quot_Var_EQ_diff*
intro!: SubstTermP_Var_same[THEN cut2] SubstTermP_Var_diff[THEN cut4] TermP_Zero)

qed (*auto intro!: SubstTermP_Zero[THEN cut1] SubstTermP_Eats[THEN cut2] SubstTermP_Ind[THEN cut3]*

TermP_Zero IndP_Q_Ind)

lemma *SubstTermP_dbtm*: *wf_dbtm t \implies*

$\{\} \vdash \text{SubstTermP} \llbracket \text{Var } i \rrbracket \text{ (quot_dbtm } t) \text{ (quot_dbtm } u) \text{ (quot_dbtm (subst_dbtm } t \text{ } i \text{ } u))}$

proof (*induct u rule: dbtm.induct*)

case (*DBVar x*)

then show *?case*

apply (*auto simp: quot_Var[symmetric]*
intro!: SubstTermP_Var_same[THEN cut2] SubstTermP_Var_diff[THEN cut4] TermP_quot_dbtm)

apply (*auto simp: quot_Var ORD_OF_EQ_diff*)

done

qed (*auto intro!: SubstTermP_Zero[THEN cut1] SubstTermP_Ind[THEN cut3] SubstTermP_Eats[THEN cut2]*

TermP_quot_dbtm IndP_Q_Ind)

lemma *SubstFormP_trivial_dbfm*:

fixes *X :: fm*

assumes *atom i # db*

shows $\{\} \vdash \text{SubstFormP} \llbracket \text{Var } i \rrbracket \text{ Zero (quot_dbfm } db) \text{ (quot_dbfm } db)$

using *assms*

by (*induct db rule: dbfm.induct*)

(auto intro!: SubstFormP_Ex[THEN cut1] SubstFormP_Neg[THEN cut1] SubstFormP_Disj[THEN cut2]

SubstFormP_Eq[THEN cut2] SubstFormP_Mem[THEN cut2] SubstTermP_trivial_dbtm)+

lemma *SubstFormP_dbfm*:

assumes *wf_dbtm t*

shows $\{\} \vdash \text{SubstFormP} \llbracket \text{Var } i \rrbracket \text{ (quot_dbtm } t) \text{ (quot_dbfm } db) \text{ (quot_dbfm (subst_dbfm } t \text{ } i \text{ } db))}$

by (*induct db rule: dbfm.induct*)

(auto intro!: SubstTermP_dbtm assms SubstFormP_Ex[THEN cut1] SubstFormP_Neg[THEN cut1]
SubstFormP_Disj[THEN cut2] SubstFormP_Eq[THEN cut2] SubstFormP_Mem[THEN cut2])+

lemmas *SubstFormP_trivial = SubstFormP_trivial_dbfm*[**where** *db=trans_fm [] A for A,*
simplified, folded quot_tm_def quot_fm_def quot_subst_eq]

lemmas *SubstFormP = SubstFormP_dbfm*[*OF wf_dbtm_trans_tm, where db=trans_fm [] A for A,*
simplified, folded quot_tm_def quot_fm_def quot_subst_eq]

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

lemma *AtomicP_Mem*:
 {*TermP x, TermP y*} ⊢ *AtomicP (Q_Mem x y)*
proof –
obtain *t::name and u::name*
where *atom t* # (*x, y*) *atom u* # (*t, x, y*)
by (*metis obtain_fresh*)
thus *?thesis*
apply (*simp add: AtomicP.simps* [*of t u*])
apply (*rule Ex_I* [**where** *x = x*], *simp*)
apply (*rule Ex_I* [**where** *x = y*], *simp*)
apply (*auto intro: Disj_I2*)
done
qed

lemma *AtomicP_Eq*:
 {*TermP x, TermP y*} ⊢ *AtomicP (Q_Eq x y)*
proof –
obtain *t::name and u::name*
where *atom t* # (*x, y*) *atom u* # (*t, x, y*)
by (*metis obtain_fresh*)
thus *?thesis*
apply (*simp add: AtomicP.simps* [*of t u*])
apply (*rule Ex_I* [**where** *x = x*], *simp*)
apply (*rule Ex_I* [**where** *x = y*], *simp*)
apply (*auto intro: Disj_I1*)
done
qed

lemma *SeqFormP_Mem*:
assumes *atom s* # (*k,x,y*) *atom k* # (*x,y*)
shows {*TermP x, TermP y*} ⊢ *Ex k (Ex s (SeqFormP (Var s) (Var k) (Q_Mem x y)))*
proof –
let *?vs = (x,y,s,k)*
obtain *l::name and sl::name and m::name and n::name and sm::name and sn::name*
where
atom l # (*?vs,sl,m,n,sm,sn*)
atom sl # (*?vs,m,n,sm,sn*)
atom m # (*?vs,n,sm,sn*) *atom n* # (*?vs,sm,sn*)
atom sm # (*?vs,sn*)
atom sn # (*?vs*)
by (*metis obtain_fresh*)
with *assms show ?thesis*
apply (*auto simp: SeqFormP.simps*[*of l Var s ___ sl m n sm sn*])
apply (*rule Ex_I* [**where** *x = Zero*], *simp*)
apply (*rule Ex_I* [**where** *x = Eats Zero (HPair Zero (Q_Mem x y))*], *auto intro!: Mem_SUCC_EH*)
apply (*rule Ex_I* [**where** *x = Q_Mem x y*], *auto intro!: Mem_Eats_I2 HPair_cong Disj_I1*)
AtomicP_Mem[*THEN cut2*])
done
qed

lemma *SeqFormP_Eq*:
assumes *atom s* # (*k,x,y*) *atom k* # (*x,y*)
shows {*TermP x, TermP y*} ⊢ *Ex k (Ex s (SeqFormP (Var s) (Var k) (Q_Eq x y)))*
proof –
let *?vs = (x,y,s,k)*


```

obtain l::name and sl::name and m::name and n::name and sm::name and sn::name
where
  atom l # (?vs,sl,m,n,sm,sn)
  atom sl # (?vs,m,n,sm,sn)
  atom m # (?vs,n,sm,sn) atom n # (?vs,sm,sn)
  atom sm # (?vs,sn)
  atom sn # (?vs)
by (metis obtain_fresh)
with assms show ?thesis
apply (auto simp: SeqFormP.simps[of l Var s _ _ sl m n sm sn])
apply (rule Ex_I [where x = Zero], simp)
apply (rule Ex_I [where x = Eats Zero (HPair Zero (Q_Eq x y))], auto intro!: Mem_SUCC_EH)
apply (rule Ex_I [where x = Q_Eq x y], auto intro!: Mem_Eats_I2 HPair_cong Disj_I1 AtomicP_Eq[THEN cut2])
done
qed

```

```

lemma FormP_Mem:
  {TermP x, TermP y} ⊢ FormP (Q_Mem x y)
proof –
  obtain s::name and k::name
  where atom s # (x, y) atom k # (s, x, y)
  by (metis obtain_fresh)
  thus ?thesis
  by (auto simp add: FormP.simps [of k s] intro!: SeqFormP_Mem)
qed

```

```

lemma FormP_Eq:
  {TermP x, TermP y} ⊢ FormP (Q_Eq x y)
proof –
  obtain s::name and k::name
  where atom s # (x, y) atom k # (s, x, y)
  by (metis obtain_fresh)
  thus ?thesis
  by (auto simp add: FormP.simps [of k s] intro!: SeqFormP_Eq)
qed

```

7.14.6 MakeForm

```

lemma MakeFormP_Neg: {} ⊢ MakeFormP (Q_Neg x) x y
proof –
  obtain a::name and b::name
  where atom a # (x, y) atom b # (a, x, y) by (metis obtain_fresh)
  then show ?thesis
  by (auto simp: MakeFormP.simps[of a _ _ b] intro: Disj_I2[OF Disj_I1])
qed

```

```

lemma MakeFormP_Disj: {} ⊢ MakeFormP (Q_Disj x y) x y
proof –
  obtain a::name and b::name
  where atom a # (x, y) atom b # (a, x, y) by (metis obtain_fresh)
  then show ?thesis
  by (auto simp: MakeFormP.simps[of a _ _ b] intro: Disj_I1)
qed

```

```

lemma MakeFormP_Ex: {AbstFormP v Zero t x} ⊢ MakeFormP (Q_Ex x) t y
proof –
  obtain a::name and b::name

```

where $atom\ a \# (v, x, t, y)$ $atom\ b \# (a, v, x, t, y)$ **by** (*metis obtain_fresh*)
then show *?thesis*
by (*subst MakeFormP.simps*[of a $---$ b])
(force intro!: *Disj_I2*[*OF Disj_I2*] *intro:* *Ex_I*[of $---$ v] *Ex_I*[of $---$ x])*+*
qed

7.14.7 Negation

lemma *SeqFormP_Neg:*

assumes $atom\ s \# (k, s1, k1, x)$ $atom\ k \# (s1, k1, x)$

shows $\{SeqFormP\ s1\ k1\ x\} \vdash Ex\ k\ (Ex\ s\ (SeqFormP\ (Var\ s)\ (Var\ k)\ (Q_Neg\ x)))$

theorem *FormP_Neg:* $\{FormP\ x\} \vdash FormP\ (Q_Neg\ x)$

proof $-$

obtain $k1::name$ **and** $s1::name$ **and** $k::name$ **and** $s::name$

where $atom\ s1 \# x$ $atom\ k1 \# (x, s1)$

$atom\ s \# (x, k1, s1)$ $atom\ k \# (x, s, k1, s1)$

by (*metis obtain_fresh*)

thus *?thesis*

by (*force simp:* *FormP.simps* [of $k\ s\ Q_Neg\ x$] *FormP.simps* [of $k1\ s1\ x$]
intro: *SeqFormP_Neg* [*THEN cut1*])

qed

7.14.8 Disjunction

lemma *SeqFormP_Disj:*

assumes $atom\ s \# (k, s1, s2, k1, k2, x, y)$ $atom\ k \# (s1, s2, k1, k2, x, y)$

shows $\{SeqFormP\ s1\ k1\ x, SeqFormP\ s2\ k2\ y\}$

$\vdash Ex\ k\ (Ex\ s\ (SeqFormP\ (Var\ s)\ (Var\ k)\ (Q_Disj\ x\ y)))$

theorem *FormP_Disj:*

$\{FormP\ x, FormP\ y\} \vdash FormP\ (Q_Disj\ x\ y)$

proof $-$

obtain $k1::name$ **and** $s1::name$ **and** $k2::name$ **and** $s2::name$ **and** $k::name$ **and** $s::name$

where $atom\ s1 \# (x, y)$ $atom\ k1 \# (x, y, s1)$

$atom\ s2 \# (x, y, k1, s1)$ $atom\ k2 \# (x, y, s2, k1, s1)$

$atom\ s \# (x, y, k2, s2, k1, s1)$ $atom\ k \# (x, y, s, k2, s2, k1, s1)$

by (*metis obtain_fresh*)

thus *?thesis*

by (*force simp:* *FormP.simps* [of $k\ s\ Q_Disj\ x\ y$]
FormP.simps [of $k1\ s1\ x$]
FormP.simps [of $k2\ s2\ y$]
intro: *SeqFormP_Disj* [*THEN cut2*])

qed

7.14.9 Existential

lemma *SeqFormP_Ex:*

assumes $atom\ s \# (k, s1, k1, x, y, v)$ $atom\ k \# (s1, k1, x, y, v)$

shows $\{SeqFormP\ s1\ k1\ x, AbstFormP\ v\ Zero\ x\ y, VarP\ v\} \vdash Ex\ k\ (Ex\ s\ (SeqFormP\ (Var\ s)\ (Var\ k)\ (Q_Ex\ y)))$

proof $-$

let $?vs = (s1, s, k1, k, x, y, v)$

obtain $km::name$ **and** $kn::name$ **and** $j::name$ **and** $k'::name$

and $l::name$ **and** $sl::name$ **and** $m::name$ **and** $n::name$

and $sm::name$ **and** $sn::name$

where $atoms2:$ $atom\ km \# (kn, j, k', l, s1, s, k1, k, x, y, v, sl, m, n, sm, sn)$

$atom\ kn \# (j, k', l, s1, s, k1, k, x, y, v, sl, m, n, sm, sn)$

$atom\ j \# (k', l, s1, s, k1, k, x, y, v, sl, m, n, sm, sn)$

and $atoms:$ $atom\ k' \# (l, s1, s, k1, k, x, y, v, sl, m, n, sm, sn)$

$atom\ l \# (s1, s, k1, k, x, y, v, sl, m, n, sm, sn)$

```

    atom sl # (s1,s,k1,k,x,y,v,m,n,sm,sn)
    atom m  # (s1,s,k1,k,x,y,v,n,sm,sn)
    atom n  # (s1,s,k1,k,x,y,v,sm,sn)
    atom sm # (s1,s,k1,k,x,y,v,sn)
    atom sn # (s1,s,k1,k,x,y,v)
  by (metis obtain_fresh)
let ?hyp = {RestrictedP s1 (SUCC k1) (Var s), OrdP k1, SeqFormP s1 k1 x,AbstFormP v Zero x y,
VarP v}
show ?thesis
using assms atoms
apply (auto simp: SeqFormP.simps [of l Var s __ sl m n sm sn])
apply (rule cut_same [where A=OrdP k1])
  apply (metis SeqFormP_imp_OrdP thin2)
apply (rule cut_same [OF exists_RestrictedP [of s s1 SUCC k1]])
  apply (rule AssumeH Ex_EH Conj_EH | simp)+
  apply (rule Ex_I [where x=(SUCC k1), simp])
  apply (rule Ex_I [where x=Eats (Var s) (HPair (SUCC k1) (Q_Ex y)), simp])
  apply (rule Conj_I)
  apply (blast intro: RestrictedP_LstSeqP_Eats [THEN cut2] SeqFormP_imp_LstSeqP [THEN cut1])
proof (rule All2_SUCC_I, simp_all)
show ?hyp ⊢ SyntaxN.Ex sn
  (HPair (SUCC k1) (Var sn) IN
   Eats (Var s) (HPair (SUCC k1) (Q_Ex y)) AND
   (AtomicP (Var sn) OR
    SyntaxN.Ex m
    (SyntaxN.Ex l
     (SyntaxN.Ex sm
      (SyntaxN.Ex sl
       (Var m IN SUCC k1 AND
        Var l IN SUCC k1 AND
        HPair (Var m) (Var sm) IN
        Eats (Var s) (HPair (SUCC k1) (Q_Ex y)) AND
        HPair (Var l) (Var sl) IN
        Eats (Var s) (HPair (SUCC k1) (Q_Ex y)) AND
        MakeFormP (Var sn) (Var sm) (Var sl))))))))
— verifying the final values
  apply (rule Ex_I [where x=Q_Ex y])
  using assms atoms apply simp
  apply (rule Conj_I, metis Mem_Eats_I2 Refl)
  apply (rule Disj_I2)
  apply (rule Ex_I [where x=k1], simp)
  apply (rule Ex_I [where x=k1], simp)
  apply (rule Ex_I [where x=x], simp)
  apply (rule Ex_I [where x=x], simp)
  apply (rule Conj_I [OF Mem_SUCC_Refl])+
  apply safe
  apply (blast intro: Disj_I2 Mem_Eats_I1 RestrictedP_Mem [THEN cut3] Mem_SUCC_Refl
    SeqFormP_imp_LstSeqP [THEN cut1] LstSeqP_imp_Mem)
  apply (blast intro: Disj_I2 Mem_Eats_I1 RestrictedP_Mem [THEN cut3] Mem_SUCC_Refl
    SeqFormP_imp_LstSeqP [THEN cut1] LstSeqP_imp_Mem)
  apply (rule MakeFormP_Ex[THEN cut1, of _ v])
  apply blast
done
next
show ?hyp ⊢ All2 n (SUCC k1)
  (SyntaxN.Ex sn
   (HPair (Var n) (Var sn) IN
    Eats (Var s) (HPair (SUCC k1) (Q_Ex y)) AND

```

```

(AtomicP (Var sn) OR
SyntaxN.Ex m
(SyntaxN.Ex l
(SyntaxN.Ex sm
(SyntaxN.Ex sl
(Var m IN Var n AND
Var l IN Var n AND
HPair (Var m) (Var sm) IN
Eats (Var s) (HPair (SUCC k1) (Q_Ex y)) AND
HPair (Var l) (Var sl) IN
Eats (Var s) (HPair (SUCC k1) (Q_Ex y)) AND
MakeFormP (Var sn) (Var sm) (Var sl))))))
apply (rule All_I Imp_I)+
using assms atoms apply simp_all
— ... the sequence buildup via s1
apply (simp add: SeqFormP.simps [of l s1 __ sl m n sm sn])
apply (rule AssumeH Ex_EH Conj_EH)+
apply (rule All2_E [THEN rotate2], auto del: Disj_EH)
apply (rule Ex_I [where x=Var sn], simp)
apply (rule Conj_I)
apply (blast intro: Mem_Eats_I1 [OF RestrictedP_Mem [THEN cut3]] del: Disj_EH)
apply (rule AssumeH Disj_IE1H Ex_EH Conj_EH Conj_I)+
apply (rule Ex_I [where x=Var m], simp)
apply (rule Ex_I [where x=Var l], simp)
apply (rule Ex_I [where x=Var sm], simp)
apply (rule Ex_I [where x=Var sl], simp)
apply auto
apply (rule Mem_Eats_I1 [OF RestrictedP_Mem [THEN cut3]] AssumeH OrdP_Trans [OF
OrdP_SUCC_I])+
done
qed

```

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

proof —

```

obtain k1::name and s1::name and k::name and s::name
where atom s1 # (i,t,x) atom k1 # (i,t,x,s1) atom s # (i,t,x,k1,s1) atom k # (i,t,x,s,k1,s1)
by (metis obtain_fresh)
thus ?thesis
by (auto simp: FormP.simps [of k s Q_Ex x] FormP.simps [of k1 s1 t]
intro!: SeqFormP_Ex [THEN cut3])

```

qed

lemma *FormP_quot_dbfm*:

```

fixes A :: dbfm
shows wf_dbfm A ⇒ {} ⊢ FormP (quot_dbfm A)
by (induct A rule: wf_dbfm.induct)
(auto simp: intro!: FormP_Mem[THEN cut2] FormP_Eq[THEN cut2] Ex_I
FormP_Neg[THEN cut1] FormP_Disj[THEN cut2] FormP_Ex[THEN cut2]
TermP_quot_dbtm AbstFormP_dbfm[where n=0, simplified])

```

lemma *FormP_quot*:

```

fixes A :: fm
shows {} ⊢ FormP «A»
unfolding quot_fm_def
by (rule FormP_quot_dbfm, rule wf_dbfm_trans_fm)

```

```

lemma Pfp_I:
  assumes {}  $\vdash$  PrfP S K A
  shows {}  $\vdash$  Pfp A
proof –
  obtain s::name and k::name where atom s  $\#$  (k,A,S,K) atom k  $\#$  (A,S,K) by (metis obtain_fresh)
  with assms show ?thesis
  apply (subst Pfp.simps[of s k]; simp)
  apply (rule Ex_I[of _ _ _ K], auto, rule Ex_I[of _ _ _ S], auto)
  done
qed

```

```

lemmas Pfp_Single_I = Pfp_I[of Eats Zero (HPair Zero «A») Zero for A]

```

```

lemma Pfp_extra: {}  $\vdash$  Pfp «extra_axiom»
proof –
  obtain l::name and sl::name and m::name and n::name and sm::name and sn::name
  where atoms:
    atom l  $\#$  (sl,m,n,sm,sn)
    atom sl  $\#$  (m,n,sm,sn)
    atom m  $\#$  (n,sm,sn)
    atom n  $\#$  (sm,sn)
    atom sm  $\#$  sn
  by (metis obtain_fresh)
  with Extra show ?thesis
  apply (intro Pfp_Single_I[of extra_axiom])
  apply (subst PrfP.simps[of l _ sl m n sm sn]; auto?)
  apply (rule Ex_I[of _ _ _ «extra_axiom»]; auto?)
  apply (rule Mem_SUCC_E[OF Mem_Zero_E])
  apply (rule Mem_Eats_I2)
  apply (rule HPair_cong[OF Assume Refl])
  apply (auto simp: AxiomP_def intro!: Disj_I1)
  done
qed

```

```

lemma SentP_I:
  assumes A  $\in$  boolean_axioms
  shows {}  $\vdash$  SentP «A»
proof –
  obtain x y z :: name where atom z  $\#$  (x,y) atom y  $\#$  x by (metis obtain_fresh)
  with assms show ?thesis
  apply (subst SentP.simps[of x y z]; simp)
  subgoal
  proof (erule boolean_axioms.cases, goal_cases Ident DisjI1 DisjCont DisjAssoc DisjConj)
  case (Ident A)
  then show ?thesis
  by (intro Ex_I[of _ _ _ «A»]; simp)+
  (auto simp: FormP_quot[THEN thin0] quot_simps intro!: Disj_I1)
  next
  case (DisjI1 A B)
  then show ?thesis
  by (intro Ex_I[of _ _ _ «A»]; simp, (intro Ex_I[of _ _ _ «B»]; simp)?) +
  (auto simp: FormP_quot[THEN thin0] quot_simps intro!: Disj_I2[OF Disj_I1])
  next
  case (DisjCont A)
  then show ?thesis
  by (intro Ex_I[of _ _ _ «A»]; simp)+
  (auto simp: FormP_quot[THEN thin0] quot_simps intro!: Disj_I2[OF Disj_I2[OF Disj_I1]])
  next

