

# Gödel's Incompleteness Theorems

Lawrence C. Paulson

20. April 2020

## **Zusammenfassung**

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

# Inhaltsverzeichnis

<b>1</b>	<b>Syntax of Terms and Formulas using Nominal Logic</b>	<b>6</b>
1.1	Terms and Formulas . . . . .	6
1.1.1	Hf is a pure permutation type . . . . .	6
1.1.2	The datatypes . . . . .	6
1.1.3	Substitution . . . . .	7
1.1.4	Semantics . . . . .	8
1.1.5	Derived syntax . . . . .	10
1.1.6	Derived logical connectives . . . . .	11
1.2	Axioms and Theorems . . . . .	12
1.2.1	Logical axioms . . . . .	12
1.2.2	Concrete variables . . . . .	13
1.2.3	The HF axioms . . . . .	13
1.2.4	Equality axioms . . . . .	14
1.2.5	The proof system . . . . .	14
1.2.6	Derived rules of inference . . . . .	15
1.2.7	The Deduction Theorem . . . . .	18
1.2.8	Cut rules . . . . .	18
1.3	Miscellaneous logical rules . . . . .	19
1.3.1	Quantifier reasoning . . . . .	23
1.3.2	Congruence rules . . . . .	24
1.4	Equality reasoning . . . . .	25
1.4.1	The congruence property for $(EQ)$ , and other basic properties of equality . . . . .	25
1.4.2	The congruence property for $(IN)$ . . . . .	25
1.4.3	The congruence properties for $Eats$ and $HPair$ . . . . .	25
1.4.4	Substitution for Equalities . . . . .	26
1.4.5	Congruence Rules for Predicates . . . . .	26
1.5	Zero and Falsity . . . . .	27
1.5.1	The Formula $Fls$ ; Consistency of the Calculus . . . . .	27
1.5.2	More properties of $Zero$ . . . . .	28
1.5.3	Basic properties of $Eats$ . . . . .	28
1.6	Bounded Quantification involving $Eats$ . . . . .	30
1.7	Induction . . . . .	31

<b>2</b>	<b>De Bruijn Syntax, Quotations, Codes, V-Codes</b>	<b>32</b>
2.1	de Bruijn Indices (locally-nameless version)	32
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	41
2.6	Quotations are Injective	43
2.6.1	Terms	43
2.6.2	Formulas	43
2.6.3	The set $\Gamma$ of Definition 1.1, constant terms used for coding	44
2.7	V-Coding for terms and formulas, for the Second Theorem	44
<b>3</b>	<b>Basic Predicates</b>	<b>47</b>
3.1	The Subset Relation	47
3.2	Extensionality	49
3.3	The Disjointness Relation	49
3.4	The Foundation Theorem	51
3.5	The Ordinal Property	51
3.6	Induction on Ordinals	54
3.7	Linearity of Ordinals	54
3.8	The predicate <i>OrdNotEqP</i>	55
3.9	Predecessor of an Ordinal	56
3.10	Case Analysis and Zero/SUCC Induction	56
3.11	The predicate <i>HFun-Sigma</i>	57
3.12	The predicate <i>HDomain-Incl</i>	58
3.13	<i>HPair</i> is Provably Injective	59
3.14	<i>SUCC</i> is Provably Injective	59
3.15	The predicate <i>LstSeqP</i>	60
<b>4</b>	<b>Sigma-Formulas and Theorem 2.5</b>	<b>63</b>
4.1	Ground Terms and Formulas	63
4.2	Sigma Formulas	64
4.2.1	Strict Sigma Formulas	64
4.2.2	Closure properties for Sigma-formulas	64
4.3	Lemma 2.2: Atomic formulas are Sigma-formulas	65
4.4	Universal Quantification Bounded by an Arbitrary Term	66
4.5	Lemma 2.3: Sequence-related concepts are Sigma-formulas	66
4.6	A Key Result: Theorem 2.5	67
4.6.1	Sigma-Eats Formulas	67

4.6.2	Preparation . . . . .	68
4.6.3	The base cases: ground atomic formulas . . . . .	68
<b>5</b>	<b>Predicates for Terms, Formulas and Substitution</b>	<b>70</b>
5.1	Predicates for atomic terms . . . . .	70
5.1.1	Free Variables . . . . .	70
5.1.2	De Bruijn Indexes . . . . .	71
5.1.3	Various syntactic lemmas . . . . .	72
5.2	The predicate <i>SeqCTermP</i> , for Terms and Constants . . . . .	72
5.3	The predicates <i>TermP</i> and <i>ConstP</i> . . . . .	73
5.3.1	Definition . . . . .	73
5.3.2	Correctness: It Corresponds to Quotations of Real Terms	74
5.3.3	Correctness properties for constants . . . . .	75
5.4	Abstraction over terms . . . . .	75
5.4.1	Defining the syntax: quantified body . . . . .	75
5.4.2	Defining the syntax: main predicate . . . . .	76
5.4.3	Correctness: It Coincides with Abstraction over real terms . . . . .	77
5.5	Substitution over terms . . . . .	77
5.5.1	Defining the syntax . . . . .	77
5.6	Abstraction over formulas . . . . .	78
5.6.1	The predicate <i>AbstAtomicP</i> . . . . .	78
5.6.2	The predicate <i>AbsMakeForm</i> . . . . .	79
5.6.3	Defining the syntax: the main <i>AbstForm</i> predicate . . . . .	81
5.6.4	Correctness: It Coincides with Abstraction over real Formulas . . . . .	81
5.7	Substitution over formulas . . . . .	82
5.7.1	The predicate <i>SubstAtomicP</i> . . . . .	82
5.7.2	The predicate <i>SubstMakeForm</i> . . . . .	83
5.7.3	Defining the syntax: the main <i>SubstForm</i> predicate . . . . .	84
5.7.4	Correctness of substitution over formulas . . . . .	85
5.8	The predicate <i>AtomicP</i> . . . . .	85
5.9	The predicate <i>MakeForm</i> . . . . .	86
5.10	The predicate <i>SeqFormP</i> . . . . .	87
5.11	The predicate <i>FormP</i> . . . . .	87
5.11.1	Definition . . . . .	87
5.11.2	Correctness: It Corresponds to Quotations of Real Formulas . . . . .	88
5.11.3	The predicate <i>VarNonOccFormP</i> (Derived from <i>SubstFormP</i> ) . . . . .	89
5.11.4	Correctness for Real Terms and Formulas . . . . .	89

<b>6</b>	<b>Formalizing Provability</b>	<b>91</b>
6.1	Section 4 Predicates (Leading up to Pf) . . . . .	91
6.1.1	The predicate <i>SentP</i> , for the Sentential (Boolean) Axioms . . . . .	91
6.1.2	The predicate <i>Equality-axP</i> , for the Equality Axioms . . . . .	92
6.1.3	The predicate <i>HF-axP</i> , for the HF Axioms . . . . .	92
6.1.4	The specialisation axioms . . . . .	92
6.1.5	The induction axioms . . . . .	93
6.1.6	The predicate <i>AxiomP</i> , for any Axioms . . . . .	94
6.1.7	The predicate <i>ModPonP</i> , for the inference rule Modus Ponens . . . . .	95
6.1.8	The predicate <i>ExistsP</i> , for the existential rule . . . . .	96
6.1.9	The predicate <i>SubstP</i> , for the substitution rule . . . . .	97
6.1.10	The predicate <i>PrfP</i> . . . . .	97
6.1.11	The predicate <i>PfP</i> . . . . .	98
6.2	Proposition 4.4 . . . . .	99
6.2.1	Left-to-Right Proof . . . . .	99
6.2.2	Right-to-Left Proof . . . . .	100
<b>7</b>	<b>Uniqueness Results: Syntactic Relations are Functions</b>	<b>102</b>
7.0.1	<i>SeqStTermP</i> . . . . .	102
7.0.2	<i>SubstAtomicP</i> . . . . .	103
7.0.3	<i>SeqSubstFormP</i> . . . . .	103
7.0.4	<i>SubstFormP</i> . . . . .	104
<b>8</b>	<b>Section 6 Material and Gdel's First Incompleteness Theorem</b>	<b>105</b>
8.1	The Function W and Lemma 6.1 . . . . .	105
8.1.1	Predicate form, defined on sequences . . . . .	105
8.1.2	Predicate form of W . . . . .	106
8.1.3	Proving that these relations are functions . . . . .	107
8.1.4	The equivalent function . . . . .	107
8.2	The Function HF and Lemma 6.2 . . . . .	108
8.2.1	Defining the syntax: quantified body . . . . .	108
8.2.2	Defining the syntax: main predicate . . . . .	109
8.2.3	Proving that these relations are functions . . . . .	109
8.2.4	Finally The Function HF Itself . . . . .	110
8.3	The Function K and Lemma 6.3 . . . . .	110
8.4	The Diagonal Lemma and Gdel's Theorem . . . . .	111
<b>9</b>	<b>Syntactic Preliminaries for the Second Incompleteness Theorem</b>	<b>112</b>
9.1	NotInDom . . . . .	112
9.2	Restriction of a Sequence to a Domain . . . . .	113