```

```

    case (DisjAssoc A B C)
  then show ?thesis
  by (intro Ex_I[of _ _ _ «A»]; simp, intro Ex_I[of _ _ _ «B»]; simp, intro Ex_I[of _ _ _ «C»];
simp)+
  (auto simp: FormP_quot[THEN thin0] quot_simps intro!: Disj_I2[OF Disj_I2[OF Disj_I2[OF
Disj_I1]]])
  next
    case (DisjConj A B C)
  then show ?thesis
  by (intro Ex_I[of _ _ _ «A»]; simp, intro Ex_I[of _ _ _ «B»]; simp, intro Ex_I[of _ _ _ «C»];
simp)+
  (auto simp: FormP_quot[THEN thin0] quot_simps intro!: Disj_I2[OF Disj_I2[OF Disj_I2[OF
Disj_I2]]])
  qed
done
qed

```

```

lemma SentP_subst [simp]: (SentP A)(j::=w) = SentP (subst j w A)
proof -
  obtain x y z ::name where atom x # (y,z,j,w,A) atom y # (z,j,w,A) atom z # (j,w,A)
  by (metis obtain_fresh)
  thus ?thesis
  by (auto simp: SentP_simps [of x y z])
qed

```

```

theorem proved_imp_proved_PfP:
  assumes {} ⊢ α
  shows {} ⊢ PfP «α»
  using assms
proof (induct {} :: fm set α rule: hfthm.induct)
  case (Hyp A)
  then show ?case
  by auto
next
  case Extra
  then show ?case by (simp add: PfP_extra)
next
  case (Bool A)
  obtain l::name and sl::name and m::name and n::name and
    sm::name and sn::name and x::name and y::name and z::name
  where atoms:
    atom l # (x,y,z,sl,m,n,sm,sn)
    atom sl # (x,y,z,m,n,sm,sn)
    atom m # (x,y,z,n,sm,sn)
    atom n # (x,y,z,sm,sn)
    atom sm # (x,y,z,sn)
    atom sn # (x,y,z)
    atom z # (x,y)
    atom y # x
  by (metis obtain_fresh)
  with Bool show ?case
  apply (intro PfP_Single_I[of A])
  apply (subst PrfP_simps[of l _ sl m n sm sn]; auto?)
  apply (rule Ex_I[of _ _ _ «A»]; auto?)
  apply (rule Mem_SUCC_E[OF Mem_Zero_E])
  apply (rule Mem_Eats_I2)
  apply (rule HPair_cong[OF Assume Refl])
  apply (rule Disj_I1)

```

```

apply (unfold AxiomP_def; simp)
apply (rule Disj_I2[OF Disj_I1])
apply (auto elim!: SentP_I[THEN thin0])
done
next
case (Eq A)
obtain l::name and sl::name and m::name and n::name and sm::name and sn::name and x::name
and y::name and z::name
  where atoms:
    atom l # (x,y,z,sl,m,n,sm,sn)
    atom sl # (x,y,z,m,n,sm,sn)
    atom m # (x,y,z,n,sm,sn)
    atom n # (x,y,z,sm,sn)
    atom sm # (x,y,z,sn)
    atom sn # (x,y,z)
    atom z # (x,y)
    atom y # x
  by (metis obtain_fresh)
with Eq show ?case
apply (intro PfP_Single_I[of A])
apply (subst PrfP_simps[of l _ sl m n sm sn]; auto?)
apply (rule Ex_I[of _ _ _ «A»]; auto?)
apply (rule Mem_SUCC_E[OF Mem_Zero_E])
apply (rule Mem_Eats_I2)
apply (rule HPair_cong[OF Assume Refl])
apply (rule Disj_I1)
apply (unfold AxiomP_def; simp)
apply (rule Disj_I2[OF Disj_I2[OF Disj_I1]])
apply (auto simp: equality_axioms_def
  intro: Disj_I1 Disj_I2[OF Disj_I1] Disj_I2[OF Disj_I2[OF Disj_I1]] Disj_I2[OF Disj_I2[OF
Disj_I2]])
done
next
case (Spec A)
obtain l::name and sl::name and m::name and n::name and
  sm::name and sn::name and x::name and y::name and z::name
  where atoms:
    atom l # (x,y,z,sl,m,n,sm,sn)
    atom sl # (x,y,z,m,n,sm,sn)
    atom m # (x,y,z,n,sm,sn)
    atom n # (x,y,z,sm,sn)
    atom sm # (x,y,z,sn)
    atom sn # (x,y,z)
    atom z # (x,y)
    atom y # x
  by (metis obtain_fresh)
let ?vs = (x,y,z,l,sl,m,n,sm,sn)
from Spec atoms show ?case
apply (intro PfP_Single_I[of A])
apply (subst PrfP_simps[of l _ sl m n sm sn]; auto?)
apply (rule Ex_I[of _ _ _ «A»]; auto?)
apply (rule Mem_SUCC_E[OF Mem_Zero_E])
apply (rule Mem_Eats_I2)
apply (rule HPair_cong[OF Assume Refl])
apply (rule Disj_I1)
apply (unfold AxiomP_def; simp)
apply (rule Disj_I2[OF Disj_I2[OF Disj_I2[OF Disj_I2[OF Disj_I1]]]])
subgoal premises prems

```

```

using prems proof (cases A rule: special_axioms.cases)
case (I X i t)
let ?vs' = (?vs, X, i, t)
  obtain AA XX ii tt res :: name
  where atoms:
    atom AA # (?vs', res, tt, ii, XX)
    atom XX # (?vs', res, tt, ii)
    atom ii # (?vs', res, tt)
    atom tt # (?vs', res)
    atom res # ?vs'
  by (metis obtain_fresh)
with I show ?thesis
  apply (subst Special_axP.simps[of ii _ res tt AA XX]; simp?)
  apply (rule Ex_I[of _ _ _ «Var i»]; auto?)
  apply (rule Ex_I[of _ _ _ «X»]; auto?)
  apply (rule Ex_I[of _ _ _ quot_dbfm (trans_fm [i] X)]; auto?)
  apply (rule Ex_I[of _ _ _ «t»]; auto?)
  apply (rule Ex_I[of _ _ _ «X(i::=t)»]; auto?)
    apply (auto simp: TermP_quot[THEN thin0] FormP_quot[THEN thin0]
      SubstFormP[THEN thin0] AbstFormP[THEN thin0]
      quot_Ex quot_Disj quot_Neg vquot_fm_def)
  done
qed
done
next
case (HF A)
obtain l::name and sl::name and m::name and n::name and
  sm::name and sn::name and x::name and y::name and z::name
  where atoms:
    atom l # (x,y,z,sl,m,n,sm,sn)
    atom sl # (x,y,z,m,n,sm,sn)
    atom m # (x,y,z,n,sm,sn)
    atom n # (x,y,z,sm,sn)
    atom sm # (x,y,z,sn)
    atom sn # (x,y,z)
    atom z # (x,y)
    atom y # x
  by (metis obtain_fresh)
with HF show ?case
  apply (intro PfP_Single_I[of A])
  apply (subst PrfP.simps[of l _ sl m n sm sn]; auto?)
  apply (rule Ex_I[of _ _ _ «A»]; auto?)
  apply (rule Mem_SUCC_E[OF Mem_Zero_E])
  apply (rule Mem_Eats_I2)
  apply (rule HPair_cong[OF Assume Refl])
  apply (rule Disj_I1)
  apply (unfold AxiomP_def; simp)
  apply (rule Disj_I2[OF Disj_I2[OF Disj_I2[OF Disj_I1]]])
  apply (auto simp: HF_axioms_def intro: Disj_I1 Disj_I2)
  done
next
case (Ind A)
obtain l::name and sl::name and m::name and n::name and
  sm::name and sn::name and x::name and y::name and z::name
  where atoms:
    atom l # (x,y,z,sl,m,n,sm,sn)
    atom sl # (x,y,z,m,n,sm,sn)
    atom m # (x,y,z,n,sm,sn)

```



```

    atom n # (x,y,z,sm,sn)
    atom sm # (x,y,z,sn)
    atom sn # (x,y,z)
    atom z # (x,y)
    atom y # x
  by (metis obtain_fresh)
let ?vs = (x,y,z,l,sl,m,n,sm,sn)
from Ind atoms show ?case
  apply (intro PfP_Single_I[of A])
  apply (subst PrfP_simps[of l _ sl m n sm sn]; auto?)
  apply (rule Ex_I[of _ _ _ «A»]; auto?)
  apply (rule Mem_SUCC_E[OF Mem_Zero_E])
  apply (rule Mem_Eats_I2)
  apply (rule HPair_cong[OF Assume Refl])
  apply (rule Disj_I1)
  apply (unfold AxiomP_def; simp)
  apply (rule Disj_I2[OF Disj_I2[OF Disj_I2[OF Disj_I2[OF Disj_I2]]]])
subgoal premises prems
using prems proof (cases A rule: induction_axioms.cases)
  case (ind j i X)
  let ?vs' = (?vs, X, i, j)
  obtain ax allw allw xevw xw x0 xa w v :: name
  where atoms:
    atom ax # (?vs', v, w, xa, x0, xw, xevw, allw, allwv)
    atom allwv # (?vs', v, w, xa, x0, xw, xevw, allw)
    atom allw # (?vs', v, w, xa, x0, xw, xevw)
    atom xevw # (?vs', v, w, xa, x0, xw)
    atom xw # (?vs', v, w, xa, x0)
    atom x0 # (?vs', v, w, xa)
    atom xa # (?vs', v, w)
    atom w # (?vs', v)
    atom v # (?vs')
  by (metis obtain_fresh)
with ind(2) show ?thesis
  unfolding ind(1)
  apply (subst Induction_axP_simps[of ax _ allw allw xevw xw x0 xa w v])
  apply simp_all
  apply (rule Ex_I[of _ _ _ «Var i»]; auto?)
  apply (rule Ex_I[of _ _ _ «Var j»]; auto?)
  apply (rule Ex_I[of _ _ _ «X»]; auto?)
  apply (rule Ex_I[of _ _ _ «X(i::=Zero)»]; auto?)
  apply (rule Ex_I[of _ _ _ «X(i::=Var j)»]; auto?)
  apply (rule Ex_I[of _ _ _ «X(i::=Eats (Var i) (Var j))»]; auto?)
  apply (rule Ex_I[of _ _ _ quot_dbfm (trans_fm [j] (X IMP (X(i::= Var j) IMP X(i::= Eats(Var
i)(Var j))))]); auto?)
  apply (rule Ex_I[of _ _ _ Q_All (quot_dbfm (trans_fm [j,i] (X IMP (X(i::= Var j) IMP X(i::=
Eats(Var i)(Var j)))))); auto?)
  apply (rule Ex_I[of _ _ _ quot_dbfm (trans_fm [i] X)]; auto?)
  subgoal
  apply (rule thin0)
  apply (rule OrdNotEqP_I)
  apply (auto simp: quot_Var ORD_OF_EQ_diff intro!: OrdP_SUCC_I0[THEN cut1])
  done
  subgoal
  by (auto simp: VarNonOccFormP_simps FormP_quot[THEN thin0] SubstFormP_trivial[THEN
thin0])
  subgoal
  by (rule SubstFormP_Zero[THEN thin0])

```

```

subgoal
  by (rule SubstFormP[THEN thin0])
subgoal
  unfolding quot_Eats[symmetric] One_nat_def[symmetric]
  by (rule SubstFormP[THEN thin0])
subgoal
  unfolding quot_simps[symmetric] quot_dbfm_simps[symmetric] trans_fm_simps[symmetric]
  by (rule AbstFormP[THEN thin0])
subgoal
  by (auto simp only: quot_simps[symmetric] quot_dbfm_simps[symmetric] trans_fm_simps[symmetric]
      fresh_Cons fresh_Nil fresh_Pair trans_fm_simps(5)[symmetric, of j []]
      quot_fm_def[symmetric] intro!: AbstFormP[THEN thin0])
subgoal
  unfolding quot_simps[symmetric] quot_dbfm_simps[symmetric] trans_fm_simps[symmetric]
  by (rule AbstFormP[THEN thin0])
subgoal
  by (auto simp: quot_simps trans_fm_simps(5)[of j [i]]
      fresh_Cons fresh_Pair)
done
qed
done
next
case (MP H A B H')
then show ?case
  by (auto elim!: Pfp_implies_ModPon_Pfp_quot)
next
case (Exists A B i)
obtain a x y z::name
  where atoms:
    atom a # (i,x,y,z)
    atom z # (i,x,y)
    atom y # (i,x)
    atom x # i
  by (metis obtain_fresh)
with Exists show ?case
  apply (auto elim!: Pfp_inference [THEN cut3] intro!: Pfp_extra Disj_I2[OF Disj_I1])
  apply (subst ExistsP_simps[of x _ _ a y z]; (auto simp: VarNonOccFormP_simps)?)
  apply (rule Ex_I[of _ _ _ «A»]; auto?)
  apply (rule Ex_I[of _ _ _ quot_dbfm (trans_fm [i] A)]; auto?)
  apply (rule Ex_I[of _ _ _ «B»]; auto?)
  apply (rule Ex_I[of _ _ _ «Var i»]; auto?)
  apply (auto simp: FormP_quot quot_Disj quot_Neg quot_Ex SubstFormP_trivial AbstFormP)
done
qed
end

```

Chapter 8

Pseudo-Coding: Section 7 Material

```
theory Pseudo_Coding
imports II_Prelims
begin
```

8.1 General Lemmas

```
lemma Collect_disj_Un: {f i |i. P i ∨ 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»
using VarP_Var[of {} i] VarP_Var[of {} j]
by (intro OrdNotEqP_I) (auto simp: VarP_def quot_Var ORD_OF_EQ_diff dest!: Conj_E1)
```

```
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]: atom j ∈ atom ' V ↔ j ∈ 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) set shows finite S ⇒ a #* insert x S ↔ a #* x ∧ 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: $\llbracket \text{atom } i \# \text{bs}; \text{finite bs} \rrbracket \implies \text{bs} \#* i$

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 $i \notin V \text{ } i \notin p \cdot V$

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

8.2 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 $\text{make_F } \text{Vs } p \equiv \lambda i. \text{if } i \in \text{Vs} \text{ then } \text{Var } (p \cdot i) \text{ else } \text{Var } i$

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 $p :: \text{perm}$ and $\text{Vs} :: \text{name set}$ and $F :: \text{name} \Rightarrow tm$

assumes $p: \text{atom } '(p \cdot \text{Vs}) \#* \text{Vs}$

and $\text{pinv}: -p = p$

and $\text{Vs}: \text{finite } \text{Vs}$

defines $F \equiv \text{make_F } \text{Vs } p$

begin

lemma F_unfold: $F \ i = (\text{if } i \in \text{Vs} \text{ then } \text{Var } (p \cdot i) \text{ else } \text{Var } i)$

by (simp add: F_def make_F_def)

lemma finite_V [simp]: $V \subseteq \text{Vs} \implies \text{finite } V$

by (metis Vs finite_subset)

lemma perm_exits_Vs: $i \in \text{Vs} \implies (p \cdot i) \notin \text{Vs}$

by (metis Vs fresh_finite_set_at_base imageI fresh_star_def mem_permute_iff p)

lemma atom_fresh_perm: $\llbracket x \in \text{Vs}; y \in \text{Vs} \rrbracket \implies \text{atom } x \# p \cdot y$

by (metis imageI Vs p fresh_finite_set_at_base fresh_star_def mem_permute_iff fresh_at_base(2))

lemma fresh_pj: $\llbracket a \# p; j \in \text{Vs} \rrbracket \implies a \# p \cdot j$

by (metis atom_fresh_perm fresh_at_base(2) fresh_perm fresh_permute_left pinv)

lemma *fresh_Vs*: $a \# p \implies a \# Vs$
by (*metis* *Vs* *fresh_def* *fresh_perm* *fresh_permute_iff* *fresh_star_def* *p* *permute_finite* *supp_finite_set_at_base*)

lemma *fresh_pVs*: $a \# p \implies a \# p \cdot Vs$
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_eqvt*)
by (*metis* *Vs* *finite_V* *set_mp* *subset_Un_eq* *supp_of_finite_union*)

lemma *qp_insert*:
fixes *i*::*name* **and** *i'*::*name*
assumes *atom* *i* $\# p$ *atom* *i'* $\# (i,p)$
shows *quote_perm* ((*atom* *i* \equiv *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*: $\text{subst } x (F x) (\text{subst } y (F y) t) = \text{subst } y (F y) (\text{subst } x (F x) t)$
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*: $\text{ssubst } t (\text{insert } i V) F = \text{subst } i (F i) (\text{ssubst } t V F)$ (**is** *?thesis1*)
and *ssubst_insert2*: $\text{ssubst } t (\text{insert } i V) F = \text{ssubst } (\text{subst } i (F i) t) V F$ (**is** *?thesis2*)
proof –
interpret *comp_fun_commute* ($\lambda i. \text{subst } i (F i)$)
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*:
 $\text{finite } V \implies$
 $\text{ssubst } t (\text{insert } i V) F = (\text{if } i \in V \text{ then } \text{ssubst } t V F$
 $\quad \text{else } \text{subst } i (F i) (\text{ssubst } t V F))$
by (*simp* *add*: *ssubst_insert* *insert_absorb*)

lemma *ssubst_single* [*simp*]: $\text{ssubst } t \{i\} F = \text{subst } i (F i) t$
by (*simp* *add*: *ssubst_insert*)

lemma *ssubst_Var_if* [*simp*]:
assumes *finite* *V*
shows $\text{ssubst } (\text{Var } i) V F = (\text{if } i \in V \text{ then } F i \text{ else } \text{Var } i)$
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*]: $\text{finite } V \implies \text{ssubst } \text{Zero } V F = \text{Zero}$

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:
[[finite V; atom i $\#$ p \cdot V]]
 \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: [[finite V; supp t = {}]] \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 $\langle t \rangle$ V F = $\langle t \rangle$
by (simp add: ground_ssubst_eq supp_conv_fresh)

lemma ssubst_quot_fm [simp]:
fixes A::fm shows finite V \implies ssubst $\langle A \rangle$ V F = $\langle A \rangle$
by (simp add: ground_ssubst_eq supp_conv_fresh)

lemma atom_in_p_Vs: [[i \in p \cdot V; V \subseteq Vs]] \implies i \in p \cdot Vs
by (metis (full_types) True_eqvt set_mp subset_eqvt)

8.3 The Main Theorems of Section 7

```

lemma SubstTermP_vquot_dbtm:
  assumes  $w \in Vs - V$  and  $V: V \subseteq Vs$   $V' = p \cdot V$ 
    and  $s: \text{supp } dbtm \subseteq \text{atom } 'Vs$ 
  shows
     $\text{insert } (ConstP (F w)) \{ConstP (F i) \mid i. i \in V\}$ 
     $\vdash \text{SubstTermP } \llbracket Var w \rrbracket (F w)$ 
       $(\text{ssubst } (vquot\_dbtm V dbtm) V F)$ 
       $(\text{subst } w (F w) (\text{ssubst } (vquot\_dbtm (\text{insert } w V) dbtm) V F))$ 
  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 \in 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  $w \in Vs - V$  and  $V: V \subseteq Vs$   $V' = p \cdot V$ 
    and  $s: \text{supp } dbfm \subseteq \text{atom } 'Vs$ 
  shows
     $\text{insert } (ConstP (F w)) \{ConstP (F i) \mid i. i \in V\}$ 
     $\vdash \text{SubstFormP } \llbracket Var w \rrbracket (F w)$ 
       $(\text{ssubst } (vquot\_dbfm V dbfm) V F)$ 
       $(\text{subst } w (F w) (\text{ssubst } (vquot\_dbfm (\text{insert } w V) dbfm) V F))$ 
  using  $w s$ 
proof (induct dbfm rule: dbfm.induct)
  case (DBMem t u) thus ?case using  $V$ 
    by (auto intro: SubstTermP_vquot_dbtm SubstFormP_Mem [THEN cut2])
next

```