9.3	Applications to LstSeqP . . . . .	114
9.4	Ordinal Addition . . . . .	115
9.4.1	Predicate form, defined on sequences . . . . .	115
9.4.2	Proving that these relations are functions . . . . .	116
9.5	A Shifted Sequence . . . . .	118
9.6	Union of Two Sets . . . . .	119
9.7	Append on Sequences . . . . .	120
9.8	LstSeqP and SeqAppendP . . . . .	121
9.9	Substitution and Abstraction on Terms . . . . .	121
9.9.1	Atomic cases . . . . .	121
9.9.2	Non-atomic cases . . . . .	122
9.9.3	Substitution over a constant . . . . .	122
9.10	Substitution on Formulas . . . . .	123
9.10.1	Membership . . . . .	123
9.10.2	Equality . . . . .	123
9.10.3	Negation . . . . .	123
9.10.4	Disjunction . . . . .	124
9.10.5	Existential . . . . .	124
9.11	Constant Terms . . . . .	124
9.12	Proofs . . . . .	124
<b>10</b>	<b>Pseudo-Coding: Section 7 Material</b>	<b>126</b>
10.1	General Lemmas . . . . .	126
10.2	Simultaneous Substitution . . . . .	127
10.3	The Main Theorems of Section 7 . . . . .	130
<b>11</b>	<b>Quotations of the Free Variables</b>	<b>132</b>
11.1	Sequence version of the “Special p-Function, F*” . . . . .	132
11.1.1	Defining the syntax: quantified body . . . . .	132
11.1.2	Correctness properties . . . . .	133
11.2	The “special function” itself . . . . .	133
11.2.1	Defining the syntax . . . . .	134
11.2.2	Correctness properties . . . . .	134
11.3	The Operator <i>quote-all</i> . . . . .	135
11.3.1	Definition and basic properties . . . . .	135
11.3.2	Transferring theorems to the level of derivability . . . . .	135
11.4	Star Property. Equality and Membership: Lemmas 9.3 and 9.4 . . . . .	137
11.5	Star Property. Universal Quantifier: Lemma 9.7 . . . . .	137
11.6	The Derivability Condition, Theorem 9.1 . . . . .	138
<b>12</b>	<b>Gdel’s Second Incompleteness Theorem</b>	<b>139</b>

# Kapitel 1

## Syntax of Terms and Formulas using Nominal Logic

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

### 1.1 Terms and Formulas

#### 1.1.1 Hf is a pure permutation type

```
instantiation hf :: pt
begin
  definition p · (s::hf) = s
  instance
    ⟨proof⟩
end

instance hf :: pure
  ⟨proof⟩

atom-decl name

declare fresh-set-empty [simp]

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

#### 1.1.2 The datatypes

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



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

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

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

### 1.1.3 Substitution

**nominal-function** *subst* :: *name*  $\Rightarrow$  *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)*  
 <proof>

**nominal-termination** (*eqvt*)

<proof>

**lemma** *fresh-subst-if [simp]*:

*j*  $\sharp$  *subst i x t*  $\longleftrightarrow$  (*atom i*  $\sharp$  *t*  $\wedge$  *j*  $\sharp$  *t*)  $\vee$  (*j*  $\sharp$  *x*  $\wedge$  (*j*  $\sharp$  *t*  $\vee$  *j* = *atom i*))  
 <proof>

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

<proof>

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

<proof>

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

*atom j*  $\sharp$  *tm*  $\implies$  *subst j u (subst i t tm)* = *subst i (subst j u t) tm*  
 <proof>

**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)*  
 <proof>

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

<proof>

**nominal-function** *subst-fm* :: *fm*  $\Rightarrow$  *name*  $\Rightarrow$  *tm*  $\Rightarrow$  *fm* ( $\text{'(-::=-)'} [1000, 0, 0]$   
 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*:  $(\text{Neg } A)(i ::= x) = \text{Neg } (A(i ::= x))$   
 | *Ex*:  $\text{atom } j \# (i, x) \implies (\text{Ex } j A)(i ::= x) = \text{Ex } j (A(i ::= x))$   
 <proof>

**nominal-termination** (*eqvt*)  
 <proof>

**lemma** *size-subst-fm* [*simp*]:  $\text{size } (A(i ::= x)) = \text{size } A$   
 <proof>

**lemma** *forget-subst-fm* [*simp*]:  $\text{atom } a \# A \implies A(a ::= x) = A$   
 <proof>

**lemma** *subst-fm-id* [*simp*]:  $A(a ::= \text{Var } a) = A$   
 <proof>

**lemma** *fresh-subst-fm-if* [*simp*]:  
 $j \# (A(i ::= x)) \longleftrightarrow (\text{atom } i \# A \wedge j \# A) \vee (j \# x \wedge (j \# A \vee j = \text{atom } i))$   
 <proof>

**lemma** *subst-fm-commute* [*simp*]:  
 $\text{atom } j \# A \implies (A(i ::= t))(j ::= u) = A(i ::= \text{subst } j \ u \ t)$   
 <proof>

**lemma** *repeat-subst-fm* [*simp*]:  $(A(i ::= t))(i ::= u) = A(i ::= \text{subst } i \ u \ t)$   
 <proof>

**lemma** *subst-fm-Ex-with-renaming*:  
 $\text{atom } i' \# (A, i, j, t) \implies (\text{Ex } i A)(j ::= t) = \text{Ex } i' (((i \leftrightarrow i') \cdot A)(j ::= t))$   
 <proof>

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

<ML>

#### 1.1.4 Semantics

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

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

$\text{eval-tm } e \ \text{Zero} = 0$   
 |  $\text{eval-tm } e \ (\text{Var } k) = \text{finfun-apply } e \ k$   
 |  $\text{eval-tm } e \ (\text{Eats } t \ u) = \text{eval-tm } e \ t \ \triangleleft \ \text{eval-tm } e \ u$   
 <proof>

**nominal-termination** (*eqvt*)  
 <proof>

**syntax**

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

**translations**

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

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

$eval\text{-}fm \ e \ (t \ IN \ u) \longleftrightarrow \llbracket t \rrbracket e \in \llbracket u \rrbracket e$   
 $| \ eval\text{-}fm \ e \ (t \ EQ \ u) \longleftrightarrow \llbracket t \rrbracket e = \llbracket u \rrbracket e$   
 $| \ eval\text{-}fm \ e \ (A \ OR \ B) \longleftrightarrow eval\text{-}fm \ e \ A \vee eval\text{-}fm \ e \ B$   
 $| \ eval\text{-}fm \ e \ (Neg \ A) \longleftrightarrow (\sim \ eval\text{-}fm \ e \ A)$   
 $| \ atom \ k \ \#\ e \Longrightarrow eval\text{-}fm \ e \ (Ex \ k \ A) \longleftrightarrow (\exists x. \ eval\text{-}fm \ (finfun\text{-}update \ e \ k \ x) \ A)$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt)$ 

$\langle proof \rangle$

**lemma**  $eval\text{-}tm\text{-}rename$ :

**assumes**  $atom \ k' \ \#\ t$

**shows**  $\llbracket t \rrbracket (finfun\text{-}update \ e \ k \ x) = \llbracket (k' \leftrightarrow k) \cdot t \rrbracket (finfun\text{-}update \ e \ k' \ x)$

$\langle proof \rangle$

**lemma**  $eval\text{-}fm\text{-}rename$ :

**assumes**  $atom \ k' \ \#\ A$

**shows**  $eval\text{-}fm \ (finfun\text{-}update \ e \ k \ x) \ A = eval\text{-}fm \ (finfun\text{-}update \ e \ k' \ x) \ ((k' \leftrightarrow k) \cdot A)$

$\langle proof \rangle$

**lemma**  $better\text{-}ex\text{-}eval\text{-}fm[simp]$ :

$eval\text{-}fm \ e \ (Ex \ k \ A) \longleftrightarrow (\exists x. \ eval\text{-}fm \ (finfun\text{-}update \ e \ k \ x) \ A)$

$\langle proof \rangle$

**lemma**  $forget\text{-}eval\text{-}tm \ [simp]$ :  $atom \ i \ \#\ t \Longrightarrow \llbracket t \rrbracket (finfun\text{-}update \ e \ i \ x) = \llbracket t \rrbracket e$ 

$\langle proof \rangle$

**lemma**  $forget\text{-}eval\text{-}fm \ [simp]$ :

$atom \ k \ \#\ A \Longrightarrow eval\text{-}fm \ (finfun\text{-}update \ e \ k \ x) \ A = eval\text{-}fm \ e \ A$

$\langle proof \rangle$

**lemma**  $eval\text{-}subst\text{-}tm$ :  $\llbracket subst \ i \ t \ u \rrbracket e = \llbracket u \rrbracket (finfun\text{-}update \ e \ i \ \llbracket t \rrbracket e)$ 

$\langle proof \rangle$

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

$\langle proof \rangle$

### 1.1.5 Derived syntax

#### Ordered pairs

**definition**  $HPair :: tm \Rightarrow tm \Rightarrow tm$

**where**  $HPair\ a\ b = Eats\ (Eats\ Zero\ (Eats\ (Eats\ Zero\ b)\ a))\ (Eats\ (Eats\ Zero\ a)\ a)$

**lemma**  $HPair\text{-}eqvt$  [*eqvt*]:  $(p \cdot HPair\ a\ b) = HPair\ (p \cdot a)\ (p \cdot b)$   
*<proof>*

**lemma**  $fresh\text{-}HPair$  [*simp*]:  $x \# HPair\ a\ b \longleftrightarrow (x \# a \wedge x \# b)$   
*<proof>*

**lemma**  $HPair\text{-}injective\text{-}iff$  [*iff*]:  $HPair\ a\ b = HPair\ a'\ b' \longleftrightarrow (a = a' \wedge b = b')$   
*<proof>*

**lemma**  $subst\text{-}tm\text{-}HPair$  [*simp*]:  $subst\ i\ x\ (HPair\ a\ b) = HPair\ (subst\ i\ x\ a)\ (subst\ i\ x\ b)$   
*<proof>*

**lemma**  $eval\text{-}tm\text{-}HPair$  [*simp*]:  $\llbracket HPair\ a\ b \rrbracket e = hpair\ \llbracket a \rrbracket e\ \llbracket b \rrbracket e$   
*<proof>*

#### Ordinals

**definition**

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

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

**where**

$ORD\text{-}OF\ 0 = Zero$   
 $| ORD\text{-}OF\ (Suc\ k) = SUCC\ (ORD\text{-}OF\ k)$

**lemma**  $eval\text{-}tm\text{-}SUCC$  [*simp*]:  $\llbracket SUCC\ t \rrbracket e = succ\ \llbracket t \rrbracket e$   
*<proof>*

**lemma**  $SUCC\text{-}fresh\text{-}iff$  [*simp*]:  $a \# SUCC\ t \longleftrightarrow a \# t$   
*<proof>*

**lemma**  $SUCC\text{-}eqvt$  [*eqvt*]:  $(p \cdot SUCC\ a) = SUCC\ (p \cdot a)$   
*<proof>*

**lemma**  $SUCC\text{-}subst$  [*simp*]:  $subst\ i\ t\ (SUCC\ k) = SUCC\ (subst\ i\ t\ k)$   
*<proof>*

**lemma**  $eval\text{-}tm\text{-}ORD\text{-}OF$  [*simp*]:  $\llbracket ORD\text{-}OF\ n \rrbracket e = ord\text{-}of\ n$   
*<proof>*

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

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

### 1.1.6 Derived logical connectives

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

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

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

#### Conjunction

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

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

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

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

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

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

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

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

#### If and only if

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

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

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

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

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

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

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

## 1.2 Axioms and Theorems

### 1.2.1 Logical axioms

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

**where**

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

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

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

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