```

    case (DBEq t u) thus ?case using V
      by (auto intro: SubstTermP_vquot_dbtm SubstFormP_Eq [THEN cut2])
next
    case (DBDisj A B) thus ?case using V
      by (auto intro: SubstFormP_Disj [THEN cut2])
next
    case (DBNeg A) thus ?case using V
      by (auto intro: SubstFormP_Neg [THEN cut1])
next
    case (DBEx A) thus ?case using V
      by (auto intro: SubstFormP_Ex [THEN cut1])
qed

```

Lemmas 7.5 and 7.6

lemma *ssubst_SubstFormP*:

fixes $A::fm$

assumes $w: w \in Vs - V$ **and** $V: V \subseteq Vs$ $V' = p \cdot V$

and $s: \text{supp } A \subseteq \text{atom } ' Vs$

shows

$\text{insert } (ConstP (F w)) \{ConstP (F i) \mid i. i \in V\}$

$\vdash \text{SubstFormP } \llbracket Var w \rrbracket (F w)$

$(\text{ssubst } \llbracket A \rrbracket V V F)$

$(\text{subst } \llbracket A \rrbracket (\text{insert } w V) (\text{insert } w V) F)$

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 $\beta: \{\} \vdash PfP \llbracket \beta \rrbracket$

and $V: V \subseteq Vs$

and $s: \text{supp } \beta \subseteq \text{atom } ' Vs$

shows $\{ConstP (F i) \mid i. i \in V\} \vdash PfP (\text{ssubst } \llbracket \beta \rrbracket V V F)$

proof –

show ?thesis **using** *finite_V [OF V] V*

proof *induction*

case *empty* **thus** ?case

by (*auto simp:* β *)*

next

case (*insert i V*)

thus ?case **using** *assms*

by (*auto simp: Collect_disj_Un fresh_finite_set_at_base*

intro: PfP_implies_SubstForm_PfP thin1 ssubst_SubstFormP)

qed

qed

end

end

Chapter 9

Quotations of the Free Variables

```
theory Quote
imports Pseudo_Coding
begin
```

9.1 Sequence version of the “Special p-Function, F*”

The definition below describes a relation, not a function. This material relates to Section 8, but omits the ordering of the universe.

9.1.1 Defining the syntax: quantified body

```
nominal_function SeqQuoteP :: tm  $\Rightarrow$  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
      HPair (Var n) (HPair (Var sn) (Var sn')) IN s AND
      Var sl EQ Eats (Var sm) (Var sn) AND
      Var sl' EQ Q_Eats (Var sm') (Var sn'))))))))))))
by (auto simp: eqvt_def SeqQuoteP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)
```

```
nominal_termination (eqvt)
by lexicographic_order
```

lemma

```
shows SeqQuoteP_fresh_iff [simp]:
  a  $\#$  SeqQuoteP t u s k  $\longleftrightarrow$  a  $\#$  t  $\wedge$  a  $\#$  u  $\wedge$  a  $\#$  s  $\wedge$  a  $\#$  k (is ?thesis1)
and SeqQuoteP_sf [iff]:
  Sigma_fm (SeqQuoteP t u s k) (is ?thsf)
and SeqQuoteP_imp_OrdP:
  { SeqQuoteP t u s k }  $\vdash$  OrdP k (is ?thord)
and SeqQuoteP_imp_LstSeqP:
  { SeqQuoteP t u s k }  $\vdash$  LstSeqP s k (HPair t u) (is ?thlstseq)
proof -
```

```

obtain l::name and sl::name and sl'::name and m::name and n::name and
  sm::name and sm'::name and sn::name and sn'::name
where atoms:
  atom l # (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 ?thlstseq
by auto (auto simp: LstSeqP.simps)
qed

```

```

lemma SeqQuoteP_subst [simp]:
  (SeqQuoteP t u s k)(j::=w) =
  SeqQuoteP (subst j w t) (subst j w u) (subst j w s) (subst j w k)

```

```

proof –
obtain l::name and sl::name and sl'::name and m::name and n::name and
  sm::name and sm'::name and sn::name and sn'::name
where atom l # (s,k,w,j,sl,sl',m,n,sm,sm',sn,sn')
  atom sl # (s,w,j,sl',m,n,sm,sm',sn,sn') atom sl' # (s,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: SeqQuoteP.simps [of l _ _ sl sl' m n sm sm' sn sn'])
qed

```

```

declare SeqQuoteP.simps [simp del]

```

9.1.2 Correctness properties

```

lemma SeqQuoteP_lemma:
fixes m::name and sm::name and sm'::name and n::name and sn::name and sn'::name
assumes atom m # (t,u,s,k,n,sm,sm',sn,sn') atom n # (t,u,s,k,sm,sm',sn,sn')
  atom sm # (t,u,s,k,sm',sn,sn') atom sm' # (t,u,s,k,sn,sn')
  atom sn # (t,u,s,k,sn') atom sn' # (t,u,s,k)
shows { SeqQuoteP t u s k }
  ⊢ (t EQ Zero AND u EQ Zero) OR
  Ex m (Ex n (Ex sm (Ex sm' (Ex sn (Ex sn' (Var m IN k AND Var n IN k AND
  SeqQuoteP (Var sm) (Var sm') s (Var m) AND
  SeqQuoteP (Var sn) (Var sn') s (Var n) AND
  t EQ Eats (Var sm) (Var sn) AND
  u EQ Q_Eats (Var sm') (Var sn'))))))))
proof –
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  $x = \text{Var } m$ ], 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_all add: SeqQuoteP.simps [of  $l\ s\ \_sl\ sl'\ m\ n\ sm\ sm'\ sn\ sn'$ ])
apply ((rule Conj_I)+, blast intro: LstSeqP_Mem)+
— first SeqQuoteP subgoal
apply (rule All2_Subset [OF Hyp])
apply (blast intro!: SUCC_Subset_Ord LstSeqP_OrdP)+
apply simp
— next SeqQuoteP subgoal
apply ((rule Conj_I)+, blast intro: LstSeqP_Mem)+
apply (rule All2_Subset [OF Hyp], blast)
apply (auto intro!: SUCC_Subset_Ord LstSeqP_OrdP intro: Trans)
done
qed

```

9.2 The “special function” itself

```

nominal_function QuoteP ::  $tm \Rightarrow tm \Rightarrow fm$ 
  where  $\llbracket atom\ s \# (t,u,k); atom\ k \# (t,u) \rrbracket \Longrightarrow$ 
     $QuoteP\ t\ u = Ex\ s\ (Ex\ k\ (SeqQuoteP\ t\ u\ (Var\ s)\ (Var\ k)))$ 
by (auto simp: eqvt_def QuoteP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)

```

```

nominal_termination (eqvt)
  by lexicographic_order

```

```

lemma
  shows QuoteP_fresh_iff [simp]:  $a \# QuoteP\ t\ u \longleftrightarrow a \# t \wedge a \# u$  (is ?thesis1)
  and QuoteP_sf [iff]:  $Sigma\_fm\ (QuoteP\ t\ u)$  (is ?thsf)
proof –
  obtain  $s::name$  and  $k::name$  where  $atom\ s \# (t,u,k)$   $atom\ k \# (t,u)$ 
  by (metis obtain_fresh)
  thus ?thesis1 ?thsf
  by auto
qed

```

```

lemma QuoteP_subst [simp]:
   $(QuoteP\ t\ u)(j::w) = QuoteP\ (subst\ j\ w\ t)\ (subst\ j\ w\ u)$ 
proof –
  obtain  $s::name$  and  $k::name$  where  $atom\ s \# (t,u,w,j,k)$   $atom\ k \# (t,u,w,j)$ 
  by (metis obtain_fresh)
  thus ?thesis
  by (simp add: QuoteP.simps [of  $s\ \_ \_ k$ ])
qed

```

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

9.2.1 Correctness properties

lemma *QuoteP_Zero*: {} ⊢ *QuoteP Zero Zero*

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
  and s :: name
  and k :: name
where atom l # (s, k, sl, sl', m, n, sm, sm', sn, sn')
  and atom sl # (s, k, sl', m, n, sm, sm', sn, sn')
  and atom sl' # (s, k, m, n, sm, sm', sn, sn')
  and atom m # (s, k, n, sm, sm', sn, sn')
  and atom n # (s, k, sm, sm', sn, sn')
  and atom sm # (s, k, sm', sn, sn')
  and atom sm' # (s, k, sn, sn')
  and atom sn # (s, k, sn')
  and atom sn' # (s, k)
  and atom k # s
by (metis obtain_fresh)
then show ?thesis
  apply (subst QuoteP.simps[of s _ _ k]; simp)
  apply (rule Ex_I[of _ _ _ Eats Zero (HPair Zero Zero)]; simp)
  apply (rule Ex_I[of _ _ _ Zero]; simp)
  apply (subst SeqQuoteP.simps[of l _ _ sl sl' m n sm sm' sn sn']; simp?)
  apply (rule Conj_I)
  apply (rule LstSeqP_single)
  apply (auto intro!: Ex_I[of _ _ _ Zero])
  apply (rule Mem_SUCC_E[OF Mem_Zero_E])
  apply (rule Mem_Eats_I2)
  apply (rule HPair_cong[OF Assume Refl])
  apply (auto intro!: Disj_I1)
done

```

qed

lemma *SeqQuoteP_Eats*:

```

assumes atom s # (k, s1, s2, k1, k2, t1, t2, u1, u2) atom k # (s1, s2, k1, k2, t1, t2, u1, u2)
shows {SeqQuoteP t1 u1 s1 k1, SeqQuoteP t2 u2 s2 k2} ⊢
  Ex s (Ex k (SeqQuoteP (Eats t1 t2) (Q_Eats u1 u2) (Var s) (Var k)))

```

proof –

```

obtain km::name and kn::name and j::name and k'::name and l::name
  and sl::name and sl'::name and m::name and n::name and sm::name
  and sm'::name and sn::name and sn'::name
where atoms2:
  atom km # (kn, j, k', l, s1, s2, s, k1, k2, k, t1, t2, u1, u2, sl, sl', m, n, sm, sm', sn, sn')
  atom kn # (j, k', l, s1, s2, s, k1, k2, k, t1, t2, u1, u2, sl, sl', m, n, sm, sm', sn, sn')
  atom j # (k', l, s1, s2, s, k1, k2, k, t1, t2, u1, u2, sl, sl', m, n, sm, sm', sn, sn')
  and atoms: atom k' # (l, s1, s2, s, k1, k2, k, t1, t2, u1, u2, sl, sl', m, n, sm, sm', sn, sn')
  atom l # (s1, s2, s, k1, k2, k, t1, t2, u1, u2, sl, sl', m, n, sm, sm', sn, sn')
  atom sl # (s1, s2, s, k1, k2, k, t1, t2, u1, u2, sl', m, n, sm, sm', sn, sn')

```

```

atom sl' # (s1,s2,s,k1,k2,k,t1,t2,u1,u2,m,n,sm,sm',sn,sn')
atom m # (s1,s2,s,k1,k2,k,t1,t2,u1,u2,n,sm,sm',sn,sn')
atom n # (s1,s2,s,k1,k2,k,t1,t2,u1,u2,sm,sm',sn,sn')
atom sm # (s1,s2,s,k1,k2,k,t1,t2,u1,u2,sm',sn,sn')
atom sm' # (s1,s2,s,k1,k2,k,t1,t2,u1,u2,sn,sn')
atom sn # (s1,s2,s,k1,k2,k,t1,t2,u1,u2,sn')
atom sn' # (s1,s2,s,k1,k2,k,t1,t2,u1,u2)
by (metis obtain_fresh)
show ?thesis
using assms atoms
apply (auto simp: SeqQuoteP.simps [of l Var s _ sl sl' m n sm sm' sn sn'])
apply (rule cut_same [where A=OrdP k1 AND OrdP k2])
apply (metis Conj_I SeqQuoteP_imp_OrdP thin1 thin2)
apply (rule cut_same [OF exists_SeqAppendP [of s s1 SUCC k1 s2 SUCC k2]])
apply (rule AssumeH Ex_EH Conj_EH | simp)+
apply (rule cut_same [OF exists_HaddP [where j=k' and x=k1 and y=k2]])
apply (rule AssumeH Ex_EH Conj_EH | simp)+
apply (rule Ex_I [where x=Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats t1 t2)
(Q_Eats u1 u2))))))
apply (simp_all (no_asm_simp))
apply (rule Ex_I [where x=SUCC (SUCC (Var k'))])
apply simp
apply (rule Conj_I [OF LstSeqP_SeqAppendP_Eats])
apply (blast intro: SeqQuoteP_imp_LstSeqP [THEN cut1])+
proof (rule All2_SUCC_I, simp_all)
show {HaddP k1 k2 (Var k'), OrdP k1, OrdP k2, SeqAppendP s1 (SUCC k1) s2 (SUCC k2) (Var s),
SeqQuoteP t1 u1 s1 k1, SeqQuoteP t2 u2 s2 k2}
  ⊢ Ex sl (Ex sl'
    (HPair (SUCC (SUCC (Var k'))) (HPair (Var sl) (Var sl')) IN
    Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats t1 t2) (Q_Eats u1 u2))) AND
    (Var sl EQ Zero AND Var sl' EQ Zero OR
    Ex m (Ex n (Ex sm (Ex sm' (Ex sn (Ex sn'
      (Var m IN SUCC (SUCC (Var k'))) AND
      Var n IN SUCC (SUCC (Var k'))) AND
      HPair (Var m) (HPair (Var sm) (Var sm')) IN
      Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats t1 t2) (Q_Eats u1 u2))))))
    AND
    HPair (Var n) (HPair (Var sn) (Var sn')) IN
    Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats t1 t2) (Q_Eats u1 u2))))))
    AND
    Var sl EQ Eats (Var sm) (Var sn) AND Var sl' EQ Q_Eats (Var sm') (Var sn'))))))))
— verifying the final values
apply (rule Ex_I [where x=Eats t1 t2])
using assms atoms apply simp
apply (rule Ex_I [where x=Q_Eats u1 u2], simp)
apply (rule Conj_I [OF Mem_Eats_I2 [OF Refl]])
apply (rule Disj_I2)
apply (rule Ex_I [where x=k1], simp)
apply (rule Ex_I [where x=SUCC (Var k')], simp)
apply (rule Ex_I [where x=t1], simp)
apply (rule Ex_I [where x=u1], simp)
apply (rule Ex_I [where x=t2], simp)
apply (rule Ex_I [where x=u2], simp)
apply (rule Conj_I)
apply (blast intro: HaddP_Mem_I Mem_SUCC_I1)
apply (rule Conj_I [OF Mem_SUCC_Refl])
apply (rule Conj_I)
apply (blast intro: Mem_Eats_I1 SeqAppendP_Mem1 [THEN cut3] Mem_SUCC_Refl)

```

```

    SeqQuoteP_imp_LstSeqP [THEN cut1] LstSeqP_imp_Mem)
  apply (blast intro: Mem_Eats_I1 SeqAppendP_Mem2 [THEN cut4] Mem_SUCC_Refl
    SeqQuoteP_imp_LstSeqP [THEN cut1] LstSeqP_imp_Mem HaddP_SUCC1 [THEN cut1])
  done
next
show {HaddP k1 k2 (Var k'), OrdP k1, OrdP k2, SeqAppendP s1 (SUCC k1) s2 (SUCC k2) (Var s),
  SeqQuoteP t1 u1 s1 k1, SeqQuoteP t2 u2 s2 k2}
  ⊢ All2 l (SUCC (SUCC (Var k')))
    (Ex sl (Ex sl'
      (HPair (Var l) (HPair (Var sl) (Var sl')) IN
        Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats t1 t2) (Q_Eats u1 u2))))
AND
    (Var sl EQ Zero AND Var sl' EQ Zero OR
      Ex m (Ex n (Ex sm (Ex sm' (Ex sn (Ex sn'
        (Var m IN Var l AND
          Var n IN Var l AND
            HPair (Var m) (HPair (Var sm) (Var sm')) IN
              Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats t1 t2) (Q_Eats u1 u2))))
AND
      HPair (Var n) (HPair (Var sn) (Var sn')) IN
        Eats (Var s) (HPair (SUCC (SUCC (Var k'))) (HPair (Eats t1 t2) (Q_Eats u1 u2))))
AND
    Var sl EQ Eats (Var sm) (Var sn) AND Var sl' EQ Q_Eats (Var sm') (Var sn'))))))))
— verifying the sequence buildup
  apply (rule cut_same [where A=HaddP (SUCC k1) (SUCC k2) (SUCC (SUCC (Var k')))]
  apply (blast intro: HaddP_SUCC1 [THEN cut1] HaddP_SUCC2 [THEN cut1])
  apply (rule All_I Imp_I)+
  apply (rule HaddP_Mem_cases [where i=j])
  using assms atoms atoms2 apply simp_all
  apply (rule AssumeH)
  apply (blast intro: OrdP_SUCC_I)
— ... the sequence buildup via s1
  apply (simp add: SeqQuoteP.simps [of l s1 __ sl sl' m n sm sm' sn sn'])
  apply (rule AssumeH Ex_EH Conj_EH)+
  apply (rule All2_E [THEN rotate2])
  apply (simp | rule AssumeH Ex_EH Conj_EH)+
  apply (rule Ex_I [where x=Var sl], simp)
  apply (rule Ex_I [where x=Var sl'], simp)
  apply (rule Conj_I)
  apply (rule Mem_Eats_I1)
  apply (metis SeqAppendP_Mem1 rotate3 thin2 thin4)
  apply (rule AssumeH Disj_IE1H Ex_EH Conj_EH)+
  apply (rule Ex_I [where x=Var m], simp)
  apply (rule Ex_I [where x=Var n], simp)
  apply (rule Ex_I [where x=Var sm], simp)
  apply (rule Ex_I [where x=Var sm'], simp)
  apply (rule Ex_I [where x=Var sn], simp)
  apply (rule Ex_I [where x=Var sn'], simp_all)
  apply (rule Conj_I, rule AssumeH)+
  apply (blast intro: OrdP_Trans [OF OrdP_SUCC_I] Mem_Eats_I1 [OF SeqAppendP_Mem1 [THEN
cut.3]] 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 sm1], simp)
apply (rule Ex_I [where x=Var sn], simp)
apply (rule Ex_I [where x=Var sn1], simp_all)
apply (rule Conj_I [OF_ Conj_I])
apply (blast intro: Hyp OrdP_SUCC_I HaddP_Mem_cancel_left [THEN Iff_MP2_same])
apply (blast intro: Hyp OrdP_SUCC_I HaddP_Mem_cancel_left [THEN Iff_MP2_same])
apply (blast intro: Hyp Mem_Eats_I1 SeqAppendP_Mem2 [THEN cut4] OrdP_Trans HaddP_imp_OrdP
[THEN cut1])
  done
qed
qed

```

lemma QuoteP_Eats: {QuoteP t1 u1, QuoteP t2 u2} ⊢ QuoteP (Eats t1 t2) (Q_Eats u1 u2)

proof –

```

obtain k1::name and s1::name and k2::name and s2::name and k::name and s::name
  where atom s1 # (t1,u1,t2,u2)          atom k1 # (t1,u1,t2,u2,s1)
        atom s2 # (t1,u1,t2,u2,k1,s1)    atom k2 # (t1,u1,t2,u2,s2,k1,s1)
        atom s  # (t1,u1,t2,u2,k2,s2,k1,s1) atom k  # (t1,u1,t2,u2,s,k2,s2,k1,s1)
  by (metis obtain_fresh)
thus ?thesis
  by (auto simp: QuoteP_simps [of s _ (Q_Eats u1 u2) k]
      QuoteP_simps [of s1 t1 u1 k1] QuoteP_simps [of s2 t2 u2 k2]
      intro!: SeqQuoteP_Eats [THEN cut2])
qed

```

lemma exists_QuoteP:

assumes j: atom j # x **shows** {} ⊢ Ex j (QuoteP x (Var j))

proof –

```

obtain i::name and j'::name and k::name
  where atoms: atom i # (j,x) atom j' # (i,j,x) atom (k::name) # (i,j,j',x)
  by (metis obtain_fresh)
have {} ⊢ Ex j (QuoteP (Var i) (Var j)) (is {} ⊢ ?scheme)
proof (rule Ind [of k])
  show atom k # (i, ?scheme) using atoms
    by simp
next
  show {} ⊢ ?scheme(i::=Zero) using j atoms
    by (auto intro: Ex_I [where x=Zero] simp add: QuoteP_Zero)
next
  show {} ⊢ All i (All k (?scheme IMP ?scheme(i::=Var k) IMP ?scheme(i::=Eats (Var i) (Var k))))
    apply (rule All_I Imp_I)+
    using atoms assms
    apply simp_all
    apply (rule Ex_E)

```

```

apply (rule Ex_E_with_renaming [where  $i'=j'$ , THEN rotate2], auto)
apply (rule Ex_I [where  $x=Q\_Eats (Var j) (Var j)$ ], auto intro: QuoteP_Eats)
done
qed
hence {}  $\vdash (Ex j (QuoteP (Var i) (Var j))) (i::= x)$ 
  by (rule Subst) auto
thus ?thesis
  using atoms j by auto
qed

lemma QuoteP_imp_ConstP: { QuoteP x y }  $\vdash$  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) }
   $\vdash All j (All j' (SeqQuoteP (Var j) (Var j') (Var s) (Var k) IMP ConstP (Var j')))$ 
  (is _  $\vdash$  ?scheme)
proof (rule OrdIndH [where  $j=l$ ])
  show  $atom l \# (k, ?scheme)$  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 cutI [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 sn
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]

9.3 The Operator *quote_all*

9.3.1 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 $quote_all\ ((atom\ i \Rightarrow atom\ i') + p)\ V = quote_all\ p\ V$
proof –
 { fix W
 assume $w: W \subseteq V$
 have $finite\ W$
 by (metis finite_V finite_subset w)
 hence $quote_all\ ((atom\ i \Rightarrow atom\ i') + p)\ W = quote_all\ p\ W$ 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