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

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

*ind*:

- |  $\text{atom } (j :: \text{name}) \# (i, A)$
- |  $\implies A(i ::= \text{Zero}) \text{ IMP } ((\text{All } i \text{ (All } j \text{ (A IMP (A(i ::= Var } j) \text{ IMP } A(i ::= \text{Eats}(\text{Var } i)(\text{Var } j))))))$
- |  $\text{IMP } (\text{All } i \text{ A})$
- |  $\in \text{induction-axioms}$

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

$atom\ j\ \#(i, A)$   
 $\implies eval\text{-}fm\ (finfun\text{-}update\ (finfun\text{-}update\ (finfun\text{-}update\ e\ i\ x)\ j\ y)\ i\ z)\ A =$   
 $eval\text{-}fm\ (finfun\text{-}update\ e\ i\ z)\ A$   
{*proof*}

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

{*proof*}

## 1.2.2 Concrete variables

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

**abbreviation**

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

**abbreviation**

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

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

**abbreviation**

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

**abbreviation**

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

**abbreviation**

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

## 1.2.3 The HF axioms

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

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

**lemma** *HF1-holds*:  $eval\text{-}fm\ e\ HF1$

{*proof*}

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

$HF2 \equiv Var\ X0\ EQ\ Eats\ (Var\ X1)\ (Var\ X2)\ IFF$

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

**lemma** *HF2-holds*:  $eval\text{-}fm\ e\ HF2$

{*proof*}

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

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

*<proof>*

### 1.2.4 Equality axioms

**definition** *refl-ax* :: *fm* **where**

*refl-ax* = *Var X1 EQ Var X1*

**lemma** *refl-ax-holds*: *eval-fm e refl-ax*

*<proof>*

**definition** *eq-cong-ax* :: *fm* **where**

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

**lemma** *eq-cong-ax-holds*: *eval-fm e eq-cong-ax*

*<proof>*

**definition** *mem-cong-ax* :: *fm* **where**

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

**lemma** *mem-cong-ax-holds*: *eval-fm e mem-cong-ax*

*<proof>*

**definition** *eats-cong-ax* :: *fm* **where**

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

**lemma** *eats-cong-ax-holds*: *eval-fm e eats-cong-ax*

*<proof>*

**definition** *equality-axioms* :: *fm set* **where**

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

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

*<proof>*

### 1.2.5 The proof system

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

**consts** *extra-axiom* :: *fm*

**specification** (*extra-axiom*)

*extra-axiom-holds*: *eval-fm e extra-axiom*

*<proof>*



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

**where**

*Hyp*:  $A \in H \Longrightarrow H \vdash A$   
| *Extra*:  $H \vdash \text{extra-axiom}$   
| *Bool*:  $A \in \text{boolean-axioms} \Longrightarrow H \vdash A$   
| *Eq*:  $A \in \text{equality-axioms} \Longrightarrow H \vdash A$   
| *Spec*:  $A \in \text{special-axioms} \Longrightarrow H \vdash A$   
| *HF*:  $A \in \text{HF-axioms} \Longrightarrow H \vdash A$   
| *Ind*:  $A \in \text{induction-axioms} \Longrightarrow H \vdash A$   
| *MP*:  $H \vdash A \text{ IMP } B \Longrightarrow H' \vdash A \Longrightarrow H \cup H' \vdash B$   
| *Exists*:  $H \vdash A \text{ IMP } B \Longrightarrow \text{atom } i \# B \Longrightarrow \forall C \in H. \text{atom } i \# C \Longrightarrow H \vdash (\text{Ex } i A) \text{ IMP } B$

Soundness theorem!

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

## 1.2.6 Derived rules of inference

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

**lemma** *thin-Un*:  $H \vdash A \Longrightarrow H \cup H' \vdash A$   
{*proof*}

**lemma** *thin*:  $H \vdash A \Longrightarrow H \subseteq H' \Longrightarrow H' \vdash A$   
{*proof*}

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

**lemma** *thin1*:  $H \vdash B \Longrightarrow \text{insert } A H \vdash B$   
{*proof*}

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

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

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

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

**lemma rotate3:**  $insert\ A3\ (insert\ A1\ (insert\ A2\ H)) \vdash B \implies insert\ A1\ (insert\ A2\ (insert\ A3\ H)) \vdash B$   
 ⟨proof⟩

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

**lemma rotate5:**  
 $insert\ A5\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ H)))) \vdash B$   
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ H)))) \vdash B$   
 ⟨proof⟩

**lemma rotate6:**  
 $insert\ A6\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ H)))))) \vdash B$   
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ H)))))) \vdash B$   
 ⟨proof⟩

**lemma rotate7:**  
 $insert\ A7\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ H)))))) \vdash B$   
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ H)))))) \vdash B$   
 ⟨proof⟩

**lemma rotate8:**  
 $insert\ A8\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ H)))))) \vdash B$   
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ H)))))) \vdash B$   
 ⟨proof⟩

**lemma rotate9:**  
 $insert\ A9\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ H)))))) \vdash B$   
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ H)))))) \vdash B$   
 ⟨proof⟩

**lemma rotate10:**  
 $insert\ A10\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ H)))))) \vdash B$   
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ H)))))) \vdash B$   
 ⟨proof⟩

**lemma rotate11:**

$insert\ A11\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ H)))))))))) \vdash B$   
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ H)))))))))) \vdash B$   
*<proof>*

**lemma rotate12:**

$insert\ A12\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ H)))))))))) \vdash B$   
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ H)))))))))) \vdash B$   
*<proof>*

**lemma rotate13:**

$insert\ A13\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ H)))))))))) \vdash B$   
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ (insert\ A13\ H)))))))))) \vdash B$   
*<proof>*

**lemma rotate14:**

$insert\ A14\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ (insert\ A13\ H)))))))))) \vdash B$   
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ (insert\ A13\ (insert\ A14\ H)))))))))) \vdash B$   
*<proof>*

**lemma rotate15:**

$insert\ A15\ (insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ (insert\ A13\ (insert\ A14\ H)))))))))) \vdash B$   
 $\implies insert\ A1\ (insert\ A2\ (insert\ A3\ (insert\ A4\ (insert\ A5\ (insert\ A6\ (insert\ A7\ (insert\ A8\ (insert\ A9\ (insert\ A10\ (insert\ A11\ (insert\ A12\ (insert\ A13\ (insert\ A14\ (insert\ A15\ H)))))))))) \vdash B$   
*<proof>*

**lemma MP-same:**  $H \vdash A \text{ IMP } B \implies H \vdash A \implies H \vdash B$   
*<proof>*

**lemma MP-thin:**  $HA \vdash A \text{ IMP } B \implies HB \vdash A \implies HA \cup HB \subseteq H \implies H \vdash B$   
*<proof>*

**lemma MP-null:**  $\{\} \vdash A \text{ IMP } B \implies H \vdash A \implies H \vdash B$   
*<proof>*

**lemma** *Disj-commute*:  $H \vdash B \text{ OR } A \implies H \vdash A \text{ OR } B$   
*<proof>*

**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>*

**lemma** *Assume*: *insert*  $A$   $H \vdash A$   
*<proof>*

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

**declare** *AssumeH* [*intro!*]

**lemma** *Imp-triv-I*:  $H \vdash B \implies H \vdash A \text{ IMP } B$   
*<proof>*

**lemma** *DisjAssoc1*:  $H \vdash A \text{ OR } (B \text{ OR } C) \implies H \vdash (A \text{ OR } B) \text{ OR } C$   
*<proof>*

**lemma** *DisjAssoc2*:  $H \vdash (A \text{ OR } B) \text{ OR } C \implies H \vdash A \text{ OR } (B \text{ OR } C)$   
*<proof>*

**lemma** *Disj-commute-Imp*:  $H \vdash (B \text{ OR } A) \text{ IMP } (A \text{ OR } B)$   
*<proof>*

**lemma** *Disj-Semicong-1*:  $H \vdash A \text{ OR } C \implies H \vdash A \text{ IMP } B \implies H \vdash B \text{ OR } C$   
*<proof>*

**lemma** *Imp-Imp-commute*:  $H \vdash B \text{ IMP } (A \text{ IMP } C) \implies H \vdash A \text{ IMP } (B \text{ IMP } C)$   
*<proof>*

### 1.2.7 The Deduction Theorem

**lemma** *deduction-Diff*: **assumes**  $H \vdash B$  **shows**  $H - \{C\} \vdash C \text{ IMP } B$   
*<proof>*

**theorem** *Imp-I* [*intro!*]: *insert*  $A$   $H \vdash B \implies H \vdash A \text{ IMP } B$   
*<proof>*

**lemma** *anti-deduction*:  $H \vdash A \text{ IMP } B \implies \text{insert } A \text{ } H \vdash B$   
*<proof>*

### 1.2.8 Cut rules

**lemma** *cut*:  $H \vdash A \implies \text{insert } A \text{ } H' \vdash B \implies H \cup H' \vdash B$

*<proof>*