```

9.3.2 Transferring theorems to the level of derivability

```

context quote_perm
begin

```

```

lemma QuoteP_imp_ConstP_F_hyps:
  assumes  $Us \subseteq Vs \{ConstP (F i) \mid i. i \in Us\} \vdash A$  shows  $quote\_all\ p\ Us \vdash A$ 
proof –
  show ?thesis using finite_V [OF ‹Us ⊆ Vs›] assms
  proof (induction arbitrary: A rule: finite_induct)
  case empty thus ?case by simp
  next
  case (insert v Us) thus ?case
    by (auto simp: Collect_disj_Un)
    (metis (lifting) anti_deduction Imp_cut [OF _ QuoteP_imp_ConstP] Disj_I2 F_unfold)
  qed
qed

```

Lemma 8.3

```

theorem quote_all_PfP_ssubst:
  assumes  $\beta: \{\} \vdash \beta$ 
  and  $V: V \subseteq Vs$ 
  and  $s: supp\ \beta \subseteq atom\ 'Vs$ 
  shows  $quote\_all\ p\ V \vdash PfP\ (ssubst\ [\beta]\ V\ V\ F)$ 
proof –
  have  $\{\} \vdash PfP\ \langle\beta\rangle$ 
  by (metis β proved_imp_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  $quote\_all\ p\ V \vdash PfP\ (ssubst\ [\alpha]\ V\ V\ F)\ IMP\ PfP\ (ssubst\ [\beta]\ V\ V\ F)$ 
using quote_all_PfP_ssubst [OF A V] s
  by (auto simp: V quot_fm_def intro: PfP_implies_ModPon_PfP thin1)

```

Lemma 8.4b

```

corollary quote_all_MonPon2_PfP_ssubst:
  assumes  $A: \{\} \vdash \alpha1\ IMP\ \alpha2\ IMP\ \beta$ 
  and  $V: V \subseteq Vs$ 
  and  $s: supp\ \alpha1 \subseteq atom\ 'Vs\ supp\ \alpha2 \subseteq atom\ 'Vs\ supp\ \beta \subseteq atom\ 'Vs$ 
  shows  $quote\_all\ p\ V \vdash PfP\ (ssubst\ [\alpha1]\ V\ V\ F)\ IMP\ PfP\ (ssubst\ [\alpha2]\ V\ V\ F)\ IMP\ PfP\ (ssubst\ [\beta]\ V\ V\ F)$ 
using quote_all_PfP_ssubst [OF A V] s

```

by (force simp: V vquot_fm_def intro: Pfp_implies_ModPon_Pfp [OF Pfp_implies_ModPon_Pfp] thin1)

lemma quote_all_Disj_I1_Pfp_ssubst:

assumes $V \subseteq Vs$ supp $\alpha \subseteq atom$ ' Vs supp $\beta \subseteq atom$ ' Vs
 and prems: $H \vdash Pfp$ (ssubst $[\alpha]$ V V F) quote_all p $V \subseteq H$
 shows $H \vdash Pfp$ (ssubst $[\alpha \text{ OR } \beta]$ V V F)

proof –

have $\{\} \vdash \alpha \text{ IMP } (\alpha \text{ OR } \beta)$

by (blast intro: Disj_I1)

hence quote_all p $V \vdash Pfp$ (ssubst $[\alpha]$ V V F) IMP Pfp (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 $V \subseteq Vs$ supp $\alpha \subseteq atom$ ' Vs supp $\beta \subseteq atom$ ' Vs
 and prems: $H \vdash Pfp$ (ssubst $[\beta]$ V V F) quote_all p $V \subseteq H$
 shows $H \vdash Pfp$ (ssubst $[\alpha \text{ OR } \beta]$ V V F)

proof –

have $\{\} \vdash \beta \text{ IMP } (\alpha \text{ OR } \beta)$

by (blast intro: Disj_I2)

hence quote_all p $V \vdash Pfp$ (ssubst $[\beta]$ V V F) IMP Pfp (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 $V \subseteq Vs$ supp $\alpha \subseteq atom$ ' Vs supp $\beta \subseteq atom$ ' Vs
 and prems: $H \vdash Pfp$ (ssubst $[\alpha]$ V V F) $H \vdash Pfp$ (ssubst $[\beta]$ V V F) quote_all p $V \subseteq H$
 shows $H \vdash Pfp$ (ssubst $[\alpha \text{ AND } \beta]$ V V F)

proof –

have $\{\} \vdash \alpha \text{ IMP } \beta \text{ IMP } (\alpha \text{ AND } \beta)$

by blast

hence quote_all p V

$\vdash Pfp$ (ssubst $[\alpha]$ V V F) IMP Pfp (ssubst $[\beta]$ V V F) IMP Pfp (ssubst $[\alpha \text{ AND } \beta]$ 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 $[\text{Neg } \alpha]$ V V F) IMP Pfp (ssubst $[\text{Fls}]$ V V F)

proof –

have $\{\} \vdash \alpha \text{ IMP } \text{Neg } \alpha \text{ IMP } \text{Fls}$

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 $\llbracket \text{atom } i \# p \cdot V; V \subseteq Vs \rrbracket \implies \text{atom } i \# \text{ssubst } (\llbracket A \rrbracket V) V F$

by (simp add: fresh_ssubst_dbfm vquot_fm_def)

end

9.4 Star Property. Equality and Membership: Lemmas 9.3 and 9.4

lemma SeqQuoteP_Mem_imp_QMem_and_Subset:

assumes atom i # (j,j',i',si,ki,sj,kj) atom i' # (j,j',si,ki,sj,kj)

atom j # (j',si,ki,sj,kj) atom j' # (si,ki,sj,kj)

atom si # (ki,sj,kj) atom sj # (ki,kj)

shows {SeqQuoteP (Var i) (Var i') (Var si) ki, SeqQuoteP (Var j) (Var j') (Var sj) kj}

\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,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)}

\vdash 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 $_ \vdash$?scheme)

proof (rule OrdIndH [where j=l])

show atom l # (k, ?scheme) using atoms

by simp

next

define V p where V = {i,j,sm,sn}

and p = (atom i \equiv atom i') + (atom j \equiv atom j') +

(atom sm \equiv atom sm') + (atom sn \equiv atom sn')

define F where F \equiv 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 \equiv make_F V p

```

    by (rule F_def)
  qed
  have V_mem:  $i \in V \ j \in V \ sm \in V \ sn \in V$ 
    by (auto simp: V_def) — Part of (2) from page 32
  have Mem1:  $\{\} \vdash (\text{Var } i \text{ IN } \text{Var } sm) \text{ IMP } (\text{Var } i \text{ IN } \text{Eats } (\text{Var } sm) (\text{Var } sn))$ 
    by (blast intro: Mem_Eats_I1)
  have Q_Mem1: quote_all p V
     $\vdash \text{PfP } (Q\_Mem (\text{Var } i') (\text{Var } sm')) \text{ IMP}$ 
     $\text{PfP } (Q\_Mem (\text{Var } i') (Q\_Eats (\text{Var } sm') (\text{Var } sn')))$ 
    using qp.quote_all_MonPon_PfP_ssubst [OF Mem1 subset_refl] assms atoms V_mem
    by (simp add: vquot_fm_def qp.Vs) (simp add: qp.F_unfold_p_def)
  have Mem2:  $\{\} \vdash (\text{Var } i \text{ EQ } \text{Var } sn) \text{ IMP } (\text{Var } i \text{ IN } \text{Eats } (\text{Var } sm) (\text{Var } sn))$ 
    by (blast intro: Mem_Eats_I2)
  have Q_Mem2: quote_all p V
     $\vdash \text{PfP } (Q\_Eq (\text{Var } i') (\text{Var } sn')) \text{ IMP}$ 
     $\text{PfP } (Q\_Mem (\text{Var } i') (Q\_Eats (\text{Var } sm') (\text{Var } sn')))$ 
    using qp.quote_all_MonPon_PfP_ssubst [OF Mem2 subset_refl] assms atoms V_mem
    by (simp add: vquot_fm_def qp.Vs) (simp add: qp.F_unfold_p_def)
  have Subs1:  $\{\} \vdash \text{Zero SUBS } \text{Var } j$ 
    by blast
  have Q_Subst1:  $\{\text{QuoteP } (\text{Var } j) (\text{Var } j')\} \vdash \text{PfP } (Q\_Subset \text{Zero } (\text{Var } j'))$ 
    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:  $\{\} \vdash \text{Var } sm \text{ SUBS } \text{Var } j \text{ IMP } \text{Var } sn \text{ IN } \text{Var } j \text{ IMP } \text{Eats } (\text{Var } sm) (\text{Var } sn) \text{ SUBS } \text{Var}$ 
  j
    by blast
  have Q_Subst2: quote_all p V
     $\vdash \text{PfP } (Q\_Subset (\text{Var } sm') (\text{Var } j')) \text{ IMP}$ 
     $\text{PfP } (Q\_Mem (\text{Var } sn') (\text{Var } j')) \text{ IMP}$ 
     $\text{PfP } (Q\_Subset (Q\_Eats (\text{Var } sm') (\text{Var } sn')) (\text{Var } j'))$ 
    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:  $\{\} \vdash \text{Var } i \text{ SUBS } \text{Var } sn \text{ IMP } \text{Var } sn \text{ SUBS } \text{Var } i \text{ IMP } \text{Var } i \text{ EQ } \text{Var } sn$ 
    by (blast intro: Equality_I)
  have Q_Ext:  $\{\text{QuoteP } (\text{Var } i) (\text{Var } i'), \text{QuoteP } (\text{Var } sn) (\text{Var } sn')\}$ 
     $\vdash \text{PfP } (Q\_Subset (\text{Var } i') (\text{Var } sn')) \text{ IMP}$ 
     $\text{PfP } (Q\_Subset (\text{Var } sn') (\text{Var } i')) \text{ IMP}$ 
     $\text{PfP } (Q\_Eq (\text{Var } i') (\text{Var } sn'))$ 
    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)
    (simp add: qp.ssubst_single)
    (simp add: vquot_fm_def qp.F_unfold_p_def V_def)
  show  $\{\} \vdash \text{All } k (\text{OrdP } (\text{Var } k) \text{ IMP } (\text{All2 } l (\text{Var } k) (?scheme(k::= \text{Var } l)) \text{ IMP } ?scheme))$ 
    apply (rule All_I Imp_I)+
    using atoms assms
    apply simp_all
    apply (rule cut_same [where  $A = \text{QuoteP } (\text{Var } i) (\text{Var } i')$ ])
    apply (blast intro: QuoteP_I)
    apply (rule cut_same [where  $A = \text{QuoteP } (\text{Var } j) (\text{Var } j')$ ])
    apply (blast intro: QuoteP_I)
    apply (rule rotate6)
    apply (rule Conj_I)
    —  $\text{Var } i \text{ IN } \text{Var } j \text{ IMP } \text{PfP } (Q\_Mem (\text{Var } i') (\text{Var } j'))$ 
    apply (rule cut_same)
    apply (rule cut1 [OF SeqQuoteP_lemma [of  $m \text{ Var } j \text{ Var } j' \text{ Var } sj \text{ Var } lj \text{ n } sm \text{ sm}' \text{ sn } sn'$ ]],
    simp_all, blast)

```

```

apply (rule Imp_I Disj_EH Conj_EH) +
— case 1, Var j EQ Zero
apply (rule cut_same [where A = Var i IN Zero])
apply (blast intro: Mem_cong [THEN Iff_MP_same], blast)
— case 2, Var j EQ Eats (Var sm) (Var sn)
apply (rule Imp_I Conj_EH Ex_EH) +
apply simp_all
apply (rule Var_Eq_subst_Iff [THEN rotate2, THEN Iff_MP_same], simp)
apply (rule cut_same [where A = QuoteP (Var sm) (Var sm')])
apply (blast intro: QuoteP_I)
apply (rule cut_same [where A = QuoteP (Var sn) (Var sn')])
apply (blast intro: QuoteP_I)
apply (rule cut_same [where A = Var i IN Eats (Var sm) (Var sn)])
apply (rule Mem_cong [OF Refl, THEN Iff_MP_same])
apply (rule AssumeH Mem_Eats_E) +
— Eats case 1. IH for sm
apply (rule cut_same [where A = OrdP (Var m)])
apply (blast intro: Hyp Ord_IN_Ord SeqQuoteP_imp_OrdP [THEN cut1])
apply (rule cut_same [OF exists_HaddP [where j=l and x=Var li and y=Var m]])
apply auto
apply (rule All2_E [where x=Var l, THEN rotate13], simp_all)
apply (blast intro: Hyp HaddP_Mem_cancel_left [THEN Iff_MP2_same] SeqQuoteP_imp_OrdP
[THEN cut1])
apply (rule All_E [where x=Var i], simp)
apply (rule All_E [where x=Var i'], simp)
apply (rule All_E [where x=Var si], simp)
apply (rule All_E [where x=Var li], simp)
apply (rule All_E [where x=Var sm], simp)
apply (rule All_E [where x=Var sm'], simp)
apply (rule All_E [where x=Var sj], simp)
apply (rule All_E [where x=Var m], simp)
apply (force intro: MP_thin [OF Q_Mem1] simp add: V_def p_def)
— Eats case 2
apply (rule rotate13)
apply (rule cut_same [where A = OrdP (Var n)])
apply (blast intro: Hyp Ord_IN_Ord SeqQuoteP_imp_OrdP [THEN cut1])
apply (rule cut_same [OF exists_HaddP [where j=l and x=Var li and y=Var n]])
apply auto
apply (rule MP_same)
apply (rule Q_Mem2 [THEN thin])
apply (simp add: V_def p_def)
apply (rule MP_same)
apply (rule MP_same)
apply (rule Q_Ext [THEN thin])
apply (simp add: V_def p_def)
— PfP (Q_Subset (Var i') (Var sn'))
apply (rule All2_E [where x=Var l, THEN rotate14], simp_all)
apply (blast intro: Hyp HaddP_Mem_cancel_left [THEN Iff_MP2_same] SeqQuoteP_imp_OrdP
[THEN cut1])
apply (rule All_E [where x=Var i], simp)
apply (rule All_E [where x=Var i'], simp)
apply (rule All_E [where x=Var si], simp)
apply (rule All_E [where x=Var li], simp)
apply (rule All_E [where x=Var sn], simp)
apply (rule All_E [where x=Var sn'], simp)
apply (rule All_E [where x=Var sj], simp)
apply (rule All_E [where x=Var n], simp)
apply (rule Imp_E, blast intro: Hyp) +

```

```

apply (rule Conj_E)
apply (rule thin1)
apply (blast intro: Imp_E EQ_imp_SUBS [THEN cut1])
— Pfp (Q_Subset (Var sn') (Var i'))
apply (rule All2_E [where x=Var l, THEN rotate14], simp_all)
apply (blast intro: Hyp HaddP_Mem_cancel_left [THEN Iff_MP2_same] SeqQuoteP_imp_OrdP
[THEN cut1])
apply (rule All_E [where x=Var sn], simp)
apply (rule All_E [where x=Var sn'], simp)
apply (rule All_E [where x=Var sj], simp)
apply (rule All_E [where x=Var n], simp)
apply (rule All_E [where x=Var i], simp)
apply (rule All_E [where x=Var i'], simp)
apply (rule All_E [where x=Var si], simp)
apply (rule All_E [where x=Var li], simp)
apply (rule Imp_E, blast intro: Hyp)+
apply (rule Imp_E)
apply (blast intro: Hyp HaddP_commute [THEN cut2] SeqQuoteP_imp_OrdP [THEN cut1])
apply (rule Conj_E)
apply (rule thin1)
apply (blast intro: Imp_E EQ_imp_SUBS2 [THEN cut1])
— Var i SUBS Var j IMP Pfp (Q_Subset (Var i') (Var j'))
apply (rule cut_same)
apply (rule cut1 [OF SeqQuoteP_lemma [of m Var i Var i' Var si Var li n sm sm' sn sn']],
simp_all, blast)
apply (rule Imp_I Disj_EH Conj_EH)+
— case 1, Var i EQ Zero
apply (rule cut_same [where A = Pfp (Q_Subset Zero (Var j'))])
apply (blast intro: Q_Subs1 [THEN cut1] SeqQuoteP_imp_QuoteP [THEN cut1])
apply (force intro: Var_Eq_subst_Iff [THEN Iff_MP_same, THEN rotate3])
— case 2, Var i EQ Eats (Var sm) (Var sn)
apply (rule Conj_EH Ex_EH)+
apply simp_all
apply (rule cut_same [where A = OrdP (Var lj)])
apply (blast intro: Hyp SeqQuoteP_imp_OrdP [THEN cut1])
apply (rule Var_Eq_subst_Iff [THEN Iff_MP_same, THEN rotate3], simp)
apply (rule cut_same [where A = QuoteP (Var sm) (Var sm')])
apply (blast intro: QuoteP_I)
apply (rule cut_same [where A = QuoteP (Var sn) (Var sn')])
apply (blast intro: QuoteP_I)
apply (rule cut_same [where A = Eats (Var sm) (Var sn) SUBS Var j])
apply (rule Subset_cong [OF _ Refl, THEN Iff_MP_same])
apply (rule AssumeH Mem_Eats_E)+
— Eats case split
apply (rule Eats_Subset_E)
apply (rule rotate15)
apply (rule MP_same [THEN MP_same])
apply (rule Q_Subs2 [THEN thin])
apply (simp add: V_def p_def)
— Eats case 1: Pfp (Q_Subset (Var sm') (Var j'))
apply (rule cut_same [OF exists_HaddP [where j=l and x=Var m and y=Var lj]])
apply (rule AssumeH Ex_EH Conj_EH | simp)+
— IH for sm
apply (rule All2_E [where x=Var l, THEN rotate15], simp_all)
apply (blast intro: Hyp HaddP_Mem_cancel_right_Mem SeqQuoteP_imp_OrdP [THEN cut1])
apply (rule All_E [where x=Var sm], simp)
apply (rule All_E [where x=Var sm'], simp)
apply (rule All_E [where x=Var si], simp)

```

```

apply (rule All_E [where x=Var m], simp)
apply (rule All_E [where x=Var j], simp)
apply (rule All_E [where x=Var j'], simp)
apply (rule All_E [where x=Var sj], simp)
apply (rule All_E [where x=Var lj], simp)
apply (blast intro: thin1 Imp_E)
— Eats case 2: Pfp (Q_Mem (Var sn') (Var j'))
apply (rule cut_same [OF exists_HaddP [where j=l and x=Var n and y=Var lj]])
apply (rule AssumeH Ex_EH Conj_EH | simp)+
— IH for sn
apply (rule All2_E [where x=Var l, THEN rotate15], simp_all)
apply (blast intro: Hyp HaddP_Mem_cancel_right_Mem SeqQuoteP_imp_OrdP [THEN cut1])
apply (rule All_E [where x=Var sn], simp)
apply (rule All_E [where x=Var sn'], simp)
apply (rule All_E [where x=Var si], simp)
apply (rule All_E [where x=Var n], simp)
apply (rule All_E [where x=Var j], simp)
apply (rule All_E [where x=Var j'], simp)
apply (rule All_E [where x=Var sj], simp)
apply (rule All_E [where x=Var lj], simp)
apply (blast intro: Hyp Imp_E)
done
qed
hence p1: {OrdP(Var k)}
  ⊢ (All i' (All si (All li
    (All j (All j' (All sj (All lj
      (SeqQuoteP (Var i) (Var i') (Var si) (Var li) IMP
      SeqQuoteP (Var j) (Var j') (Var sj) (Var lj) IMP
      HaddP (Var li) (Var lj) (Var k) IMP
      (Var i IN Var j IMP Pfp (Q_Mem (Var i') (Var j')))) AND
      (Var i SUBS Var j IMP Pfp (Q_Subset (Var i') (Var j')))))))) (i' ::= Var i)
    by (metis All_D)
have p2: {OrdP(Var k)}
  ⊢ (All si (All li
    (All j (All j' (All sj (All lj
      (SeqQuoteP (Var i) (Var i') (Var si) (Var li) IMP
      SeqQuoteP (Var j) (Var j') (Var sj) (Var lj) IMP
      HaddP (Var li) (Var lj) (Var k) IMP
      (Var i IN Var j IMP Pfp (Q_Mem (Var i') (Var j')))) AND
      (Var i SUBS Var j IMP Pfp (Q_Subset (Var i') (Var j')))))))) (i' ::= Var i')
    apply (rule All_D)
  using atoms p1 by simp
have p3: {OrdP(Var k)}
  ⊢ (All li
    (All j (All j' (All sj (All lj
      (SeqQuoteP (Var i) (Var i') (Var si) (Var li) IMP
      SeqQuoteP (Var j) (Var j') (Var sj) (Var lj) IMP
      HaddP (Var li) (Var lj) (Var k) IMP
      (Var i IN Var j IMP Pfp (Q_Mem (Var i') (Var j')))) AND
      (Var i SUBS Var j IMP Pfp (Q_Subset (Var i') (Var j')))))))) (si ::= Var si)
    apply (rule All_D)
  using atoms p2 by simp
have p4: {OrdP(Var k)}
  ⊢ (All j (All j' (All sj (All lj
    (SeqQuoteP (Var i) (Var i') (Var si) (Var li) IMP
    SeqQuoteP (Var j) (Var j') (Var sj) (Var lj) IMP
    HaddP (Var li) (Var lj) (Var k) IMP
    (Var i IN Var j IMP Pfp (Q_Mem (Var i') (Var j')))) AND
  
```



```

      (Var i SUBS Var j IMP PfP (Q_Subset (Var i') (Var j')))))))) (li::= ki)
apply (rule All_D)
using atoms p3 by simp
have p5: {OrdP (Var k)}
  ⊢ (All j' (All sj (All lj
    (SeqQuoteP (Var i) (Var i') (Var si) ki IMP
    SeqQuoteP (Var j) (Var j') (Var sj) (Var lj) IMP
    HaddP ki (Var lj) (Var k) IMP
    (Var i IN Var j IMP PfP (Q_Mem (Var i') (Var j'))) AND
    (Var i SUBS Var j IMP PfP (Q_Subset (Var i') (Var j')))))))) (j::= Var j)
apply (rule All_D)
using atoms assms p4 by simp
have p6: {OrdP (Var k)}
  ⊢ (All sj (All lj
    (SeqQuoteP (Var i) (Var i') (Var si) ki IMP
    SeqQuoteP (Var j) (Var j') (Var sj) (Var lj) IMP
    HaddP ki (Var lj) (Var k) IMP
    (Var i IN Var j IMP PfP (Q_Mem (Var i') (Var j'))) AND
    (Var i SUBS Var j IMP PfP (Q_Subset (Var i') (Var j')))))))) (j'::= Var j')
apply (rule All_D)
using atoms p5 by simp
have p7: {OrdP (Var k)}
  ⊢ (All lj (SeqQuoteP (Var i) (Var i') (Var si) ki IMP
    SeqQuoteP (Var j) (Var j') (Var sj) (Var lj) IMP
    HaddP ki (Var lj) (Var k) IMP
    (Var i IN Var j IMP PfP (Q_Mem (Var i') (Var j'))) AND
    (Var i SUBS Var j IMP PfP (Q_Subset (Var i') (Var j')))))) (sj::= Var sj)
apply (rule All_D)
using atoms p6 by simp
have p8: {OrdP (Var k)}
  ⊢ (SeqQuoteP (Var i) (Var i') (Var si) ki IMP
    SeqQuoteP (Var j) (Var j') (Var sj) (Var lj) IMP
    HaddP ki (Var lj) (Var k) IMP
    (Var i IN Var j IMP PfP (Q_Mem (Var i') (Var j'))) AND
    (Var i SUBS Var j IMP PfP (Q_Subset (Var i') (Var j')))) (lj::= kj)
apply (rule All_D)
using atoms p7 by simp
hence p9: {OrdP (Var k)}
  ⊢ SeqQuoteP (Var i) (Var i') (Var si) ki IMP
    SeqQuoteP (Var j) (Var j') (Var sj) kj IMP
    HaddP ki kj (Var k) IMP
    (Var i IN Var j IMP PfP (Q_Mem (Var i') (Var j'))) AND
    (Var i SUBS Var j IMP PfP (Q_Subset (Var i') (Var j')))
using assms atoms by simp
have p10: { HaddP ki kj (Var k),
  SeqQuoteP (Var i) (Var i') (Var si) ki,
  SeqQuoteP (Var j) (Var j') (Var sj) kj, OrdP (Var k) }
  ⊢ (Var i IN Var j IMP PfP (Q_Mem (Var i') (Var j'))) AND
    (Var i SUBS Var j IMP PfP (Q_Subset (Var i') (Var j')))
apply (rule MP_same [THEN MP_same [THEN MP_same]])
apply (rule p9 [THEN thin])
apply (auto intro: MP_same)
done
show ?thesis
apply (rule cut_same [OF exists_HaddP [where j=k and x=ki and y=kj]])
apply (metis SeqQuoteP_imp_OrdP thin1)
prefer 2
apply (rule Ex_E)

```

```

apply (rule p10 [THEN cut4])
using assms atoms
apply (auto intro: HaddP_OrdP SeqQuoteP_imp_OrdP [THEN cut1])
done
qed

```

lemma

```

assumes atom i # (j,j',i') atom i' # (j,j') atom j # (j')
shows QuoteP_Mem_imp_QMem:
  {QuoteP (Var i) (Var i'), QuoteP (Var j) (Var j'), Var i IN Var j}
  ⊢ Pfp (Q_Mem (Var i') (Var j')) (is ?thesis1)
and QuoteP_Mem_imp_QSubset:
  {QuoteP (Var i) (Var i'), QuoteP (Var j) (Var j'), Var i SUBS Var j}
  ⊢ Pfp (Q_Subset (Var i') (Var j')) (is ?thesis2)

```

proof –

```

obtain si::name and ki::name and sj::name and kj::name
where atoms: atom si # (ki,sj,kj,i,j,j',i') atom ki # (sj,kj,i,j,j',i')
  atom sj # (kj,i,j,j',i') atom kj # (i,j,j',i')
by (metis obtain_fresh)
hence C: {QuoteP (Var i) (Var i'), QuoteP (Var j) (Var j')}
  ⊢ (Var i IN Var j IMP Pfp (Q_Mem (Var i') (Var j'))) AND
  (Var i SUBS Var j IMP Pfp (Q_Subset (Var i') (Var j')))
using assms
by (auto simp: QuoteP.simps [of si Var i _ ki] QuoteP.simps [of sj Var j _ kj]
  intro!: SeqQuoteP_Mem_imp_QMem_and_Subset del: Conj_I)
show ?thesis1
by (best intro: Conj_E1 [OF C, THEN MP_thin])
show ?thesis2
by (best intro: Conj_E2 [OF C, THEN MP_thin])

```

qed

9.5 Star Property. Universal Quantifier: Lemma 9.7

lemma (in quote_perm) SeqQuoteP_Mem_imp_All2:

```

assumes IH: insert (QuoteP (Var i) (Var i')) (quote_all p Vs)
  ⊢ α IMP Pfp (ssubst [α](insert i Vs) (insert i Vs) Fi)
and sp: supp α – {atom i} ⊆ atom ' Vs
and j: j ∈ Vs and j': p · j = j'
and pi: pi = (atom i ⇒ atom i') + p
and Fi: Fi = make_F (insert i Vs) pi
and atoms: atom i # (j,j',s,k,p) atom i' # (i,p,α)
  atom j # (j',s,k,α) atom j' # (s,k,α)
  atom s # (k,α) atom k # (α,p)
shows insert (SeqQuoteP (Var j) (Var j') (Var s) (Var k)) (quote_all p (Vs–{j}))
  ⊢ All2 i (Var j) α IMP Pfp (ssubst [All2 i (Var j) α] Vs Vs F)

```

proof –

```

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)
define V' p'
  where V' = {sm,sn} ∪ Vs
  and p' = (atom sm ≡ atom sm') + (atom sn ≡ atom sn') + p
define F' where F' ≡ make_F V' p'
interpret qp': quote_perm p' V' F'
proof unfold_locales
  show finite V' by (simp add: V'_def)
  show atom '(p' · V') #* V'
    using atoms atoms' p
  by (auto simp: p'_def V'_def swap_fresh_fresh fresh_at_base_permI
      fresh_star_finite_insert fresh_finite_insert atom_fresh_star_atom_set_conv)
  show F' ≡ make_F V' p'
  by (rule F'_def)
  show - p' = p' using atoms atoms' pinv
  by (simp add: p'_def add.assoc perm_self_inverseI fresh_swap_fresh_plus_perm)
qed
have All2_Zero: {} ⊢ All2 i Zero α
  by auto
have Q_All2_Zero:
  quote_all p Vs ⊢ PFP (Q_All (Q_Imp (Q_Mem (Q_Ind Zero) Zero)
    (ssubst (vquot_dbfm Vs (trans_fm [i] α)) Vs F)))
  using quote_all_PFP_ssubst [OF All2_Zero] assms
  by (force simp add: vquot_fm_def supp_conv_fresh)
have All2_Eats: {} ⊢ All2 i (Var sm) α IMP α(i::=Var sn) IMP All2 i (Eats (Var sm) (Var sn)) α
  using atoms' apply auto
  apply (rule Ex_I [where x = Var i], auto)
  apply (rule rotate2)
  apply (blast intro: ContraProve Var_Eq_imp_subst_Iff [THEN Iff_MP_same])
  done
have [simp]: F' sm = Var sm' F' sn = Var sn' using atoms'
  by (auto simp: V'_def p'_def qp'.F_unfold swap_fresh_fresh fresh_at_base_permI)
have smn' [simp]: sm ∈ V' sn ∈ V' sm ∉ Vs sn ∉ Vs using atoms'
  by (auto simp: V'_def fresh_finite_set_at_base [symmetric])
hence Q_All2_Eats: quote_all p' V'
  ⊢ PFP (ssubst [All2 i (Var sm) α] V' V' F') IMP
  PFP (ssubst [α(i::=Var sn)] V' V' F') IMP
  PFP (ssubst [All2 i (Eats (Var sm) (Var sn)) α] V' V' F')
  using sp qp'.quote_all_MonPon2_PFP_ssubst [OF All2_Eats subset_ref]
  by (simp add: supp_conv_fresh subset_eq V'_def)
  (metis Diff_iff empty_iff fresh_ineq_at_base insertE mem_Collect_eq)
interpret qpi: quote_perm pi insert i Vs Fi
  unfolding pi
  apply (rule qp_insert) using atoms
  apply (auto simp: Fi pi)
  done
have F'_eq_F: ∧name. name ∈ Vs ⇒ F' name = F name
  using atoms'
  by (auto simp: F_unfold qp'.F_unfold p'_def swap_fresh_fresh V'_def fresh_pj)
{ fix t::dbtm
  assume supp t ⊆ atom ' V' supp t ⊆ atom ' Vs
  hence ssubst (vquot_dbtm V' t) V' F' = ssubst (vquot_dbtm Vs t) Vs F
    by (induction t rule: dbtm.induct) (auto simp: F'_eq_F)
} note ssubst_v_tm = this
{ fix A::dbfm
  assume supp A ⊆ atom ' V' supp A ⊆ atom ' Vs

```

```

hence  $ssubst (vquot\_dbfm V' A) V' F' = ssubst (vquot\_dbfm Vs A) Vs F$ 
  by (induction A rule: dbfm.induct) (auto simp: ssubst_v_tm F'_eq_F)
} note  $ssubst\_v\_fm = this$ 
have  $ss\_noprimess: ssubst (vquot\_dbfm V' (trans\_fm [i] \alpha)) V' F' =$ 
   $ssubst (vquot\_dbfm Vs (trans\_fm [i] \alpha)) Vs F$ 
  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  $subst\ i' (Var\ sn') (ssubst (vquot\_dbfm (insert\ i\ Vs)\ A) (insert\ i\ Vs)\ Fi) =$ 
   $ssubst (vquot\_dbfm V' (subst\_dbfm (DBVar\ sn)\ i\ A)) V' F'$ 
  by (induct A rule: dbfm.induct) (auto simp: Un_Diff perm_v_tm)
} note  $perm\_v\_fm = this$ 
have  $quote\_all\ p\ Vs \vdash QuoteP (Var\ i) (Var\ i') IMP$ 
   $(\alpha\ IMP\ Pfp (ssubst\ [\alpha](insert\ i\ Vs) (insert\ i\ Vs)\ Fi))$ 
  using IH by auto
hence  $quote\_all\ p\ Vs$ 
   $\vdash (QuoteP (Var\ i) (Var\ i') IMP$ 
   $(\alpha\ IMP\ Pfp (ssubst\ [\alpha](insert\ i\ Vs) (insert\ i\ Vs)\ Fi))) (i'::=Var\ sn')$ 
  using atoms IH
  by (force intro!: Subst elim!: fresh_quote_all_mem)
hence  $quote\_all\ p\ Vs$ 
   $\vdash QuoteP (Var\ i) (Var\ sn') IMP$ 
   $(\alpha\ IMP\ Pfp (subst\ i' (Var\ sn') (ssubst\ [\alpha](insert\ i\ Vs) (insert\ i\ Vs)\ Fi)))$ 
  using atoms by simp
moreover have  $subst\ i' (Var\ sn') (ssubst\ [\alpha](insert\ i\ Vs) (insert\ i\ Vs)\ Fi)$ 
   $= ssubst\ [\alpha(i::=Var\ sn)] V' V' F'$ 
  using sp
  by (auto simp: vquot_fm_def perm_v_fm supp_conv_fresh subst_fm_trans_commute [symmetric])
ultimately
have  $quote\_all\ p\ Vs$ 
   $\vdash QuoteP (Var\ i) (Var\ sn') IMP (\alpha\ IMP\ Pfp (ssubst\ [\alpha(i::=Var\ sn)] V' V' F'))$ 
  by simp
hence  $quote\_all\ p\ Vs$ 
   $\vdash (QuoteP (Var\ i) (Var\ sn') IMP (\alpha\ IMP\ Pfp (ssubst\ [\alpha(i::=Var\ sn)] V' V' F')))$  ( $i::=Var\ sn$ )
  using  $\langle atom\ i\ \# \_ \rangle$ 
  by (force intro!: Subst elim!: fresh_quote_all_mem)
hence  $quote\_all\ p\ Vs$ 
   $\vdash (QuoteP (Var\ sn) (Var\ sn') IMP$ 
   $(\alpha(i::=Var\ sn)\ IMP\ Pfp (subst\ i (Var\ sn) (ssubst\ [\alpha(i::=Var\ sn)] V' V' F'))))$ 
  using atoms atoms' by simp
moreover have  $subst\ i (Var\ sn) (ssubst\ [\alpha(i::=Var\ sn)] V' V' F')$ 
   $= ssubst\ [\alpha(i::=Var\ sn)] V' V' F'$ 
  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
  ⊢ QuoteP (Var sn) (Var sn') IMP (α(i::=Var sn) IMP PfP (ssubst [α(i::=Var sn)] V' V' F'))
  using atoms atoms' by simp
hence star0: insert (QuoteP (Var sn) (Var sn')) (quote_all p Vs)
  ⊢ α(i::=Var sn) IMP PfP (ssubst [α(i::=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_p Vs
  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_p Vs)
  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_noprimes j')
    apply (rule cut_same [where A = All2 i (Eats (Var sm) (Var sn)) α])
    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) α] V' V' F')
    apply (force intro!: Imp_E [THEN rotate3] simp add: vquot_fm_def substj j' ss_noprimes)
    — Proving Pfp (ssubst [α(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}))
  ⊢ (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))) (j::=Var j)

```

```

by (metis All_D)
have insert (OrdP (Var k)) (quote_all p (Vs-{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)) (j' ::= Var j)
apply (rule All_D)
using p1 atoms by simp
thus ?thesis
using atoms
by simp (metis SeqQuoteP_imp_OrdP Imp_cut anti_deduction)
qed

lemma (in quote_perm) quote_all_Mem_imp_All2:
  assumes IH: insert (QuoteP (Var i) (Var i')) (quote_all p Vs)
    ⊢ α IMP Pfp (ssubst [α](insert i Vs) (insert i Vs) Fi)
  and supp (All2 i (Var j) α) ⊆ atom ' Vs
  and j: atom j # (i,α) and i: atom i # p and i': atom i' # (i,p,α)
  and pi: pi = (atom i ≡ atom i') + p
  and Fi: Fi = make_F (insert i Vs) pi
  shows insert (All2 i (Var j) α) (quote_all p Vs) ⊢ Pfp (ssubst [All2 i (Var j) α] Vs Vs F)
proof -
  have sp: supp α - {atom i} ⊆ atom ' Vs and jV: j ∈ Vs
  using assms
  by (auto simp: fresh_def supp_Pair)
  obtain s::name and k::name
  where atoms: atom s # (k,i,j,p·j,α,p) atom k # (i,j,p·j,α,p)
  by (metis obtain_fresh)
  hence ii: atom i # (j, p · j, s, k, p) using i j
  by (simp add: fresh_Pair) (metis fresh_at_base(2) fresh_perm fresh_permute_left pinv)
  have jj: atom j # (p · j, s, k, α) using atoms j
  by (auto simp: fresh_Pair) (metis atom_fresh_perm jV)
  have pj: atom (p · j) # (s, k, α) using atoms ii sp jV
  by (simp add: fresh_Pair) (auto simp: fresh_def perm_exits_Vs dest!: subsetD)
  show ?thesis
  apply (rule cut_same [where A = QuoteP (Var j) (Var (p · j))])
  apply (force intro: jV Hyp simp add: quote_all_def)
  using atoms
  apply (auto simp: QuoteP.simps [of s _ _ k] elim!: fresh_quote_all_mem)
  apply (rule MP_same)
  apply (rule SeqQuoteP_Mem_imp_All2 [OF IH sp jV refl pi Fi ii i' jj pj, THEN thin])
  apply (auto simp: fresh_at_base_permI quote_all_def intro!: fresh_ssubst')
  done
qed

```

9.6 The Derivability Condition, Theorem 9.1

```

lemma SpecI: H ⊢ A IMP Ex i A
  by (metis Imp_I Assume Ex_I subst_fm_id)

```

```

lemma star:
  fixes p :: perm and F :: name ⇒ tm
  assumes C: ss_fm α
  and p: atom ' (p · V) #* V -p = p
  and V: finite V supp α ⊆ atom ' V
  and F: F = make_F V p
  shows insert α (quote_all p V) ⊢ Pfp (ssubst [α] V V F)
using C V p F
proof (nominal_induct avoiding: p arbitrary: V F rule: ss_fm.strong_induct)
  case (MemI i j) show ?case

```

```

proof (cases i=j)
  case True thus ?thesis
    by auto
next
  case False
  hence ij: atom i # j {i, j} ⊆ V using MemI
    by auto
  interpret qp: quote_perm p V F
    by unfold_locales (auto simp: image_iff F make_F_def p MemI)
  have insert (Var i IN Var j) (quote_all p V) ⊢ Pfp (Q_Mem (Var (p · i)) (Var (p · j)))
    apply (rule QuoteP_Mem_imp_QMem [of i j, THEN cut3])
    using ij apply (auto simp: quote_all_def qp.atom_fresh_perm intro: Hyp)
    apply (metis atom_eqvt fresh_Pair fresh_at_base(2) fresh_permute_iff qp.atom_fresh_perm)
    done
  thus ?thesis
    apply (simp add: vquot_fm_def)
    using MemI apply (auto simp: make_F_def)
    done
qed
next
  case (DisjI A B)
    interpret qp: quote_perm p V F
      by unfold_locales (auto simp: image_iff DisjI)
    show ?case
      apply auto
      apply (rule_tac [2] qp.quote_all_Disj_I2_Pfp_ssubst)
      apply (rule qp.quote_all_Disj_I1_Pfp_ssubst)
      using DisjI by auto
next
  case (ConjI A B)
    interpret qp: quote_perm p V F
      by unfold_locales (auto simp: image_iff ConjI)
    show ?case
      apply (rule qp.quote_all_Conj_I_Pfp_ssubst)
      using ConjI by (auto intro: thin1 thin2)
next
  case (ExI A i)
    interpret qp: quote_perm p V F
      by unfold_locales (auto simp: image_iff ExI)
    obtain i'::name where i': atom i' # (i,p,A)
      by (metis obtain_fresh)
    define p' where p' = (atom i ⇒ atom i') + p
    define F' where 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))
   $\vdash$  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))
   $\vdash$  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)  $\subseteq$   $\_$  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))
   $\vdash$  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))
   $\vdash$  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)  $\vdash$  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)  $\vdash$  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)
define p' where p' = (atom i  $\equiv$  atom i') + p
define F' where 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))
   $\vdash$  PfP (ssubst [A](insert i V) (insert i V) F')
  apply (rule All2I.hyps)
  using (supp (All2 i _ A)  $\subseteq$   $\_$  qp'.p qp'.pinv)
  apply (auto simp: F'_def fresh_finite_insert)
  done
hence insert (QuoteP (Var i) (Var i')) (quote_all p V)
   $\vdash$  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)  $\vdash$  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  $\alpha$  ground_fm  $\alpha$ 
  shows  $\{\alpha\} \vdash \text{PfP } \langle\langle \alpha \rangle\rangle$ 
proof –
  obtain  $\beta$  where  $\beta: \text{ss\_fm } \beta \text{ ground\_fm } \beta \ \{\} \vdash \alpha \text{ IFF } \beta$  using assms
    by (auto simp: Sigma_fm_def ground_fm_aux_def)
  hence  $\{\beta\} \vdash \text{PfP } \langle\langle \beta \rangle\rangle$  using star [of  $\beta$  0  $\{\}$ ]
    by (auto simp: ground_fm_aux_def fresh_star_def)
  then have  $\{\alpha\} \vdash \text{PfP } \langle\langle \beta \rangle\rangle$  using  $\beta$ 
    by (metis Iff_MP_left')
  moreover have  $\{\} \vdash \text{PfP } \langle\langle \beta \text{ IMP } \alpha \rangle\rangle$  using  $\beta$ 
    by (metis Conj_E2 Iff_def proved_imp_proved_PfP)
  ultimately show ?thesis
    by (metis PfP_implies_ModPon_PfP_quot thin0)
qed

end