**lemma** *cut-same*:  $H \vdash A \implies \text{insert } A \ H \vdash B \implies H \vdash B$   
*<proof>*

**lemma** *cut-thin*:  $HA \vdash A \implies \text{insert } A \ HB \vdash B \implies HA \cup HB \subseteq H \implies H \vdash B$   
*<proof>*

**lemma** *cut0*:  $\{\} \vdash A \implies \text{insert } A \ H \vdash B \implies H \vdash B$   
*<proof>*

**lemma** *cut1*:  $\{A\} \vdash B \implies H \vdash A \implies H \vdash B$   
*<proof>*

**lemma** *rcut1*:  $\{A\} \vdash B \implies \text{insert } B \ H \vdash C \implies \text{insert } A \ H \vdash C$   
*<proof>*

**lemma** *cut2*:  $[\{A,B\} \vdash C; H \vdash A; H \vdash B] \implies H \vdash C$   
*<proof>*

**lemma** *rcut2*:  $\{A,B\} \vdash C \implies \text{insert } C \ H \vdash D \implies H \vdash B \implies \text{insert } A \ H \vdash D$   
*<proof>*

**lemma** *cut3*:  $[\{A,B,C\} \vdash D; H \vdash A; H \vdash B; H \vdash C] \implies H \vdash D$   
*<proof>*

**lemma** *cut4*:  $[\{A,B,C,D\} \vdash E; H \vdash A; H \vdash B; H \vdash C; H \vdash D] \implies H \vdash E$   
*<proof>*

### 1.3 Miscellaneous logical rules

**lemma** *Disj-I1*:  $H \vdash A \implies H \vdash A \text{ OR } B$   
*<proof>*

**lemma** *Disj-I2*:  $H \vdash B \implies H \vdash A \text{ OR } B$   
*<proof>*

**lemma** *Peirce*:  $H \vdash (\text{Neg } A) \text{ IMP } A \implies H \vdash A$   
*<proof>*

**lemma** *Contra*:  $\text{insert } (\text{Neg } A) \ H \vdash A \implies H \vdash A$   
*<proof>*

**lemma** *Imp-Neg-I*:  $H \vdash A \text{ IMP } B \implies H \vdash A \text{ IMP } (\text{Neg } B) \implies H \vdash \text{Neg } A$   
*<proof>*

**lemma** *NegNeg-I*:  $H \vdash A \implies H \vdash \text{Neg } (\text{Neg } A)$   
*<proof>*

**lemma** *NegNeg-D*:  $H \vdash \text{Neg} (\text{Neg} A) \Longrightarrow H \vdash A$   
*<proof>*

**lemma** *Neg-D*:  $H \vdash \text{Neg} A \Longrightarrow H \vdash A \Longrightarrow H \vdash B$   
*<proof>*

**lemma** *Disj-Neg-1*:  $H \vdash A \text{ OR } B \Longrightarrow H \vdash \text{Neg} B \Longrightarrow H \vdash A$   
*<proof>*

**lemma** *Disj-Neg-2*:  $H \vdash A \text{ OR } B \Longrightarrow H \vdash \text{Neg} A \Longrightarrow H \vdash B$   
*<proof>*

**lemma** *Neg-Disj-I*:  $H \vdash \text{Neg} A \Longrightarrow H \vdash \text{Neg} B \Longrightarrow H \vdash \text{Neg} (A \text{ OR } B)$   
*<proof>*

**lemma** *Conj-I [intro!]*:  $H \vdash A \Longrightarrow H \vdash B \Longrightarrow H \vdash A \text{ AND } B$   
*<proof>*

**lemma** *Conj-E1*:  $H \vdash A \text{ AND } B \Longrightarrow H \vdash A$   
*<proof>*

**lemma** *Conj-E2*:  $H \vdash A \text{ AND } B \Longrightarrow H \vdash B$   
*<proof>*

**lemma** *Conj-commute*:  $H \vdash B \text{ AND } A \Longrightarrow H \vdash A \text{ AND } B$   
*<proof>*

**lemma** *Conj-E*: **assumes** *insert A (insert B H) ⊢ C* **shows** *insert (A AND B) H ⊢ C*  
*<proof>*

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

**lemma** *Neg-I0*: **assumes**  $(\bigwedge B. \text{atom } i \# B \Longrightarrow \text{insert } A \text{ H } \vdash B)$  **shows**  $H \vdash \text{Neg } A$   
*<proof>*

**lemma** *Neg-mono*:  $\text{insert } A \text{ H } \vdash B \Longrightarrow \text{insert } (\text{Neg } B) \text{ H } \vdash \text{Neg } A$   
*<proof>*

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

**lemma** *Disj-mono*:

**assumes** *insert A H ⊢ B insert C H ⊢ D* **shows** *insert (A OR C) H ⊢ B OR D*  
 ⟨*proof*⟩

**lemma** *Disj-E*:

**assumes** *A: insert A H ⊢ C and B: insert B H ⊢ C* **shows** *insert (A OR B) H ⊢ C*  
 ⟨*proof*⟩

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

**lemma** *Contra'*: *insert A H ⊢ Neg A ⇒ H ⊢ Neg A*  
 ⟨*proof*⟩

**lemma** *NegNeg-E [intro!]*: *insert A H ⊢ B ⇒ insert (Neg (Neg A)) H ⊢ B*  
 ⟨*proof*⟩

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

**lemma** *Imp-E*:

**assumes** *A: H ⊢ A and B: insert B H ⊢ C* **shows** *insert (A IMP B) H ⊢ C*  
 ⟨*proof*⟩

**lemma** *Imp-cut*:

**assumes** *insert C H ⊢ A IMP B {A} ⊢ C*  
**shows** *H ⊢ A IMP B*  
 ⟨*proof*⟩

**lemma** *Iff-I [intro!]*: *insert A H ⊢ B ⇒ insert B H ⊢ A ⇒ H ⊢ A IFF B*  
 ⟨*proof*⟩

**lemma** *Iff-MP-same*: *H ⊢ A IFF B ⇒ H ⊢ A ⇒ H ⊢ B*  
 ⟨*proof*⟩

**lemma** *Iff-MP2-same*: *H ⊢ A IFF B ⇒ H ⊢ B ⇒ H ⊢ A*  
 ⟨*proof*⟩

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

**lemma** *Iff-sym*:  $H \vdash A \text{ IFF } B \implies H \vdash B \text{ IFF } A$   
*<proof>*

**lemma** *Iff-trans*:  $H \vdash A \text{ IFF } B \implies H \vdash B \text{ IFF } C \implies H \vdash A \text{ IFF } C$   
*<proof>*

**lemma** *Iff-E*:  
 $\text{insert } A (\text{insert } B H) \vdash C \implies \text{insert } (\text{Neg } A) (\text{insert } (\text{Neg } B) H) \vdash C \implies \text{insert } (A \text{ IFF } B) H \vdash C$   
*<proof>*

**lemma** *Iff-E1*:  
assumes  $A: H \vdash A$  and  $B: \text{insert } B H \vdash C$  shows  $\text{insert } (A \text{ IFF } B) H \vdash C$   
*<proof>*

**lemma** *Iff-E2*:  
assumes  $A: H \vdash A$  and  $B: \text{insert } B H \vdash C$  shows  $\text{insert } (B \text{ IFF } A) H \vdash C$   
*<proof>*

**lemma** *Iff-MP-left*:  $H \vdash A \text{ IFF } B \implies \text{insert } A H \vdash C \implies \text{insert } B H \vdash C$   
*<proof>*

**lemma** *Iff-MP-left'*:  $H \vdash A \text{ IFF } B \implies \text{insert } B H \vdash C \implies \text{insert } A H \vdash C$   
*<proof>*

**lemma** *Swap*:  $\text{insert } (\text{Neg } B) H \vdash A \implies \text{insert } (\text{Neg } A) H \vdash B$   
*<proof>*

**lemma** *Cases*:  $\text{insert } A H \vdash B \implies \text{insert } (\text{Neg } A) H \vdash B \implies H \vdash B$   
*<proof>*

**lemma** *Neg-Conj-E*:  $H \vdash B \implies \text{insert } (\text{Neg } A) H \vdash C \implies \text{insert } (\text{Neg } (A \text{ AND } B)) H \vdash C$   
*<proof>*

**lemma** *Disj-CI*:  $\text{insert } (\text{Neg } B) H \vdash A \implies H \vdash A \text{ OR } B$   
*<proof>*

**lemma** *Disj-3I*:  $\text{insert } (\text{Neg } A) (\text{insert } (\text{Neg } C) H) \vdash B \implies H \vdash A \text{ OR } B \text{ OR } C$   
*<proof>*

**lemma** *Contrapos1*:  $H \vdash A \text{ IMP } B \implies H \vdash \text{Neg } B \text{ IMP } \text{Neg } A$   
*<proof>*

**lemma** *Contrapos2*:  $H \vdash (\text{Neg } B) \text{ IMP } (\text{Neg } A) \implies H \vdash A \text{ IMP } B$   
*<proof>*

**lemma** *ContraAssumeN [intro]*:  $B \in H \implies \text{insert } (\text{Neg } B) H \vdash A$



*<proof>*

**lemma** *ContraAssume*:  $Neg B \in H \implies insert B H \vdash A$   
*<proof>*

**lemma** *ContraProve*:  $H \vdash B \implies insert (Neg B) H \vdash A$   
*<proof>*

**lemma** *Disj-IE1*:  $insert B H \vdash C \implies insert (A OR B) H \vdash A OR C$   
*<proof>*

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

### 1.3.1 Quantifier reasoning

**lemma** *Ex-I*:  $H \vdash A(i::=x) \implies H \vdash Ex i A$   
*<proof>*

**lemma** *Ex-E*:  
**assumes**  $insert A H \vdash B \text{ atom } i \# B \vee C \in H. \text{ atom } i \# C$   
**shows**  $insert (Ex i A) H \vdash B$   
*<proof>*

**lemma** *Ex-E-with-renaming*:  
**assumes**  $insert ((i \leftrightarrow i') \cdot A) H \vdash B \text{ atom } i' \# (A, i, B) \vee C \in H. \text{ atom } i' \# C$   
**shows**  $insert (Ex i A) H \vdash B$   
*<proof>*

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

**lemma** *Ex-mono*:  $insert A H \vdash B \implies \forall C \in H. \text{ atom } i \# C \implies insert (Ex i A) H \vdash (Ex i B)$   
*<proof>*

**lemma** *All-I* [*intro!*]:  $H \vdash A \implies \forall C \in H. \text{ atom } i \# C \implies H \vdash All i A$   
*<proof>*

**lemma** *All-D*:  $H \vdash All i A \implies H \vdash A(i::=x)$   
*<proof>*

**lemma** *All-E*:  $insert (A(i::=x)) H \vdash B \implies insert (All i A) H \vdash B$

*<proof>*

**lemma** *All-E'*:  $H \vdash \text{All } i \ A \implies \text{insert } (A(i::=x)) \ H \vdash B \implies H \vdash B$   
*<proof>*

**lemma** *All2-E*:  $\llbracket \text{atom } i \ \sharp \ t; H \vdash x \ \text{IN } t; \text{insert } (A(i::=x)) \ H \vdash B \rrbracket \implies \text{insert}$   
 $(\text{All2 } i \ t \ A) \ H \vdash B$   
*<proof>*

**lemma** *All2-E'*:  $\llbracket H \vdash \text{All2 } i \ t \ A; H \vdash x \ \text{IN } t; \text{insert } (A(i::=x)) \ H \vdash B; \text{atom } i \ \sharp$   
 $t \rrbracket \implies H \vdash B$   
*<proof>*

### 1.3.2 Congruence rules

**lemma** *Neg-cong*:  $H \vdash A \ \text{IFF } A' \implies H \vdash \text{Neg } A \ \text{IFF } \text{Neg } A'$   
*<proof>*

**lemma** *Disj-cong*:  $H \vdash A \ \text{IFF } A' \implies H \vdash B \ \text{IFF } B' \implies H \vdash A \ \text{OR } B \ \text{IFF } A'$   
 $\text{OR } B'$   
*<proof>*

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

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

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