```

Chapter 10

Uniqueness Results: Syntactic Relations are Functions

```
theory Functions
imports Coding_Predicates
begin
```

10.0.1 SeqStTermP

```
lemma not_IndP_VarP: {IndP x, VarP x}  $\vdash$  A
proof -
  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 }
    ⊢ ((t EQ v AND u EQ i) OR
      ((IndP t OR t NEQ v) AND u EQ t)) OR
      Ex m (Ex n (Ex sm (Ex sm' (Ex sn (Ex sn' (Var m IN k AND Var n IN k AND
        SeqStTermP v i (Var sm) (Var sm') s (Var m) AND
        SeqStTermP v i (Var sn) (Var sn') s (Var n) AND
        t EQ Q_Eats (Var sm) (Var sn) AND
        u EQ Q_Eats (Var sm') (Var sn'))))))))
proof –
obtain l::name and sl::name and sl'::name
  where atom l # (v,i,t,u,s,k,sl,sl',m,n,sm,sm',sn,sn')
    atom sl # (v,i,t,u,s,k,sl',m,n,sm,sm',sn,sn')
    atom sl' # (v,i,t,u,s,k,m,n,sm,sm',sn,sn')
  by (metis obtain_fresh)
thus ?thesis using assms
apply (simp add: SeqStTermP.simps [of l s k v i sl sl' m n sm sm' sn sn'])
apply (rule Conj_EH Ex_EH All2_SUCC_E [THEN rotate2] | simp)+
apply (rule cut_same [where A = HPair t u EQ HPair (Var sl) (Var sl')])
apply (metis Assume AssumeH(4) LstSeqP_EQ)
apply clarify
apply (rule Disj_EH)
apply (rule Disj_I1)
apply (rule anti_deduction)
apply (rule Var_Eq_subst_Iff [THEN Sym_L, THEN Iff_MP_same])
apply (rule Sym_L [THEN rotate2])
apply (rule Var_Eq_subst_Iff [THEN Iff_MP_same], force)
  — now the quantified case
  — auto could be used but is VERY SLOW
apply (rule Ex_EH Conj_EH)+
apply simp_all
apply (rule Disj_I2)
apply (rule Ex_I [where x = Var m], simp)
apply (rule Ex_I [where x = Var n], simp)
apply (rule Ex_I [where x = Var sm], simp)
apply (rule Ex_I [where x = Var sm'], simp)
apply (rule Ex_I [where x = Var sn], simp)
apply (rule Ex_I [where x = Var sn'], simp)
apply (simp_all add: SeqStTermP.simps [of l s _ v i sl sl' m n sm sm' sn sn'])
apply ((rule Conj_I)+, blast intro: LstSeqP_Mem)+
  — first SeqStTermP subgoal
apply (rule All2_Subset [OF Hyp], blast)
apply (blast intro!: SUCC_Subset_Ord LstSeqP_OrdP, blast, simp)
  — next SeqStTermP subgoal
apply ((rule Conj_I)+, blast intro: LstSeqP_Mem)+
apply (rule All2_Subset [OF Hyp], blast)
apply (blast intro!: SUCC_Subset_Ord LstSeqP_OrdP, blast, simp)
  — finally, the equality pair
apply (blast intro: Trans)
done
qed

lemma SeqStTermP_unique: {SeqStTermP v a t u s kk, SeqStTermP v a t u' s' kk'} ⊢ 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)
by (metis obtain_fresh)
have { OrdP (Var k), VarP v }
      ⊢ All i (All j (All j' (All k' (SeqStTermP v a (Var i) (Var j) s (Var k)
        IMP (SeqStTermP v a (Var i) (Var j') s' (Var k') IMP Var j' EQ Var j))))))
apply (rule OrdIndH [where j=l])
using atoms apply auto
apply (rule Swap)
apply (rule cut_same)
apply (rule cut1 [OF SeqStTermP_lemma [of m v a Var i Var j s Var k n sm sm' sn sn']], simp_all,
blast)
apply (rule cut_same)
apply (rule cut1 [OF SeqStTermP_lemma [of m2 v a Var i Var j' s' Var k' n2 sm2 sm2' sn2 sn2']],
simp_all, blast)
apply (rule Disj_EH Conj_EH)+
— case 1, both sides equal "v"
apply (blast intro: Trans Sym)
— case 2, Var i EQ v and also IndP (Var i) OR Var i NEQ v
apply (rule Conj_EH Disj_EH)+
apply (blast intro: IndP_cong [THEN Iff_MP_same] not_IndP_VarP [THEN cut2])
apply (metis Assume OrdNotEqP_E)
— case 3, both a variable and a pair
apply (rule Ex_EH Conj_EH)+
apply simp_all
apply (rule cut_same [where A = VarP (Q_Eats (Var sm) (Var sn))])
apply (blast intro: Trans Sym VarP_cong [where x=v, THEN Iff_MP_same] Hyp, blast)
— towards remaining cases
apply (rule Disj_EH Ex_EH)+
— case 4, Var i EQ v and also IndP (Var i) OR Var i NEQ v
apply (blast intro: IndP_cong [THEN Iff_MP_same] not_IndP_VarP [THEN cut2] OrdNotEqP_E)
— case 5, Var i EQ v for both
apply (blast intro: Trans Sym)
— case 6, both an atom and a pair
apply (rule Ex_EH Conj_EH)+
apply simp_all
apply (rule atom_HPairE)
apply (simp add: HTuple.simps)
apply (blast intro: Trans)
— towards remaining cases
apply (rule Conj_EH Disj_EH Ex_EH)+
apply simp_all
— case 7, both an atom and a pair
apply (rule cut_same [where A = VarP (Q_Eats (Var sm2) (Var sn2))])
apply (blast intro: Trans Sym VarP_cong [where x=v, THEN Iff_MP_same] Hyp, blast)
— case 8, both an atom and a pair
apply (rule Ex_EH Conj_EH)+
apply simp_all
apply (rule atom_HPairE)

```

```

apply (simp add: HTuple.simps)
apply (blast intro: Trans)
— case 9, two Eats terms
apply (rule Ex_EH Disj_EH Conj_EH)+
apply simp_all
apply (rule All_E' [OF Hyp, where x=Var m], blast)
apply (rule All_E' [OF Hyp, where x=Var n], blast, simp)
apply (rule Disj_EH, blast intro: thin1 ContraProve)+
apply (rule All_E [where x=Var sm], simp)
apply (rule All_E [where x=Var sm'], simp)
apply (rule All_E [where x=Var sm2'], simp)
apply (rule All_E [where x=Var m2], simp)
apply (rule All_E [where x=Var sn, THEN rotate2], simp)
apply (rule All_E [where x=Var sn'], simp)
apply (rule All_E [where x=Var sn2'], simp)
apply (rule All_E [where x=Var n2], simp)
apply (rule cut_same [where A = Q_Eats (Var sm) (Var sn) EQ Q_Eats (Var sm2) (Var sn2)])
apply (blast intro: Sym Trans, clarify)
apply (rule cut_same [where A = SeqStTermP v a (Var sn) (Var sn2') s' (Var n2)])
apply (blast intro: Hyp SeqStTermP_cong [OF Hyp Refl Refl, THEN Iff_MP2_same])
apply (rule cut_same [where A = SeqStTermP v a (Var sm) (Var sm2') s' (Var m2)])
apply (blast intro: Hyp SeqStTermP_cong [OF Hyp Refl Refl, THEN Iff_MP2_same])
apply (rule Disj_EH, blast intro: thin1 ContraProve)+
apply (blast intro: HPair_cong Trans [OF Hyp Sym])
done
hence p1: {OrdP (Var k), VarP v}
  ⊢ (All j (All j' (All k' (SeqStTermP v a (Var i) (Var j) s (Var k)
    IMP (SeqStTermP v a (Var i) (Var j') s' (Var k') IMP Var j' EQ Var j))))(i::=t)
  by (metis All_D)
have p2: {OrdP (Var k), VarP v}
  ⊢ (All j' (All k' (SeqStTermP v a t (Var j) s (Var k)
    IMP (SeqStTermP v a t (Var j') s' (Var k') IMP Var j' EQ Var j))))(j::=u)
  apply (rule All_D)
  using atoms p1 by simp
have p3: {OrdP (Var k), VarP v}
  ⊢ (All k' (SeqStTermP v a t u s (Var k) IMP (SeqStTermP v a t (Var j') s' (Var k') IMP Var
j' EQ u)))(j::=u')
  apply (rule All_D)
  using atoms p2 by simp
have p4: {OrdP (Var k), VarP v}
  ⊢ (SeqStTermP v a t u s (Var k) IMP (SeqStTermP v a t u' s' (Var k') IMP u' EQ u))(k::=kk')
  apply (rule All_D)
  using atoms p3 by simp
hence {SeqStTermP v a t u s (Var k), VarP v} ⊢ SeqStTermP v a t u s (Var k) IMP (SeqStTermP v a
t u' s' kk' IMP u' EQ u)
  using atoms apply simp
  by (metis SeqStTermP_imp_OrdP rcut1)
hence {VarP v} ⊢ ((SeqStTermP v a t u s (Var k) IMP (SeqStTermP v a t u' s' kk' IMP u' EQ u))
by (metis Assume MP_same Imp_I)
hence {VarP v} ⊢ ((SeqStTermP v a t u s (Var k) IMP (SeqStTermP v a t u' s' kk' IMP u' EQ
u)))(k::=kk)
  using atoms by (force intro!: Subst)
hence {VarP v} ⊢ SeqStTermP v a t u s kk IMP (SeqStTermP v a t u' s' kk' IMP u' EQ u)
  using atoms by simp
hence {SeqStTermP v a t u s kk} ⊢ SeqStTermP v a t u s kk IMP (SeqStTermP v a t u' s' kk' IMP u'
EQ u)
  by (metis SeqStTermP_imp_VarP rcut1)
thus ?thesis

```

by (metis Assume AssumeH(2) MP_same rcut1)
qed

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

obtain $s::name$ **and** $s'::name$ **and** $k::name$ **and** $k'::name$
where $atom\ s\ \# (v,tm,t,u,u',k,k')$ $atom\ s'\ \# (v,tm,t,u,u',k,k',s)$
 $atom\ k\ \# (v,tm,t,u,u')$ $atom\ k'\ \# (v,tm,t,u,u',k)$
by (metis obtain_fresh)
thus ?thesis
by (auto simp: SubstTermP.simps [of s v tm t u k] SubstTermP.simps [of s' v tm t u' k'])
(metis SeqStTermP_unique rotate3 thin1)

qed

10.0.2 SubstAtomicP

lemma *SubstTermP_eq*:

$\llbracket H \vdash SubstTermP\ v\ tm\ x\ z;\ insert\ (SubstTermP\ v\ tm\ y\ z)\ H \vdash A \rrbracket \implies insert\ (x\ EQ\ y)\ H \vdash A$
by (metis Assume rotate2 Iff_E1 cut_same thin1 SubstTermP_cong [OF Refl Refl _ Refl])

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

obtain $t::name$ **and** $ts::name$ **and** $u::name$ **and** $us::name$
and $t'::name$ **and** $ts'::name$ **and** $u'::name$ **and** $us'::name$
where $atom\ t\ \# (v,tm,x,y,y',ts,u,us)$ $atom\ ts\ \# (v,tm,x,y,y',u,us)$
 $atom\ u\ \# (v,tm,x,y,y',us)$ $atom\ us\ \# (v,tm,x,y,y')$
 $atom\ t'\ \# (v,tm,x,y,y',t,ts,u,us,ts',u',us')$ $atom\ ts'\ \# (v,tm,x,y,y',t,ts,u,us,u',us')$
 $atom\ u'\ \# (v,tm,x,y,y',t,ts,u,us,us')$ $atom\ us'\ \# (v,tm,x,y,y',t,ts,u,us)$
by (metis obtain_fresh)
thus ?thesis
apply (simp add: SubstAtomicP.simps [of t v tm x y ts u us]
SubstAtomicP.simps [of t' v tm x y' ts' u' us'])
apply (rule Ex_EH Disj_EH Conj_EH)+
apply simp_all
apply (rule Eq_Trans_E [OF Hyp], auto simp: HTS)
apply (rule SubstTermP_eq [THEN thin1], blast)
apply (rule SubstTermP_eq [THEN rotate2], blast)
apply (rule Trans [OF Hyp Sym], blast)
apply (rule Trans [OF Hyp], blast)
apply (metis Assume AssumeH(8) HPair_cong Refl cut2 [OF SubstTermP_unique] thin1)
apply (rule Eq_Trans_E [OF Hyp], blast, force simp add: HTS)
apply (rule Eq_Trans_E [OF Hyp], blast, force simp add: HTS)
apply (rule Eq_Trans_E [OF Hyp], auto simp: HTS)
apply (rule SubstTermP_eq [THEN thin1], blast)
apply (rule SubstTermP_eq [THEN rotate2], blast)
apply (rule Trans [OF Hyp Sym], blast)
apply (rule Trans [OF Hyp], blast)
apply (metis Assume AssumeH(8) HPair_cong Refl cut2 [OF SubstTermP_unique] thin1)
done

qed

10.0.3 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 EQ Q_Neg z, SubstAtomicP v tm y y'} † Fls (is ?thesis1)
    and Disj_SubstAtomicP_Fls: {y EQ Q_Disj z w, SubstAtomicP v tm y y'} † Fls (is ?thesis2)
    and Ex_SubstAtomicP_Fls: {y EQ Q_Ex z, SubstAtomicP v tm y y'} † Fls (is ?thesis3)
  proof –
    obtain t::name and u::name and t':name and u':name
      where atom t ‡ (z,w,v,tm,y,y',t',u,u') atom t' ‡ (z,w,v,tm,y,y',u,u')
            atom u ‡ (z,w,v,tm,y,y',u') atom u' ‡ (z,w,v,tm,y,y')
    by (metis obtain_fresh)
  
```


thus *?thesis1 ?thesis2 ?thesis3*
by (*auto simp: SubstAtomicP.simps [of t v tm y y' t' u u'] HTS intro: Eq_Trans_E [OF Hyp]*)
qed

lemma *SeqSubstFormP_eq*:
 $\llbracket H \vdash \text{SeqSubstFormP } v \text{ tm } x \text{ z } s \text{ k}; \text{ insert } (\text{SeqSubstFormP } v \text{ tm } y \text{ z } s \text{ k}) H \vdash A \rrbracket$
 $\implies \text{insert } (x \text{ EQ } y) H \vdash A$
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*: $\{\text{SeqSubstFormP } v \text{ a } x \text{ y } s \text{ kk}, \text{SeqSubstFormP } v \text{ a } x \text{ y' } s' \text{ kk}'\} \vdash y' \text{ EQ } y$
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*: $\text{atom } i \# (s, s', v, a, x, y, y')$ $\text{atom } j \# (s, s', v, a, x, i, x, y, y')$
 $\text{atom } j' \# (s, s', v, a, x, i, j, x, y, y')$
 $\text{atom } k \# (s, s', v, a, x, y, y', k, k', i, j, j')$ $\text{atom } k' \# (s, s', v, a, x, y, y', k, i, j, j')$
 $\text{atom } l \# (s, s', v, a, x, i, j, j', k, k')$
 $\text{atom } m \# (s, s', v, a, i, j, j', k, k', l)$ $\text{atom } n \# (s, s', v, a, i, j, j', k, k', l, m)$
 $\text{atom } sm \# (s, s', v, a, i, j, j', k, k', l, m, n)$ $\text{atom } sn \# (s, s', v, a, i, j, j', k, k', l, m, n, sm)$
 $\text{atom } sm' \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn)$ $\text{atom } sn' \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm')$
 $\text{atom } m2 \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm', sn')$ $\text{atom } n2 \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2)$
 $\text{atom } sm2 \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2, n2)$ $\text{atom } sn2 \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2, n2, sn2)$
 $\text{atom } sm2' \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2, n2, sm2, sn2)$ $\text{atom } sn2' \# (s, s', v, a, i, j, j', k, k', l, m, n, sm, sn, sm', sn', m2, n2, sm2, sn2, sm2')$
by (*metis obtain_fresh*)
have $\{ \text{OrdP } (\text{Var } k) \}$
 $\vdash \text{All } i \text{ (All } j \text{ (All } j' \text{ (All } k' \text{ (SeqSubstFormP } v \text{ a } (\text{Var } i) \text{ (Var } j) \text{ s } (\text{Var } k)$
 $\text{IMP } (\text{SeqSubstFormP } v \text{ a } (\text{Var } i) \text{ (Var } j') \text{ s' } (\text{Var } k') \text{ IMP } \text{Var } j' \text{ EQ } \text{Var } j))))))$
apply (*rule OrdIndH [where j=l]*)
using *atoms apply auto*
apply (*rule Swap*)
apply (*rule cut_same*)
apply (*rule cut1 [OF SeqSubstFormP_lemma [of m v a Var i Var j s Var k n sm sm' sn sn'], simp_all, blast]*)
apply (*rule cut_same*)
apply (*rule cut1 [OF SeqSubstFormP_lemma [of m2 v a Var i Var j' s' Var k' n2 sm2 sm2' sn2 sn2'], simp_all, blast]*)
apply (*rule Disj_EH Conj_EH*) +
— case 1, both sides are atomic
apply (*blast intro: cut2 [OF SubstAtomicP_unique]*)
— case 2, atomic and also not
apply (*rule Ex_EH Conj_EH Disj_EH*) +
apply *simp_all*
apply (*metis Assume AssumeH(7) Disj_I1 Neg_I anti_deduction cut2 [OF Disj_SubstAtomicP_Fls]*)
apply (*rule Conj_EH Disj_EH*) +
apply (*metis Assume AssumeH(7) Disj_I1 Neg_I anti_deduction cut2 [OF Neg_SubstAtomicP_Fls]*)
apply (*rule Conj_EH*) +
apply (*metis Assume AssumeH(7) Disj_I1 Neg_I anti_deduction cut2 [OF Ex_SubstAtomicP_Fls]*)
— towards remaining cases
apply (*rule Conj_EH Disj_EH Ex_EH*) +
apply *simp_all*
apply (*metis Assume AssumeH(7) Disj_I1 Neg_I anti_deduction cut2 [OF Disj_SubstAtomicP_Fls]*)
apply (*rule Conj_EH Disj_EH*) +
apply (*metis Assume AssumeH(7) Disj_I1 Neg_I anti_deduction cut2 [OF Neg_SubstAtomicP_Fls]*)
apply (*rule Conj_EH*) +

```

apply (metis Assume AssumeH(7) Disj_I1 Neg_I anti_deduction cut2 [OF Ex_SubstAtomicP_Fls])
— towards remaining cases
apply (rule Conj_EH Disj_EH Ex_EH) +
apply simp_all
— case two Disj terms
apply (rule All_E' [OF Hyp, where x=Var m], blast)
apply (rule All_E' [OF Hyp, where x=Var n], blast, simp)
apply (rule Disj_EH, blast intro: thin1 ContraProve) +
apply (rule All_E [where x=Var sm], simp)
apply (rule All_E [where x=Var sm1], 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 sn1], simp)
apply (rule All_E [where x=Var sn2], simp)
apply (rule All_E [where x=Var n2], simp)
apply (rule rotate3)
apply (rule Eq_Trans_E [OF Hyp], blast)
apply (clarsimp simp add: HTS)
apply (rule thin1)
apply (rule Disj_EH [OF ContraProve], blast intro: thin1 SeqSubstFormP_eq) +
apply (blast intro: HPair_cong Trans [OF Hyp Sym])
— towards remaining cases
apply (rule Conj_EH Disj_EH) +
— Negation = Disjunction?
apply (rule Eq_Trans_E [OF Hyp], blast, force simp add: HTS)
— Existential = Disjunction?
apply (rule Conj_EH)
apply (rule Eq_Trans_E [OF Hyp], blast, force simp add: HTS)
— towards remaining cases
apply (rule Conj_EH Disj_EH Ex_EH) +
apply simp_all
— Disjunction = Negation?
apply (rule Eq_Trans_E [OF Hyp], blast, force simp add: HTS)
apply (rule Conj_EH Disj_EH) +
— case two Neg terms
apply (rule Eq_Trans_E [OF Hyp], blast, clarify)
apply (rule thin1)
apply (rule All_E' [OF Hyp, where x=Var m], blast, simp)
apply (rule Disj_EH, blast intro: thin1 ContraProve) +
apply (rule All_E [where x=Var sm], simp)
apply (rule All_E [where x=Var sm1], 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  $x = \text{Var } m$ ], 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  $x = \text{Var } sm^1$ ], simp)
apply (rule All_E [where  $x = \text{Var } sm^2$ ], simp)
apply (rule All_E [where  $x = \text{Var } m^2$ ], simp)
apply (rule Disj_EH [OF ContraProve], blast intro: SeqSubstFormP_eq Sym_L) +
apply (blast intro: HPair_cong Sym Trans [OF Hyp])
done
hence  $p1: \{ \text{OrdP } (\text{Var } k) \}$ 
   $\vdash ( \text{All } j ( \text{All } j' ( \text{All } k' ( \text{SeqSubstFormP } v a (\text{Var } i) (\text{Var } j) s (\text{Var } k)$ 
     $\text{IMP } ( \text{SeqSubstFormP } v a (\text{Var } i) (\text{Var } j') s' (\text{Var } k') \text{IMP } \text{Var } j' \text{EQ } \text{Var } j) ) ) ) ) (i ::= x)$ 
  by (metis All_D)
have  $p2: \{ \text{OrdP } (\text{Var } k) \}$ 
   $\vdash ( \text{All } j' ( \text{All } k' ( \text{SeqSubstFormP } v a x (\text{Var } j) s (\text{Var } k)$ 
     $\text{IMP } ( \text{SeqSubstFormP } v a x (\text{Var } j') s' (\text{Var } k') \text{IMP } \text{Var } j' \text{EQ } \text{Var } j) ) ) ) (j ::= y)$ 
  apply (rule All_D)
  using atoms p1 by simp
have  $p3: \{ \text{OrdP } (\text{Var } k) \}$ 
   $\vdash ( \text{All } k' ( \text{SeqSubstFormP } v a x y s (\text{Var } k)$ 
     $\text{IMP } ( \text{SeqSubstFormP } v a x (\text{Var } j') s' (\text{Var } k') \text{IMP } \text{Var } j' \text{EQ } y) ) ) (j' ::= y')$ 
  apply (rule All_D)
  using atoms p2 by simp
have  $p4: \{ \text{OrdP } (\text{Var } k) \}$ 
   $\vdash ( \text{SeqSubstFormP } v a x y s (\text{Var } k) \text{IMP } ( \text{SeqSubstFormP } v a x y' s' (\text{Var } k') \text{IMP } y' \text{EQ}$ 
 $y) ) (k' ::= kk')$ 
  apply (rule All_D)
  using atoms p3 by simp
hence  $\{ \text{OrdP } (\text{Var } k) \} \vdash \text{SeqSubstFormP } v a x y s (\text{Var } k) \text{IMP } ( \text{SeqSubstFormP } v a x y' s' kk' \text{IMP}$ 
 $y' \text{EQ } y)$ 
  using atoms by simp
hence  $\{ \text{SeqSubstFormP } v a x y s (\text{Var } k) \}$ 
   $\vdash \text{SeqSubstFormP } v a x y s (\text{Var } k) \text{IMP } ( \text{SeqSubstFormP } v a x y' s' kk' \text{IMP } y' \text{EQ } y)$ 
  by (metis SeqSubstFormP_imp_OrdP rcut1)
hence  $\{ \} \vdash \text{SeqSubstFormP } v a x y s (\text{Var } k) \text{IMP } ( \text{SeqSubstFormP } v a x y' s' kk' \text{IMP } y' \text{EQ } y)$ 
  by (metis Assume Disj_Neg_2 Disj_commute anti_deduction Imp_I)
hence  $\{ \} \vdash ( ( \text{SeqSubstFormP } v a x y s (\text{Var } k) \text{IMP } ( \text{SeqSubstFormP } v a x y' s' kk' \text{IMP } y' \text{EQ}$ 
 $y) ) ) (k ::= kk)$ 
  using atoms by (force intro!: Subst)
  thus ?thesis
  using atoms by simp (metis DisjAssoc2 Disj_commute anti_deduction)
qed

```

10.0.4 *SubstFormP*

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

proof –

```

obtain  $s :: \text{name}$  and  $s' :: \text{name}$  and  $k :: \text{name}$  and  $k' :: \text{name}$ 
  where  $\text{atom } s \# (v, \text{tm}, x, y, y', k, k')$   $\text{atom } s' \# (v, \text{tm}, x, y, y', k, k', s)$ 
   $\text{atom } k \# (v, \text{tm}, x, y, y')$   $\text{atom } k' \# (v, \text{tm}, x, y, y', k)$ 
  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 11

Section 6 Material and Gödel's First Incompleteness Theorem

```
theory Goedel_I
imports Pf_Predicates Functions II_Prelims
begin
```

11.1 The Function W and Lemma 6.1

11.1.1 Predicate form, defined on sequences

```
nominal_function SeqWRP :: tm  $\Rightarrow$  tm  $\Rightarrow$  tm  $\Rightarrow$  fm
  where  $\llbracket$ atom l  $\#$  (s,k,sl); atom sl  $\#$  (s) $\rrbracket \Longrightarrow$ 
    SeqWRP s k y = LstSeqP s k y AND
      HPair Zero Zero IN s AND
      All2 l k (Ex sl (HPair (Var l) (Var sl) IN s AND
        HPair (SUCC (Var l)) (Q_Succ (Var sl)) IN s))
  by (auto simp: eqvt_def SeqWRP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)
```

```
nominal_termination (eqvt)
  by lexicographic_order
```

lemma

```
shows SeqWRP_fresh_iff [simp]: a  $\#$  SeqWRP s k y  $\longleftrightarrow$  a  $\#$  s  $\wedge$  a  $\#$  k  $\wedge$  a  $\#$  y (is ?thesis1)
  and SeqWRP_sf [iff]: Sigma_fm (SeqWRP s k y) (is ?thsf)
  and SeqWRP_imp_OrdP: {SeqWRP s k t}  $\vdash$  OrdP k (is ?thOrd)
  and SeqWRP_LstSeqP: {SeqWRP s k t}  $\vdash$  LstSeqP s k t (is ?thlstseq)
```

proof –

```
  obtain l::name and sl::name where atom l  $\#$  (s,k,sl) atom sl  $\#$  (s)
  by (metis obtain_fresh)
  thus ?thesis1 ?thsf ?thOrd ?thlstseq
  by (auto intro: LstSeqP_OrdP[THEN cut1])
```

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 $H \vdash s \text{ EQ } s'$ **and** $H \vdash k \text{ EQ } k'$ **and** $H \vdash y \text{ EQ } y'$
shows $H \vdash \text{SeqWRP } s \ k \ y \text{ IFF } \text{SeqWRP } s' \ k' \ y'$
by (*rule P3_cong [OF _ assms]*, *auto*)

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

11.1.2 Predicate form of W

nominal_function *WRP* :: $tm \Rightarrow tm \Rightarrow fm$

where $\llbracket atom \ s \ \# \ (x,y) \rrbracket \Longrightarrow$

$WRP \ x \ y = Ex \ s \ (\text{SeqWRP} \ (\text{Var} \ s) \ x \ y)$

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 *sigma_fm_WRP* [*simp*]: $\text{Sigma_fm} \ (WRP \ x \ y)$ (**is** *?thsf*)

proof –

obtain *s::name* **where** $atom \ s \ \# \ (x,y)$

by (*metis obtain_fresh*)

thus *?thesis1* *?thsf*

by *auto*

qed

lemma *WRP_subst* [*simp*]: $(WRP \ x \ y)(i::=t) = WRP \ (\text{subst} \ i \ t \ x) \ (\text{subst} \ i \ t \ y)$

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*: $H \vdash t \text{ EQ } t' \Longrightarrow H \vdash u \text{ EQ } u' \Longrightarrow H \vdash WRP \ t \ u \text{ IFF } WRP \ t' \ u'$

by (*rule P2_cong*) *auto*

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

lemma *ground_WRP* [*simp*]: $\text{ground_fm} \ (WRP \ x \ y) \longleftrightarrow \text{ground} \ x \wedge \text{ground} \ y$

by (*auto simp: ground_aux_def ground_fm_aux_def supp_conv_fresh*)

lemma *SeqWRP_Zero*: $\{\} \vdash \text{SyntaxN.Ex} \ s \ (\text{SeqWRP} \ (\text{Var} \ s) \ \text{Zero} \ \text{Zero})$

proof –

obtain *l sl :: name* **where** $atom \ l \ \# \ (s, \text{sl}) \wedge atom \ \text{sl} \ \# \ s$ **by** (*metis obtain_fresh*)

then show *?thesis*

apply (*subst SeqWRP.simps[of l _ _ sl]; simp*)

apply (*rule Ex_I[where x=(Eats Zero (HPair Zero Zero))]; simp*)

apply (*auto intro!: Mem_Eats_I2*)

done

qed

lemma *WRP_Zero*: $\{\} \vdash WRP \ \text{Zero} \ \text{Zero}$

by (*subst WRP.simps[of undefined]*) (*auto simp: SeqWRP_Zero*)

```

lemma SeqWRP_HPpair_Zero_Zero: {SeqWRP s k y} ⊢ HPair Zero Zero IN s
proof –
  let ?vs = (s,k,y)
  obtain l::name and sl::name
    where atom l # (?vs,sl) atom sl # (?vs) by (metis obtain_fresh)
  then show ?thesis
    by (subst SeqWRP.simps[of l _ _ sl]) auto
qed

lemma SeqWRP_Succ:
  assumes atom s # (s1,k1,y)
  shows {SeqWRP s1 k1 y} ⊢ SyntaxN.Ex s (SeqWRP (Var s) (SUCC k1) (Q_Succ y))
proof –
  let ?vs = (s,s1,k1,y)
  obtain l::name and sl::name and l1::name and sl1::name
    where atoms:
      atom l # (?vs,sl1,l1,sl)
      atom sl # (?vs,sl1,l1)
      atom l1 # (?vs,sl1)
      atom sl1 # (?vs)
    by (metis obtain_fresh)
  let ?hyp = {RestrictedP s1 (SUCC k1) (Var s), OrdP k1, SeqWRP s1 k1 y}
  show ?thesis
    using assms atoms
    apply (auto simp: SeqWRP.simps [of l Var s _ sl])
    apply (rule cut_same [where A=OrdP k1])
    apply (rule SeqWRP_imp_OrdP)
    apply (rule cut_same [OF exists_RestrictedP [of s s1 SUCC k1]])
    apply (rule AssumeH Ex_EH Conj_EH | simp)+
    apply (rule Ex_I [where x=Eats (Var s) (HPair (SUCC k1) (Q_Succ y))])
    apply (simp_all (no_asm_simp))
    apply (rule Conj_I)
    apply (blast intro: RestrictedP_LstSeqP_Eats[THEN cut2] SeqWRP_LstSeqP[THEN cut1])
    apply (rule Conj_I)
    apply (rule Mem_Eats_I1)
    apply (blast intro: RestrictedP_Mem[THEN cut3] SeqWRP_HPpair_Zero_Zero[THEN cut1] Zero_In_SUCC[THEN
cut1])
  proof (rule All2_SUCC_I, simp_all)
    show ?hyp ⊢ SyntaxN.Ex sl
      (HPair k1 (Var sl) IN Eats (Var s) (HPair (SUCC k1) (Q_Succ y)) AND
       HPair (SUCC k1) (Q_Succ (Var sl)) IN
       Eats (Var s) (HPair (SUCC k1) (Q_Succ y)))
      — verifying the final values
    apply (rule Ex_I [where x=y])
    using assms atoms apply simp
    apply (rule Conj_I[rotated])
    apply (rule Mem_Eats_I2, rule Refl)
    apply (rule Mem_Eats_I1)
    apply (rule RestrictedP_Mem[THEN cut3])
    apply (rule AssumeH)
    apply (simp add: LstSeqP_imp_Mem SeqWRP_LstSeqP thin1)
    apply (rule Mem_SUCC_Refl)
    done
  next
  show ?hyp ⊢ All2 l k1
    (SyntaxN.Ex sl
     (HPair (Var l) (Var sl) IN
      Eats (Var s) (HPair (SUCC k1) (Q_Succ y)) AND

```

```

    HPair (SUCC (Var l)) (Q_Succ (Var sl)) IN
    Eats (Var s) (HPair (SUCC k1) (Q_Succ y)))
— verifying the sequence buildup
apply (rule All_I Imp_I)+
using assms atoms apply simp_all
— ... the sequence buildup via s1
apply (simp add: SeqWRP.simps [of l s1 _ sl])
apply (rule AssumeH Ex_EH Conj_EH)+
apply (rule All2_E [THEN rotate2], auto del: Disj_EH)
apply (rule Ex_I [where x=Var sl], simp)
apply (rule Conj_I)
apply (blast intro: Mem_Eats_I1 [OF RestrictedP_Mem [THEN cut3]] Mem_SUCC_I1)
apply (blast intro: Mem_Eats_I1 [OF RestrictedP_Mem [THEN cut3]] OrdP_IN_SUCC)
done
qed
qed

lemma WRP_Succ: {OrdP i, WRP i y} ⊢ WRP (SUCC i) (Q_Succ y)
proof —
  obtain s t :: name where atom s # (i, y) atom t # (s, i, y) by (metis obtain_fresh)
  then show ?thesis
    by (subst WRP.simps[of s], simp, subst WRP.simps[of t], simp) (force intro: SeqWRP_Succ[THEN
cut1])
qed

lemma WRP: {} ⊢ WRP (ORD_OF i) «ORD_OF i»
  by (induct i)
  (auto simp: WRP_Zero quot_Succ intro!: WRP_Succ[THEN cut2])

lemma prove_WRP: {} ⊢ WRP «Var x» ««Var x»»
  unfolding quot_Var quot_Succ
  by (rule WRP_Succ[THEN cut2]) (auto simp: WRP)



### 11.1.3 Proving that these relations are functions

lemma SeqWRP_Zero_E:
  assumes insert (y EQ Zero) H ⊢ A H ⊢ k EQ Zero
  shows insert (SeqWRP s k y) H ⊢ A
proof —
  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} ⊢ Ex y' (SeqWRP s k (Var y') AND y EQ Q_Succ (Var y'))
proof —
  obtain l::name and sl::name
  where atoms: atom l # (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 ⊢ k' EQ (SUCC k)
  shows insert (SeqWRP s k' y) H ⊢ Ex y' (SeqWRP s k (Var y') AND y EQ Q_Succ (Var y'))
  using SeqWRP_cong [OF Refl k' Refl] cut1 [OF SeqWRP_SUCC_lemma [of y' s k y]]
  by (metis Assume Iff_MP_left Iff_sym y')

lemma SeqWRP_unique: {OrdP x, SeqWRP s x y, SeqWRP s' x y'} ⊢ y' EQ y
proof —
  obtain i::name and j::name and j'::name and k::name and sl::name and sl'::name and l::name and
  pi::name
    where i: atom i # (s,s',y,y') and j: atom j # (s,s',i,x,y,y') and j': atom j' # (s,s',i,j,x,y,y')
      and atoms: atom k # (s,s',i,j,j') atom sl # (s,s',i,j,j',k) atom sl' # (s,s',i,j,j',k,sl)
        atom pi # (s,s',i,j,j',k,sl,sl')
    by (metis obtain_fresh)
  have {OrdP (Var i)} ⊢ All j (All j' (SeqWRP s (Var i) (Var j) IMP (SeqWRP s' (Var i) (Var j') IMP
  Var j' EQ Var j)))
    apply (rule OrdIndH [where j=k])
    using i j j' atoms apply auto
    apply (rule rotate4)
    apply (rule OrdP_cases_E [where k=pi], simp_all)
    — Zero case
    apply (rule SeqWRP_Zero_E [THEN rotate3])
    prefer 2 apply blast
    apply (rule SeqWRP_Zero_E [THEN rotate4])
    prefer 2 apply blast
    apply (blast intro: ContraProve [THEN rotate4] Sym Trans)
    — SUCC case
    apply (rule Ex_I [where x = Var pi], auto)
    apply (metis ContraProve EQ_imp_SUBS2 Mem_SUCC_I2 Refl Subset_D)
    apply (rule cut_same)
    apply (rule SeqWRP_SUCC_E [of sl' s' Var pi, THEN rotate4], auto)
    apply (rule cut_same)
    apply (rule SeqWRP_SUCC_E [of sl s Var pi, THEN rotate7], auto)
    apply (rule All_E [where x = Var sl, THEN rotate5], simp)
    apply (rule All_E [where x = Var sl'], simp)
    apply (rule Imp_E, blast)+
    apply (rule cut_same [OF Q_Succ_cong [OF Assume]])
    apply (blast intro: Trans [OF Hyp Sym] HPair_cong)
  done
  hence {OrdP (Var i)} ⊢ (All j' (SeqWRP s (Var i) (Var j) IMP (SeqWRP s' (Var i) (Var j') IMP Var
  j' EQ Var j)))(j::=y)
    by (metis All_D)
  hence {OrdP (Var i)} ⊢ (SeqWRP s (Var i) y IMP (SeqWRP s' (Var i) (Var j') IMP Var j' EQ
  y))(j'::=y')
    using j j'
    by simp (drule All_D [where x=y'], simp)
  hence {} ⊢ OrdP (Var i) IMP (SeqWRP s (Var i) y IMP (SeqWRP s' (Var i) y' IMP y' EQ y))
    using j j'

```

by *simp* (*metis Imp_I*)
 hence $\{ \} \vdash (\text{OrdP } (\text{Var } i) \text{ IMP } (\text{SeqWRP } s \ (\text{Var } i) \ y \ \text{IMP } (\text{SeqWRP } s' \ (\text{Var } i) \ y' \ \text{IMP } y' \ \text{EQ } y))) (i ::= x)$
 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

11.2 The Function HF and Lemma 6.2

11.2.1 Defining the syntax: quantified body

nominal_function *SeqHRP* :: $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$

where $\llbracket \text{atom } l \# (s, k, sl, sl', m, n, sm, sm', sn, sn');$
 $\text{atom } sl \# (s, sl', m, n, sm, sm', sn, sn');$
 $\text{atom } sl' \# (s, m, n, sm, sm', sn, sn');$
 $\text{atom } m \# (s, n, sm, sm', sn, sn');$
 $\text{atom } n \# (s, sm, sm', sn, sn');$
 $\text{atom } sm \# (s, sm', sn, sn');$
 $\text{atom } sm' \# (s, sn, sn');$
 $\text{atom } sn \# (s, sn');$
 $\text{atom } sn' \# (s) \rrbracket \implies$
 $\text{SeqHRP } x \ x' \ s \ k =$
 $\text{LstSeqP } s \ k \ (\text{HPair } x \ x') \ \text{AND}$
 $\text{All2 } l \ (\text{SUCC } k) \ (\text{Ex } sl \ (\text{Ex } sl' \ (\text{HPair } (\text{Var } l) \ (\text{HPair } (\text{Var } sl) \ (\text{Var } sl')) \ \text{IN } s \ \text{AND}$
 $((\text{OrdP } (\text{Var } sl) \ \text{AND } \text{WRP } (\text{Var } sl) \ (\text{Var } sl')) \ \text{OR}$
 $\text{Ex } m \ (\text{Ex } n \ (\text{Ex } sm \ (\text{Ex } sm' \ (\text{Ex } sn \ (\text{Ex } sn' \ (\text{Var } m \ \text{IN } \text{Var } l \ \text{AND } \text{Var } n \ \text{IN } \text{Var } l \ \text{AND}$
 $\text{HPair } (\text{Var } m) \ (\text{HPair } (\text{Var } sm) \ (\text{Var } sm')) \ \text{IN } s \ \text{AND}$
 $\text{HPair } (\text{Var } n) \ (\text{HPair } (\text{Var } sn) \ (\text{Var } sn')) \ \text{IN } s \ \text{AND}$
 $\text{Var } sl \ \text{EQ } \text{HPair } (\text{Var } sm) \ (\text{Var } sn) \ \text{AND}$
 $\text{Var } sl' \ \text{EQ } \text{Q_HPair } (\text{Var } sm') \ (\text{Var } sn'))))))))$
 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*]:

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

and *SeqHRP_sf* [*iff*]: *Sigma_fm* (*SeqHRP* *x x' s k*) (**is** *?thsf*)

and *SeqHRP_imp_OrdP*: $\{ \text{SeqHRP } x \ y \ s \ k \} \vdash \text{OrdP } k$ (**is** *?thord*)

and *SeqHRP_imp_LstSeqP*: $\{ \text{SeqHRP } x \ y \ s \ k \} \vdash \text{LstSeqP } s \ k \ (\text{HPair } x \ y)$ (**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*:

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

$atom\ sn\ \#(s,sn')$ $atom\ sn'\ \#(s)$
by (*metis obtain_fresh*)
thus *?thesis1 ?thsf ?thord ?thlstseq*
by (*auto intro: LstSeqP_OrdP*)
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*)

11.2.2 Defining the syntax: main predicate

nominal_function *HRP* $::\ tm \Rightarrow tm \Rightarrow fm$
where $\llbracket atom\ s\ \#(x,x',k); atom\ k\ \#(x,x') \rrbracket \Longrightarrow$
 $HRP\ x\ x' = Ex\ s\ (Ex\ k\ (SeqHRP\ x\ x'\ (Var\ s)\ (Var\ k)))$
by (*auto simp: eqvt_def HRP_graph_aux_def flip_fresh_fresh*) (*metis obtain_fresh*)

nominal_termination (*eqvt*)
by *lexicographic_order*

lemma
shows *HRP_fresh_iff [simp]*: $a\ \#(HRP\ x\ x') \longleftrightarrow a\ \#x \wedge a\ \#x'$ (*is ?thesis1*)
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 ?thsf*
by *auto*
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