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

**lemma** *All-cong*:  $H \vdash A \ \text{IFF } A' \implies \forall C \in H. \text{atom } i \ \sharp \ C \implies H \vdash (\text{All } i \ A) \ \text{IFF}$   
 $(\text{All } i \ A')$   
*<proof>*

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

## 1.4 Equality reasoning

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

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

**lemma** *Refl [iff]*:  $H \vdash t EQ t$   
*<proof>*

Apparently necessary in order to prove the congruence property.

**lemma** *Sym*: **assumes**  $H \vdash t EQ u$  **shows**  $H \vdash u EQ t$   
*<proof>*

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

**lemma** *Trans*: **assumes**  $H \vdash x EQ y$   $H \vdash y EQ z$  **shows**  $H \vdash x EQ z$   
*<proof>*

**lemma** *Eq-cong*:  
**assumes**  $H \vdash t EQ t'$   $H \vdash u EQ u'$  **shows**  $H \vdash t EQ u \text{ IFF } t' EQ u'$   
*<proof>*

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

### 1.4.2 The congruence property for ( $IN$ )

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

**lemma** *Mem-cong*:  
**assumes**  $H \vdash t EQ t'$   $H \vdash u EQ u'$  **shows**  $H \vdash t IN u \text{ IFF } t' IN u'$   
*<proof>*

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

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

**lemma** *Eats-cong*:  $\llbracket H \vdash t EQ t'; H \vdash u EQ u' \rrbracket \implies H \vdash Eats t u EQ Eats t' u'$   
*<proof>*

**lemma** *HPair-cong*:  $\llbracket H \vdash t EQ t'; H \vdash u EQ u' \rrbracket \implies H \vdash HPair t u EQ HPair t' u'$   
*<proof>*

**lemma** *SUCC-cong*:  $H \vdash t \text{ EQ } t' \implies H \vdash \text{SUCC } t \text{ EQ } \text{SUCC } t'$   
 ⟨proof⟩

#### 1.4.4 Substitution for Equalities

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

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

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

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

#### 1.4.5 Congruence Rules for Predicates

**lemma** *P1-cong*:

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

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

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

⟨proof⟩

**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⟩

**lemma** *P3-cong*:

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

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

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

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

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

⟨proof⟩

**lemma** *P4-cong*:

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

**assumes**  $\text{sub}: \bigwedge i \ t1 \ t2 \ t3 \ t4 \ x. \ \text{atom } i \ \# \ tms \implies$

$(P \ t1 \ t2 \ t3 \ t4)(i::=x) = P \ (\text{subst } i \ x \ t1) \ (\text{subst } i \ x \ t2) \ (\text{subst } i \ x \ t3)$

$(\text{subst } i \ x \ t4)$

**and**  $\text{eq}: H \vdash x1 \ \text{EQ} \ x1' \ H \vdash x2 \ \text{EQ} \ x2' \ H \vdash x3 \ \text{EQ} \ x3' \ H \vdash x4 \ \text{EQ} \ x4'$

**shows**  $H \vdash P \ x1 \ x2 \ x3 \ x4 \ \text{IFF} \ P \ x1' \ x2' \ x3' \ x4'$

⟨proof⟩

## 1.5 Zero and Falsity

**lemma** *Mem-Zero-iff*:

**assumes** *atom i # t shows*  $H \vdash (t \text{ EQ Zero}) \text{ IFF } (\text{All } i \text{ (Neg ((Var } i) \text{ IN } t)))$   
*<proof>*

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

*<proof>*

**declare** *Mem-Zero-E* [*THEN rotate2, intro!*]

**declare** *Mem-Zero-E* [*THEN rotate3, intro!*]

**declare** *Mem-Zero-E* [*THEN rotate4, intro!*]

**declare** *Mem-Zero-E* [*THEN rotate5, intro!*]

**declare** *Mem-Zero-E* [*THEN rotate6, intro!*]

**declare** *Mem-Zero-E* [*THEN rotate7, intro!*]

**declare** *Mem-Zero-E* [*THEN rotate8, intro!*]

### 1.5.1 The Formula *Fls*; Consistency of the Calculus

**definition** *Fls* **where**  $Fls \equiv \text{Zero IN Zero}$

**lemma** *Fls-eqt* [*eqt*]:  $(p \cdot Fls) = Fls$

*<proof>*

**lemma** *Fls-fresh* [*simp*]:  $a \# Fls$

*<proof>*

**lemma** *Neg-I* [*intro!*]: *insert*  $A \ H \vdash Fls \implies H \vdash \text{Neg } A$

*<proof>*

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

*<proof>*

**declare** *Neg-E* [*THEN rotate2, intro!*]

**declare** *Neg-E* [*THEN rotate3, intro!*]

**declare** *Neg-E* [*THEN rotate4, intro!*]

**declare** *Neg-E* [*THEN rotate5, intro!*]

**declare** *Neg-E* [*THEN rotate6, intro!*]

**declare** *Neg-E* [*THEN rotate7, intro!*]

**declare** *Neg-E* [*THEN rotate8, intro!*]

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

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

*<proof>*

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

*<proof>*

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

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

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

**lemma** *truth-provable*: *H*  $\vdash$  (*Neg Fls*)  
 ⟨*proof*⟩

**lemma** *ExFalso*: *H*  $\vdash$  *Fls*  $\implies$  *H*  $\vdash$  *A*  
 ⟨*proof*⟩

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

**proposition** *consistent*:  $\neg \{\}$   $\vdash$  *Fls*  
 ⟨*proof*⟩

### 1.5.2 More properties of *Zero*

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

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

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

### 1.5.3 Basic properties of *Eats*

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

**shows**  $H \vdash (z \text{ EQ Eats } t \ u) \text{ IFF } (\text{All } i \ (\text{Var } i \ \text{IN } z \ \text{IFF } \text{Var } i \ \text{IN } t \ \text{OR } \text{Var } i \ \text{EQ } u))$   
 $\langle \text{proof} \rangle$

**lemma** *Eq-Eats-I*:

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

**lemma** *Mem-Eats-Iff*:

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

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

**lemma** *Mem-Eats-E*:

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

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

$\langle \text{proof} \rangle$

**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*:  $H \vdash u \ \text{IN } t \implies H \vdash u \ \text{IN } \text{SUCC } t$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

**lemma** *Mem-SUCC-Refl* [simp]:  $H \vdash k \ \text{IN } \text{SUCC } k$

$\langle \text{proof} \rangle$

**lemma** *Mem-SUCC-E*:

**assumes**  $\text{insert } (u \ \text{IN } t) \ H \vdash C$   $\text{insert } (u \ \text{EQ } t) \ H \vdash C$  **shows**  $\text{insert } (u \ \text{IN } \text{SUCC } t) \ H \vdash C$

$\langle \text{proof} \rangle$

**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*:  $\text{insert } (Eats\ t\ u\ EQ\ Zero)\ H \vdash A$   
 ⟨*proof*⟩

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

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

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

## 1.6 Bounded Quantification involving *Eats*

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

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

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

**lemma** *All2-Eats-I*:  
 $\llbracket atom\ i\ \# (t, u); H \vdash All2\ i\ t\ A; H \vdash A(i ::= u) \rrbracket \implies H \vdash (All2\ i\ (Eats\ t\ u)\ A)$   
 ⟨*proof*⟩

**lemma** *All2-Eats-E1*:  
 $\llbracket atom\ i\ \# (t, u); \forall C \in H. atom\ i\ \# C \rrbracket \implies \text{insert } (All2\ i\ (Eats\ t\ u)\ A)\ H \vdash All2$   
 $i\ t\ A$   
 ⟨*proof*⟩

**lemma** *All2-Eats-E2*:  
 $\llbracket atom\ i\ \# (t, u); \forall C \in H. atom\ i\ \# C \rrbracket \implies \text{insert } (All2\ i\ (Eats\ t\ u)\ A)\ H \vdash$   
 $A(i ::= u)$   
 ⟨*proof*⟩

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



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

**lemma** *All2-SUCC-I*:

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

**lemma** *All2-SUCC-E*:

**assumes**  $\text{atom } i \# t$   
**and**  $\text{insert } (All2\ i\ t\ A)\ (\text{insert } (A(i::=t))\ H) \vdash B$   
**shows**  $\text{insert } (All2\ i\ (SUCC\ t)\ A)\ H \vdash B$   
 $\langle proof \rangle$

**lemma** *All2-SUCC-E'*:

**assumes**  $H \vdash u\ EQ\ SUCC\ t$   
**and**  $\text{atom } i \# t \forall C \in H. \text{atom } i \# C$   
**and**  $\text{insert } (All2\ i\ t\ A)\ (\text{insert } (A(i::=t))\ H) \vdash B$   
**shows**  $\text{insert } (All2\ i\ u\ A)\ H \vdash B$   
 $\langle proof \rangle$

## 1.7 Induction

**lemma** *Ind*:

**assumes**  $j: \text{atom } (j::name) \# (i, A)$   
**and**  $\text{prems}: H \vdash A(i::=Zero)\ H \vdash All\ i\ (All\ j\ (A\ IMP\ (A(i::= Var\ j)\ IMP\ A(i::= Eats\ (Var\ i)\ (Var\ j))))))$   
**shows**  $H \vdash A$   
 $\langle proof \rangle$

**end**

## Kapitel 2

# De Bruijn Syntax, Quotations, Codes, V-Codes

```
theory Coding
imports SyntaxN
begin
```

```
declare fresh-Nil [iff]
```

### 2.1 de Bruijn Indices (locally-nameless version)

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

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

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

```
fun lookup :: name list  $\Rightarrow$  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  $\implies$  name  $\notin$  set e
  <proof>
```

```
lemma lookup-notin: x  $\notin$  set e  $\implies$  lookup e n x = DBVar x
  <proof>
```

**lemma** *lookup-in*:

$x \in \text{set } e \implies \exists k. \text{lookup } e \ n \ x = \text{DBInd } k \wedge n \leq k \wedge k < n + \text{length } e$   
 $\langle \text{proof} \rangle$

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

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

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

**nominal-function** *trans-tm* :: *name list*  $\Rightarrow$  *tm*  $\Rightarrow$  *dbtm*

**where**

$\text{trans-tm } e \ \text{Zero} = \text{DBZero}$   
 $\text{trans-tm } e \ (\text{Var } k) = \text{lookup } e \ 0 \ k$   
 $\text{trans-tm } e \ (\text{Eats } t \ u) = \text{DBEats } (\text{trans-tm } e \ t) \ (\text{trans-tm } e \ u)$   
 $\langle \text{proof} \rangle$

**nominal-termination** (*eqvt*)  
 $\langle \text{proof} \rangle$

**lemma** *fresh-trans-tm-iff* [*simp*]:  $i \# \text{trans-tm } e \ t \longleftrightarrow i \# t \vee i \in \text{atom } \text{'set } e$   
 $\langle \text{proof} \rangle$

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

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

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

**where**

$\text{trans-fm } e \ (\text{Mem } t \ u) = \text{DBMem } (\text{trans-tm } e \ t) \ (\text{trans-tm } e \ u)$   
 $\text{trans-fm } e \ (\text{Eq } t \ u) = \text{DBEq } (\text{trans-tm } e \ t) \ (\text{trans-tm } e \ u)$   
 $\text{trans-fm } e \ (\text{Disj } A \ B) = \text{DBDisj } (\text{trans-fm } e \ A) \ (\text{trans-fm } e \ B)$   
 $\text{trans-fm } e \ (\text{Neg } A) = \text{DBNeg } (\text{trans-fm } e \ A)$   
 $\text{atom } k \ \# \ e \implies \text{trans-fm } e \ (\text{Ex } k \ A) = \text{DBEx } (\text{trans-fm } (k\#e) \ A)$   
 $\langle \text{proof} \rangle$

**nominal-termination** (*eqvt*)  
 $\langle \text{proof} \rangle$

**lemma** *fresh-trans-fm* [*simp*]:  $i \# \text{trans-fm } e \ A \longleftrightarrow i \# A \vee i \in \text{atom } \text{'set } e$   
 $\langle \text{proof} \rangle$

**abbreviation** *DBConj* :: *dbfm*  $\Rightarrow$  *dbfm*  $\Rightarrow$  *dbfm*

**where**  $\text{DBConj } t \ u \equiv \text{DBNeg } (\text{DBDisj } (\text{DBNeg } t) \ (\text{DBNeg } u))$

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

**lemma** *trans-tm-inject* [iff]:  $(\text{trans-tm } e \ t = \text{trans-tm } e \ u) \longleftrightarrow t = u$   
 ⟨proof⟩

**lemma** *trans-fm-inject* [iff]:  $(\text{trans-fm } e \ A = \text{trans-fm } e \ B) \longleftrightarrow A = B$   
 ⟨proof⟩

**lemma** *trans-fm-perm*:  
 assumes  $c: \text{atom } c \ \# \ (i, j, A, B)$   
 and  $t: \text{trans-fm } [i] \ A = \text{trans-fm } [j] \ B$   
 shows  $(i \leftrightarrow c) \cdot A = (j \leftrightarrow c) \cdot B$   
 ⟨proof⟩

## 2.2 Characterising the Well-Formed de Bruijn Formulas

### 2.2.1 Well-Formed Terms

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

**equivariance** *wf-dbtm*

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

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

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

**lemma** *wf-dbtm-trans-tm*:  $\text{wf-dbtm } (\text{trans-tm } [] \ t)$   
 ⟨proof⟩

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

**nominal-function** *abst-dbtm* ::  $\text{name} \Rightarrow \text{nat} \Rightarrow \text{dbtm} \Rightarrow \text{dbtm}$   
 where

$abst-dbtm\ name\ i\ DBZero = DBZero$   
 $|\ abst-dbtm\ name\ i\ (DBVar\ name') = (if\ name = name'\ then\ DBInd\ i\ else\ DBVar\ name')$   
 $|\ abst-dbtm\ name\ i\ (DBInd\ j) = DBInd\ j$   
 $|\ abst-dbtm\ name\ i\ (DBEats\ t1\ t2) = DBEats\ (abst-dbtm\ name\ i\ t1)\ (abst-dbtm\ name\ i\ t2)$   
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)

$\langle proof \rangle$

**nominal-function**  $subst-dbtm :: dbtm \Rightarrow name \Rightarrow dbtm \Rightarrow dbtm$

**where**

$subst-dbtm\ u\ i\ DBZero = DBZero$   
 $|\ subst-dbtm\ u\ i\ (DBVar\ name) = (if\ i = name\ then\ u\ else\ DBVar\ name)$   
 $|\ subst-dbtm\ u\ i\ (DBInd\ j) = DBInd\ j$   
 $|\ subst-dbtm\ u\ i\ (DBEats\ t1\ t2) = DBEats\ (subst-dbtm\ u\ i\ t1)\ (subst-dbtm\ u\ i\ t2)$   
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)

$\langle proof \rangle$

**lemma** *fresh-iff-non-subst-dbtm*:  $subst-dbtm\ DBZero\ i\ t = t \iff atom\ i\ \# t$

$\langle proof \rangle$

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

$\langle proof \rangle$

**lemma** *trans-tm-abs*:  $trans-tm\ (e@[name])\ t = abst-dbtm\ name\ (length\ e)\ (trans-tm\ e\ t)$

$\langle proof \rangle$

## 2.2.2 Well-Formed Formulas

**nominal-function**  $abst-dbfm :: name \Rightarrow nat \Rightarrow dbfm \Rightarrow dbfm$

**where**

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

**nominal-termination** (*eqvt*)

$\langle proof \rangle$

**nominal-function**  $subst\text{-}dbfm :: dbtm \Rightarrow name \Rightarrow dbfm \Rightarrow dbfm$

**where**

$subst\text{-}dbfm\ u\ i\ (DBMem\ t1\ t2) = DBMem\ (subst\text{-}dbtm\ u\ i\ t1)\ (subst\text{-}dbtm\ u\ i\ t2)$   
 $| subst\text{-}dbfm\ u\ i\ (DBEq\ t1\ t2) = DBEq\ (subst\text{-}dbtm\ u\ i\ t1)\ (subst\text{-}dbtm\ u\ i\ t2)$   
 $| subst\text{-}dbfm\ u\ i\ (DBDisj\ A1\ A2) = DBDisj\ (subst\text{-}dbfm\ u\ i\ A1)\ (subst\text{-}dbfm\ u\ i\ A2)$   
 $| subst\text{-}dbfm\ u\ i\ (DBNeg\ A) = DBNeg\ (subst\text{-}dbfm\ u\ i\ A)$   
 $| subst\text{-}dbfm\ u\ i\ (DBEx\ A) = DBEx\ (subst\text{-}dbfm\ u\ i\ A)$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt)$

$\langle proof \rangle$

**lemma**  $fresh\text{-}iff\text{-}non\text{-}subst\text{-}dbfm: subst\text{-}dbfm\ DBZero\ i\ t = t \longleftrightarrow atom\ i\ \# t$

$\langle proof \rangle$

## 2.3 Well formed terms and formulas (de Bruijn representation)

**inductive**  $wf\text{-}dbfm :: dbfm \Rightarrow bool$

**where**

$Mem: wf\text{-}dbtm\ t1 \Longrightarrow wf\text{-}dbtm\ t2 \Longrightarrow wf\text{-}dbfm\ (DBMem\ t1\ t2)$   
 $| Eq: wf\text{-}dbtm\ t1 \Longrightarrow wf\text{-}dbtm\ t2 \Longrightarrow wf\text{-}dbfm\ (DBEq\ t1\ t2)$   
 $| Disj: wf\text{-}dbfm\ A1 \Longrightarrow wf\text{-}dbfm\ A2 \Longrightarrow wf\text{-}dbfm\ (DBDisj\ A1\ A2)$   
 $| Neg: wf\text{-}dbfm\ A \Longrightarrow wf\text{-}dbfm\ (DBNeg\ A)$   
 $| Ex: wf\text{-}dbfm\ A \Longrightarrow wf\text{-}dbfm\ (DBEx\ (abst\text{-}dbfm\ name\ 0\ A))$

**equivariance**  $wf\text{-}dbfm$

**lemma**  $atom\text{-}fresh\text{-}abst\text{-}dbtm\ [simp]: atom\ i\ \#\ abst\text{-}dbtm\ i\ n\ t$

$\langle proof \rangle$

**lemma**  $atom\text{-}fresh\text{-}abst\text{-}dbfm\ [simp]: atom\ i\ \#\ abst\text{-}dbfm\ i\ n\ A$

$\langle proof \rangle$

Setting up strong induction: “avoiding” for name. Necessary to allow some proofs to go through

**nominal-inductive**  $wf\text{-}dbfm$

**avoids**  $Ex: name$

$\langle proof \rangle$

**inductive-cases**  $Mem\text{-}wf\text{-}dbfm\ [elim!]: wf\text{-}dbfm\ (DBMem\ t1\ t2)$

**inductive-cases**  $Eq\text{-}wf\text{-}dbfm\ [elim!]: wf\text{-}dbfm\ (DBEq\ t1\ t2)$

**inductive-cases**  $Disj\text{-}wf\text{-}dbfm\ [elim!]: wf\text{-}dbfm\ (DBDisj\ A1\ A2)$

**inductive-cases**  $Neg\text{-}wf\text{-}dbfm\ [elim!]: wf\text{-}dbfm\ (DBNeg\ A)$

**inductive-cases**  $Ex\text{-}wf\text{-}dbfm\ [elim!]: wf\text{-}dbfm\ (DBEx\ z)$

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

**lemma** *trans-fm-abs*:  $\text{trans-fm } (e@[name]) A = \text{abst-dbfm name } (\text{length } e) (\text{trans-fm } e A)$   
*<proof>*

**lemma** *abst-trans-fm*:  $\text{abst-dbfm name } 0 (\text{trans-fm } [] A) = \text{trans-fm } [name] A$   
*<proof>*

**lemma** *abst-trans-fm2*:  $i \neq j \implies \text{abst-dbfm } i (\text{Suc } 0) (\text{trans-fm } [j] A) = \text{trans-fm } [j, i] A$   
*<proof>*

**lemma** *wf-dbfm-imp-is-fm*:  
**assumes** *wf-dbfm* *x* **shows**  $\exists A::fm. x = \text{trans-fm } [] A$   
*<proof>*

**lemma** *wf-dbfm-trans-fm*:  $\text{wf-dbfm } (\text{trans-fm } [] A)$   
*<proof>*

**lemma** *wf-dbfm-iff-is-fm*:  $\text{wf-dbfm } x \longleftrightarrow (\exists A::fm. x = \text{trans-fm } [] A)$   
*<proof>*

**lemma** *dbtm-abst-ignore* [*simp*]:  
 $\text{abst-dbtm name } i (\text{abst-dbtm name } j t) = \text{abst-dbtm name } j t$   
*<proof>*

**lemma** *abst-dbtm-fresh-ignore* [*simp*]:  $\text{atom name } \# u \implies \text{abst-dbtm name } j u = u$   
*<proof>*

**lemma** *dbtm-subst-ignore* [*simp*]:  
 $\text{subst-dbtm } u \text{ name } (\text{abst-dbtm name } j t) = \text{abst-dbtm name } j t$   
*<proof>*

**lemma** *dbtm-abst-swap-subst*:  
 $\text{name } \neq \text{name}' \implies \text{atom name}' \# u \implies$   
 $\text{subst-dbtm } u \text{ name } (\text{abst-dbtm name}' j t) = \text{abst-dbtm name}' j (\text{subst-dbtm } u \text{ name } t)$   
*<proof>*

**lemma** *dbfm-abst-swap-subst*:  
 $\text{name } \neq \text{name}' \implies \text{atom name}' \# u \implies$   
 $\text{subst-dbfm } u \text{ name } (\text{abst-dbfm name}' j A) = \text{abst-dbfm name}' j (\text{subst-dbfm } u \text{ name } A)$   
*<proof>*

**lemma** *subst-trans-commute* [*simp*]:

$atom\ i \ \# \ e \implies subst\ dbtm\ (trans\ tm\ e\ u)\ i\ (trans\ tm\ e\ t) = trans\ tm\ e\ (subst\ i\ u\ t)$   
 ⟨proof⟩

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

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

## 2.4 Quotations

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

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

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

**lemma** *fresh-HTuple* [simp]:  $x \ \# \ HTuple\ n$   
 ⟨proof⟩

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

**lemma** *htuple-nonzero* [simp]:  $htuple\ k \neq 0$   
 ⟨proof⟩

**lemma** *htuple-inject* [iff]:  $htuple\ i = htuple\ j \longleftrightarrow i=j$   
 ⟨proof⟩

### 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]:  $nat\ of\ name\ n1 = nat\ of\ name\ n2 \longleftrightarrow n1 = n2$   
 ⟨proof⟩

**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*  
 ⟨proof⟩

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

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

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

Infinite support, so we cannot use nominal primrec.

**function** *quot-dbtm* :: *dbtm* ⇒ *tm*

**where**

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

**termination**

⟨proof⟩

**lemma** *quot-dbtm-inject-lemma* [simp]:  $\llbracket \text{quot-dbtm } t \rrbracket e = \llbracket \text{quot-dbtm } u \rrbracket e \longleftrightarrow t = u$   
 ⟨proof⟩

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

## 2.4.2 Quotations of de Bruijn formulas

Infinite support, so we cannot use nominal primrec.

**function** *quot-dbfm* :: *dbfm* ⇒ *tm*

**where**

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

**termination**

⟨proof⟩

```

lemma htuple-minus-1:  $n > 0 \implies \text{htuple } n = \langle 0, \text{htuple } (n - 1) \rangle$ 
  <proof>

lemma HTuple-minus-1:  $n > 0 \implies \text{HTuple } n = \text{HPair Zero } (\text{HTuple } (n - 1))$ 
  <proof>