11.2.3 Proving that these relations are functions

lemma *SeqHRP_lemma*:
assumes $atom\ m\ \#(x,x',s,k,n,sm,sm',sn,sn')$ $atom\ n\ \#(x,x',s,k,sm,sm',sn,sn')$

```

    atom sm # (x,x',s,k,sm',sn,sn') atom sm' # (x,x',s,k,sn,sn')
    atom sn # (x,x',s,k,sn') atom sn' # (x,x',s,k)
shows { SeqHRP x x' s k }
  ⊢ (OrdP x AND WRP x x') OR
    Ex m (Ex n (Ex sm (Ex sm' (Ex sn (Ex sn' (Var m IN k AND Var n IN k AND
      SeqHRP (Var sm) (Var sm') s (Var m) AND
      SeqHRP (Var sn) (Var sn') s (Var n) AND
      x EQ HPair (Var sm) (Var sn) AND
      x' EQ Q_HPair (Var sm') (Var sn'))))))))
proof –
obtain l::name and sl::name and sl'::name
  where atoms:
    atom l # (x,x',s,k,sl,sl',m,n,sm,sm',sn,sn')
    atom sl # (x,x',s,k,sl',m,n,sm,sm',sn,sn')
    atom sl' # (x,x',s,k,m,n,sm,sm',sn,sn')
  by (metis obtain_fresh)
thus ?thesis using atoms assms
apply (simp add: SeqHRP.simps [of l s k sl sl' m n sm sm' sn sn'])
apply (rule Conj_E)
apply (rule All2_SUCC_E' [where t=k, THEN rotate2], simp_all)
apply (rule rotate2)
apply (rule Ex_E Conj_E)+
apply (rule cut_same [where A = HPair x x' EQ HPair (Var sl) (Var sl')])
apply (metis Assume LstSeqP_EQ rotate4, simp_all, clarify)
apply (rule Disj_E [THEN rotate4])
apply (rule Disj_I1)
apply (metis Assume AssumeH(3) Sym thin1 Iff_MP_same [OF Conj_cong [OF OrdP_cong
WRP_cong] Assume])
  — auto could be used but is VERY SLOW
apply (rule Disj_I2)
apply (rule Ex_E Conj_EH)+
apply simp_all
apply (rule Ex_I [where x = Var m], simp)
apply (rule Ex_I [where x = Var n], simp)
apply (rule Ex_I [where x = Var sm], simp)
apply (rule Ex_I [where x = Var sm'], simp)
apply (rule Ex_I [where x = Var sn], simp)
apply (rule Ex_I [where x = Var sn'], simp)
apply (simp add: SeqHRP.simps [of l _ _ sl sl' m n sm sm' sn sn'])
apply (rule Conj_I, blast)+
  — first SeqHRP subgoal
apply (rule Conj_I)+
apply (blast intro: LstSeqP_Mem)
apply (rule All2_Subset [OF Hyp], blast)
apply (blast intro!: SUCC_Subset_Ord LstSeqP_OrdP, blast, simp)
  — next SeqHRP subgoal
apply (rule Conj_I)+
apply (blast intro: LstSeqP_Mem)
apply (rule All2_Subset [OF Hyp], blast)
apply (auto intro!: SUCC_Subset_Ord LstSeqP_OrdP)
  — finally, the equality pair
apply (blast intro: Trans)+
done
qed

lemma SeqHRP_unique: {SeqHRP x y s u, SeqHRP x y' s' u'} ⊢ y' EQ y
proof –
  obtain i::name and j::name and j'::name and k::name and k'::name and l::name

```

```

and  $m::name$  and  $n::name$  and  $sm::name$  and  $sn::name$  and  $sm'::name$  and  $sn'::name$ 
and  $m2::name$  and  $n2::name$  and  $sm2::name$  and  $sn2::name$  and  $sm2'::name$  and  $sn2'::name$ 
where  $atoms$ :  $atom\ i \# (s,s',y,y')$   $atom\ j \# (s,s',i,x,y,y')$   $atom\ j' \# (s,s',i,j,x,y,y')$ 
 $atom\ k \# (s,s',x,y,y',u',i,j,j')$   $atom\ k' \# (s,s',x,y,y',k,i,j,j')$   $atom\ l \# (s,s',i,j,j',k,k')$ 
 $atom\ m \# (s,s',i,j,j',k,k',l)$   $atom\ n \# (s,s',i,j,j',k,k',l,m)$ 
 $atom\ sm \# (s,s',i,j,j',k,k',l,m,n)$   $atom\ sn \# (s,s',i,j,j',k,k',l,m,n,sm)$ 
 $atom\ sm' \# (s,s',i,j,j',k,k',l,m,n,sm,sn)$   $atom\ sn' \# (s,s',i,j,j',k,k',l,m,n,sm,sn,sm')$ 
 $atom\ m2 \# (s,s',i,j,j',k,k',l,m,n,sm,sn,sm',sn')$   $atom\ n2 \# (s,s',i,j,j',k,k',l,m,n,sm,sn,sm',sn',m2)$ 
 $atom\ sm2 \# (s,s',i,j,j',k,k',l,m,n,sm,sn,sm',sn',m2,n2)$   $atom\ sn2 \# (s,s',i,j,j',k,k',l,m,n,sm,sn,sm',sn',m2,n2,sm2)$ 
 $atom\ sm2' \# (s,s',i,j,j',k,k',l,m,n,sm,sn,sm',sn',m2,n2,sm2,sn2)$   $atom\ sn2' \#$ 
 $(s,s',i,j,j',k,k',l,m,n,sm,sn,sm',sn',m2,n2,sm2,sn2,sm2')$ 
by (metis obtain_fresh)
have {OrdP (Var  $k$ )}
   $\vdash$  All  $i$  (All  $j$  (All  $j'$  (All  $k'$  (SeqHRP (Var  $i$ ) (Var  $j$ )  $s$  (Var  $k$ ) IMP (SeqHRP (Var  $i$ ) (Var  $j'$ )  $s'$ 
(Var  $k'$ ) IMP Var  $j'$  EQ Var  $j$ ))))))
apply (rule OrdIndH [where  $j=l$ ])
using  $atoms$  apply auto
apply (rule Swap)
apply (rule cut_same)
apply (rule cut1 [OF SeqHRP_lemma [of  $m$  Var  $i$  Var  $j$   $s$  Var  $k$   $n$   $sm$   $sm'$   $sn$   $sn'$ ], simp_all, blast])
apply (rule cut_same)
apply (rule cut1 [OF SeqHRP_lemma [of  $m2$  Var  $i$  Var  $j'$   $s'$  Var  $k'$   $n2$   $sm2$   $sm2'$   $sn2$   $sn2'$ ], simp_all,
blast])
apply (rule Disj_EH Conj_EH)+
— case 1, both are ordinals
apply (blast intro: cut3 [OF WRP_unique])
— case 2, OrdP (Var  $i$ ) but also a pair
apply (rule Conj_EH Ex_EH)+
apply simp_all
apply (rule cut_same [where  $A = \text{OrdP} (\text{HPair} (\text{Var } sm) (\text{Var } sn))$ ])
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  $A = \text{OrdP} (\text{HPair} (\text{Var } sm2) (\text{Var } sn2))$ ])
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  $x=\text{Var } m$ ], blast)
apply (rule All_E' [OF Hyp, where  $x=\text{Var } n$ ], blast, simp_all)
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  $A = \text{HPair} (\text{Var } sm) (\text{Var } sn) \text{EQ} \text{HPair} (\text{Var } sm2) (\text{Var } sn2)$ ])
apply (blast intro: Sym Trans)
apply (rule cut_same [where  $A = \text{SeqHRP} (\text{Var } sn) (\text{Var } sn2') s' (\text{Var } n2)$ ])
apply (blast intro: SeqHRP_cong [OF Hyp Refl Refl, THEN Iff_MP2_same])
apply (rule cut_same [where  $A = \text{SeqHRP} (\text{Var } sm) (\text{Var } sm2') s' (\text{Var } m2)$ ])
apply (blast intro: SeqHRP_cong [OF Hyp Refl Refl, THEN Iff_MP2_same])
apply (rule Disj_EH, blast intro: thin1 ContraProve)+
apply (blast intro: Trans [OF Hyp Sym] intro!: HPair_cong)
done

```

```

hence {OrdP (Var k)}
  ⊢ 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

```

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

proof –

```

let ?vs = (i)
obtain s k 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 s # (?vs,sl,sl',m,n,sm,sm',sn,sn',l,k)
    atom k # (?vs,sl,sl',m,n,sm,sm',sn,sn',l)
    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)
then show ?thesis
apply (subst HRP.simps[of s _ _ k]; simp)
apply (subst SeqHRP.simps[of l _ _ sl sl' m n sm sm' sn sn']; simp?)
apply (rule Ex_I[where x=Eats Zero (HPair Zero (HPair (ORD_OF i) «ORD_OF i»))]; simp)
apply (rule Ex_I[where x=Zero]; simp)
apply (rule Conj_I[OF LstSeqP_single])
apply (rule All2_SUCC_I, simp)
apply auto [2]

```

```

apply (rule Ex_I[where  $x=ORD\_OF\ i$ ], simp)
apply (rule Ex_I[where  $x\ll ORD\_OF\ i$ ], simp)
apply (auto intro!: Disj_I1 WRP Mem_Eats_I2)
done

```

qed

lemma SeqHRP_HPPair:

```

assumes atom s # (k,s1,s2,k1,k2,x,y,x',y') atom k # (s1,s2,k1,k2,x,y,x',y')
shows {SeqHRP x x' s1 k1,
        SeqHRP y y' s2 k2}
   $\vdash$  Ex s (Ex k (SeqHRP (HPair x y) (Q_HPPair x' y') (Var s) (Var k)))

```

lemma HRP_HPPair: {HRP x x', HRP y y'} \vdash HRP (HPair x y) (Q_HPPair x' y')

proof –

```

obtain k1::name and s1::name and k2::name and s2::name and k::name and s::name
where atom s1 # (x,y,x',y')      atom k1 # (x,y,x',y',s1)
      atom s2 # (x,y,x',y',k1,s1) atom k2 # (x,y,x',y',s2,k1,s1)
      atom s  # (x,y,x',y',k2,s2,k1,s1) atom k  # (x,y,x',y',s,k2,s2,k1,s1)
by (metis obtain_fresh)
thus ?thesis
by (force simp: HRP.simps [of s HPair x y _ k]
      HRP.simps [of s1 x _ k1]
      HRP.simps [of s2 y _ k2]
      intro: SeqHRP_HPPair [THEN cut2])

```

qed

lemma HRP_HPPair_quot: {HRP x «x», HRP y «y»} \vdash HRP (HPair x y) «HPair x y»
using HRP_HPPair[of x «x» y «y»]
unfolding HPair_def quot_simps **by** auto

lemma prove_HRP_coding_tm: **fixes** t::tm **shows** coding_tm t \implies {} \vdash HRP t «t»
by (induct t rule: coding_tm.induct)
 (auto simp: quot_simps HRP_ORD_OF HRP_HPPair_quot[THEN cut2])

lemmas prove_HRP = prove_HRP_coding_tm[OF quot_fm_coding]

11.3 The Function K and Lemma 6.3

nominal_function KRP :: tm \Rightarrow tm \Rightarrow tm \Rightarrow fm

```

where atom y # (v,x,x')  $\implies$ 
      KRP v x x' = Ex y (HRP x (Var y) AND SubstFormP v (Var y) x x')
by (auto simp: eqt_def KRP_graph_aux_def flip_fresh_fresh) (metis obtain_fresh)

```

nominal_termination (eqt)

by lexicographic_order

lemma KRP_fresh_iff [simp]: $a \# KRP\ v\ x\ x' \iff a \# v \wedge a \# x \wedge a \# x'$

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: {} ⊢ SubstFormP «Var i» «A» «A» «A(i::=«A»)»
  using SubstFormP by blast

lemma prove_KRP: {} ⊢ 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'} ⊢ 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» «β» (Var j)} ⊢ Var j EQ «β(i::=«β»)»
  by (metis KRP_unique cut0 prove_KRP)

end

```


Chapter 12

The Instantiation

definition $Fvars\ t = \{a :: name. \neg\ atom\ a\ \#\ t\}$

lemma $Fvars_tm_simps[simp]$:

$Fvars\ Zero = \{\}$
 $Fvars\ (Var\ a) = \{a\}$
 $Fvars\ (Eats\ x\ y) = Fvars\ x \cup Fvars\ y$
by (*auto simp: Fvars_def fresh_at_base(2)*)

lemma $finite_Fvars_tm[simp]$:

fixes $t :: tm$
shows $finite\ (Fvars\ t)$
by (*induct t rule: tm.induct*) *auto*

lemma $Fvars_fm_simps[simp]$:

$Fvars\ (x\ IN\ y) = Fvars\ x \cup Fvars\ y$
 $Fvars\ (x\ EQ\ y) = Fvars\ x \cup Fvars\ y$
 $Fvars\ (A\ OR\ B) = Fvars\ A \cup Fvars\ B$
 $Fvars\ (A\ AND\ B) = Fvars\ A \cup Fvars\ B$
 $Fvars\ (A\ IMP\ B) = Fvars\ A \cup Fvars\ B$
 $Fvars\ Fls = \{\}$
 $Fvars\ (Neg\ A) = Fvars\ A$
 $Fvars\ (Ex\ a\ A) = Fvars\ A - \{a\}$
 $Fvars\ (All\ a\ A) = Fvars\ A - \{a\}$
by (*auto simp: Fvars_def fresh_at_base(2)*)

lemma $finite_Fvars_fm[simp]$:

fixes $A :: fm$
shows $finite\ (Fvars\ A)$
by (*induct A rule: fm.induct*) *auto*

lemma $subst_tm_subst_tm[simp]$:

$x \neq y \implies atom\ x\ \#\ u \implies subst\ y\ u\ (subst\ x\ t\ v) = subst\ x\ (subst\ y\ u\ t)\ (subst\ y\ u\ v)$
by (*induct v rule: tm.induct*) *auto*

lemma $subst_fm_subst_fm[simp]$:

$x \neq y \implies atom\ x\ \#\ u \implies (A(x::=t))(y::=u) = (A(y::=u))(x::=subst\ y\ u\ t)$
by (*nominal_induct A avoiding: x t y u rule: fm.strong_induct*) *auto*

lemma $Fvars_ground_aux: Fvars\ t \subseteq B \implies ground_aux\ t\ (atom\ 'B)$

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

```

lemma ground_Fvars:  $ground\ t \longleftrightarrow Fvars\ t = \{\}$ 
  apply (rule iffI)
  apply (auto simp only: Fvars_def ground_fresh) []
  apply (auto intro: Fvars_ground_aux[of t {}], simplified)
  done

```

```

lemma Fvars_ground_fm_aux:  $Fvars\ A \subseteq B \implies ground\_fm\_aux\ A\ (atom\ 'B)$ 
  apply (induct A arbitrary: B rule: fm.induct)
  apply (auto simp: Diff_subset_conv Fvars_ground_aux)
  apply (drule meta_spec, drule meta_mp, assumption)
  apply auto
  done

```

```

lemma ground_fm_Fvars:  $ground\_fm\ A \longleftrightarrow Fvars\ A = \{\}$ 
  apply (rule iffI)
  apply (auto simp only: Fvars_def ground_fresh) []
  apply (auto intro: Fvars_ground_fm_aux[of A {}], simplified)
  done

```

interpretation *Generic_Syntax* **where**

```

  var = UNIV :: name set
  and trm = UNIV :: tm set
  and fmla = UNIV :: fm set
  and Var = Var
  and FvarsT = Fvars
  and substT =  $\lambda t\ u\ x. subst\ x\ u\ t$ 
  and Fvars = Fvars
  and subst =  $\lambda A\ u\ x. subst\_fm\ A\ x\ u$ 
  apply unfold_locales
  subgoal by simp
  subgoal by simp
  subgoal by simp
  subgoal by simp
  subgoal by simp
  subgoal by simp
  subgoal by simp
  subgoal by simp
  subgoal for t by (induct t rule: tm.induct) auto
  subgoal by simp
  subgoal by simp
  subgoal by simp
  subgoal unfolding Fvars_def fresh_subst_fm_if by auto
  subgoal unfolding Fvars_def by auto
  subgoal unfolding Fvars_def by simp
  subgoal by simp
  subgoal unfolding Fvars_def by simp
  done

```

```

lemma coding_tm_Fvars_empty[simp]:  $coding\_tm\ t \implies Fvars\ t = \{\}$ 
  by (induct t rule: coding_tm.induct) (auto simp: Fvars_def)

```

```

lemma Fvars_empty_ground[simp]:  $Fvars\ t = \{\} \implies ground\ t$ 
  by (induct t rule: tm.induct) auto

```

interpretation *Syntax_with_Numerals* **where**

```

  var = UNIV :: name set
  and trm = UNIV :: tm set
  and fmla = UNIV :: fm set

```

```

and num = {t. ground t}
and Var = Var
and FvarsT = Fvars
and substT =  $\lambda t u x. \text{subst } x u t$ 
and Fvars = Fvars
and subst =  $\lambda A u x. \text{subst\_fm } A x u$ 
apply unfold_locales
subgoal by (auto intro!: exI[of _ Zero])
subgoal by simp
subgoal by (simp add: ground_Fvars)

```

done

declare FvarsT_num[simp del]

interpretation Deduct2_with_False **where**

```

  var = UNIV :: name set
and trm = UNIV :: tm set
and fmla = UNIV :: fm set
and num = {t. ground t}
and Var = Var
and FvarsT = Fvars
and substT =  $\lambda t u x. \text{subst } x u t$ 
and Fvars = Fvars
and subst =  $\lambda A u x. \text{subst\_fm } A x u$ 
and eql = (EQ)
and cnj = (AND)
and imp = (IMP)
and all = All
and exi = Ex
and fls = Fls
and prv = ( $\vdash$ ) {}
and bprv = ( $\vdash$ ) {}
apply unfold_locales
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal by simp
subgoal using MP_null by blast
subgoal by blast
subgoal for A B C
  apply (rule Imp_I)+
  apply (rule MP_same[of _ B])
  apply (rule MP_same[of _ C])

```

```

    apply (auto intro: Neg_D)
  done
subgoal by blast
subgoal by blast
subgoal by blast
subgoal unfolding Fvars_def by (auto intro: MP_null)
subgoal unfolding Fvars_def by (auto intro: MP_null)
subgoal by (auto intro: All_D)
subgoal by (auto intro: Ex_I)
subgoal by simp
subgoal by (metis Conj_E2 Iff_def Imp_I Var_Eq_subst_Iff)
subgoal by blast
subgoal by simp
done

```

interpretation HBL1 where

```

  var = UNIV :: name set
  and trm = UNIV :: tm set
  and fmla = UNIV :: fm set
  and num = {t. ground t}
  and Var = Var
  and FvarsT = Fvars
  and substT =  $\lambda t u x. \text{subst } x u t$ 
  and Fvars = Fvars
  and subst =  $\lambda A u x. \text{subst\_fm } A x u$ 
  and eql = (EQ)
  and cnj = (AND)
  and imp = (IMP)
  and all = All
  and exi = Ex
  and prv = ( $\vdash$ ) {}
  and bprv = ( $\vdash$ ) {}
  and enc = quot
  and P = PfP (Var xx)
  apply unfold_locales
  subgoal by (simp add: quot_fm_coding)
  subgoal by simp
  subgoal unfolding Fvars_def by (auto simp: fresh_at_base(2))
  subgoal by (auto simp: proved_imp_proved_PfP)
done

```

interpretation Goedel_Form where

```

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

```

```

and bprv = ( $\vdash$ ) {}
and enc = quot
and S = KRP (quot (Var xx)) (Var xx) (Var yy)
and P = Pfp (Var xx)
apply unfold_locales
subgoal by simp
subgoal unfolding Fvars_def by (auto simp: fresh_at_base(2))
subgoal
  unfolding Let_def
  by (subst psubst_eq_rawpsubst2)
  (auto simp: quot_fm_coding prove_KRP Fvars_def)
subgoal
  unfolding Let_def
  by (subst (1 2) psubst_eq_rawpsubst2)
  (auto simp: quot_fm_coding KRP_unique[THEN Sym] Fvars_def)
done

```

interpretation g2: *Goedel_Second_Assumptions* **where**

```

  var = UNIV :: name set
and trm = UNIV :: tm set
and fmla = UNIV :: fm set
and num = {t. ground t}
and Var = Var
and FvarsT = Fvars
and substT =  $\lambda t u x. \text{subst } x u t$ 
and Fvars = Fvars
and subst =  $\lambda A u x. \text{subst\_fm } A x u$ 
and eql = (EQ)
and conj = (AND)
and imp = (IMP)
and all = All
and exi = Ex
and fls = Fls
and prv = ( $\vdash$ ) {}
and bprv = ( $\vdash$ ) {}
and enc = quot
and S = KRP (quot (Var xx)) (Var xx) (Var yy)
and P = Pfp (Var xx)
apply unfold_locales
subgoal by (auto simp: PP_def intro: Pfp_implies_ModPon_Pfp_quot)
subgoal by (auto simp: PP_def quot_fm_coding Provability)
done

```

theorem $\neg \{ \} \vdash \text{Fls} \implies \neg \{ \} \vdash \text{neg } (\text{Pfp } (\text{quot } \text{Fls}))$

by (rule g2.goedel_second[unfolded consistent_def PP_def Pfp_subst subst.simps simp_thms if_True])