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

lemma quot-dbfm-inject-lemma [simp]:  $\llbracket \text{quot-dbfm } A \rrbracket e = \llbracket \text{quot-dbfm } B \rrbracket e \longleftrightarrow A=B$ 
  <proof>

class quot =
  fixes quot :: 'a  $\Rightarrow$  tm ( $\langle\langle - \rangle\rangle$ )

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

  instance <proof>
end

lemma quot-dbtm-fresh [simp]:  $s \# (\text{quot-dbtm } t)$ 
  <proof>

lemma quot-tm-fresh [simp]: fixes t::tm shows  $s \# \langle\langle t \rangle\rangle$ 
  <proof>

lemma quot-Zero [simp]:  $\langle\langle \text{Zero} \rangle\rangle = \text{Zero}$ 
  <proof>

lemma quot-Var:  $\langle\langle \text{Var } x \rangle\rangle = \text{SUCC } (\text{ORD-OF } (\text{nat-of-name } x))$ 
  <proof>

lemma quot-Eats:  $\langle\langle \text{Eats } x \ y \rangle\rangle = \text{HPair } (\text{HTuple } 1) (\text{HPair } \langle\langle x \rangle\rangle \langle\langle y \rangle\rangle)$ 
  <proof>

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

lemma eval-quot-dbtm-ignore:
   $\llbracket \text{quot-dbtm } t \rrbracket e = \llbracket \text{quot-dbtm } t \rrbracket e'$ 
  <proof>

lemma eval-quot-dbfm-ignore:
   $\llbracket \text{quot-dbfm } A \rrbracket e = \llbracket \text{quot-dbfm } A \rrbracket e'$ 
  <proof>

```

```

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

  instance ⟨proof⟩
end

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

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

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

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

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

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

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

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

lemma eval-quot-fm-ignore: fixes A::fm shows [[«A»]]e = [[«A»]]e'
  ⟨proof⟩

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

```

## 2.5 Definitions Involving Coding

```

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

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

abbreviation Q-Eats :: tm ⇒ tm ⇒ tm

```

**where**  $Q\text{-Eats } t u \equiv \text{HPair } (\text{HTuple } (\text{Suc } 0)) (\text{HPair } t u)$

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

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

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

**lemma**  $\text{quot-Succ}: \langle \text{SUCC } x \rangle = Q\text{-Succ } \langle x \rangle$   
*<proof>*

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

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

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

**definition**  $q\text{-Mem} :: hf \Rightarrow hf \Rightarrow hf$   
**where**  $q\text{-Mem } x y \equiv \langle \text{htuple } 0, x, y \rangle$

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

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

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

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

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

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

**abbreviation**  $Q\text{-Conj} :: tm \Rightarrow tm \Rightarrow tm$

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

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

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

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

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

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

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

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

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

**lemma**  $q\text{-Eats-iff} \ [iff]: q\text{-Eats } x \ y = q\text{-Eats } x' \ y' \longleftrightarrow x=x' \wedge y=y'$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{quot-subst-eq}: \ll A(i::=t) \gg = \text{quot-dbfm } (\text{subst-dbfm } (\text{trans-tm } [] \ t) \ i) (\text{trans-fm}$   
 $[] \ A)$   
 $\langle \text{proof} \rangle$

**lemma**  $Q\text{-Succ-cong}: H \vdash x \ EQ \ x' \Longrightarrow H \vdash Q\text{-Succ } x \ EQ \ Q\text{-Succ } x'$   
 $\langle \text{proof} \rangle$

## 2.6 Quotations are Injective

### 2.6.1 Terms

**lemma**  $\text{eval-tm-inject} \ [simp]: \text{fixes } t::tm \text{ shows } \ll t \gg \ e = \ll u \gg \ e \longleftrightarrow t=u$   
 $\langle \text{proof} \rangle$

### 2.6.2 Formulas

**lemma**  $\text{eval-fm-inject} \ [simp]: \text{fixes } A::fm \text{ shows } \ll A \gg \ e = \ll B \gg \ e \longleftrightarrow A=B$   
 $\langle \text{proof} \rangle$

### 2.6.3 The set $\Gamma$ of Definition 1.1, constant terms used for coding

```

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

declare coding-tm.intros [intro]

lemma coding-tm-Zero [intro]: coding-tm Zero
  <proof>

lemma coding-tm-HTuple [intro]: coding-tm (HTuple k)
  <proof>

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

lemma quot-dbtm-coding [simp]: coding-tm (quot-dbtm t)
  <proof>

lemma quot-dbfm-coding [simp]: coding-tm (quot-dbfm fm)
  <proof>

lemma quot-fm-coding: fixes A::fm shows coding-tm  $\ll A \gg$ 
  <proof>

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

declare coding-hf.intros [intro]

lemma coding-hf-0 [intro]: coding-hf 0
  <proof>

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

lemma coding-tm-hf [simp]: coding-tm t  $\Longrightarrow \text{coding-hf } \llbracket t \rrbracket e$ 
  <proof>

```

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

Infinite support, so we cannot use nominal primrec.

```

function vquot-dbtm :: name set  $\Rightarrow$  dbtm  $\Rightarrow$  tm
  where

```

```

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

```

**termination**

⟨proof⟩

**lemma** *fresh-vquot-dbtm* [simp]:  $i \# vquot-dbtm V tm \longleftrightarrow i \# tm \vee i \notin atom \text{ ' } V$   
 ⟨proof⟩

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)
⟨proof⟩

```

**termination**

⟨proof⟩

**lemma** *fresh-vquot-dbfm* [simp]:  $i \# vquot-dbfm V fm \longleftrightarrow i \# fm \vee i \notin atom \text{ ' } V$   
 ⟨proof⟩

**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** ⟨proof⟩

**end**

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

**lemma** *vquot-tm-empty* [simp]: **fixes** t::*tm* **shows** [t]{} = <t>  
 ⟨proof⟩

**lemma** *vquot-dbtm-eq*:  $atom \text{ ' } V \cap supp \ t = atom \text{ ' } W \cap supp \ t \implies vquot-dbtm \ V \ t = vquot-dbtm \ W \ t$   
 ⟨*proof*⟩

**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** ⟨*proof*⟩

**end**

**lemma** *vquot-fm-fresh* [*simp*]: **fixes** *A*::*fm* **shows**  $i \# [A] \ V \longleftrightarrow i \# A \vee i \notin atom \text{ ' } V$   
 ⟨*proof*⟩

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

**lemma** *vquot-fm-empty* [*simp*]: **fixes** *A*::*fm* **shows**  $[A]\{\} = \ll A \gg$   
 ⟨*proof*⟩

**lemma** *vquot-dbfm-eq*:  $atom \text{ ' } V \cap supp \ A = atom \text{ ' } W \cap supp \ A \implies vquot-dbfm \ V \ A = vquot-dbfm \ W \ A$   
 ⟨*proof*⟩

**lemma** *vquot-fm-insert*:

**fixes** *A*::*fm* **shows**  $atom \ i \notin supp \ A \implies [A](insert \ i \ V) = [A] \ V$

⟨*proof*⟩

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

**end**



# Kapitel 3

## Basic Predicates

```
theory Predicates
imports SyntaxN
begin
```

### 3.1 The Subset Relation

```
nominal-function Subset :: tm  $\Rightarrow$  tm  $\Rightarrow$  fm (infixr SUBS 150)
  where atom z  $\#$  (t, u)  $\Longrightarrow$  t SUBS u = All2 z t ((Var z) IN u)
  <proof>
```

```
nominal-termination (eqvt)
  <proof>
```

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

```
lemma Subset-fresh-iff [simp]: a  $\#$  t SUBS u  $\longleftrightarrow$  a  $\#$  t  $\wedge$  a  $\#$  u
  <proof>
```

```
lemma eval-fm-Subset [simp]: eval-fm e (Subset t u)  $\longleftrightarrow$  ( $\llbracket t \rrbracket e \leq \llbracket u \rrbracket e$ )
  <proof>
```

```
lemma subst-fm-Subset [simp]: (t SUBS u)(i::=x) = (subst i x t) SUBS (subst i
x u)
  <proof>
```

```
lemma Subset-I:
  assumes insert ((Var i) IN t) H  $\vdash$  (Var i) IN u atom i  $\#$  (t,u)  $\forall B \in H. atom$ 
i  $\#$  B
  shows H  $\vdash$  t SUBS u
  <proof>
```

```
lemma Subset-D:
  assumes major: H  $\vdash$  t SUBS u and minor: H  $\vdash$  a IN t shows H  $\vdash$  a IN u
  <proof>
```

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

**lemma** *Subset-cong*:  $H \vdash t \text{ EQ } t' \implies H \vdash u \text{ EQ } u' \implies H \vdash t \text{ SUBS } u \text{ IFF } t' \text{ SUBS } u'$   
 ⟨proof⟩

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

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

**lemma** *Zero-SubsetE*:  $H \vdash A \implies \text{insert } (\text{Zero} \text{ SUBS } X) H \vdash A$   
 ⟨proof⟩

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

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

**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⟩

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

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

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

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

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

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

**lemma** *Subset-trans0*:  $\{ a \text{ SUBS } b, b \text{ SUBS } c \} \vdash a \text{ SUBS } c$   
 ⟨proof⟩

**lemma** *Subset-trans*:  $H \vdash a \text{ SUBS } b \implies H \vdash b \text{ SUBS } c \implies H \vdash a \text{ SUBS } c$   
 ⟨proof⟩

**lemma** *Subset-SUCC*:  $H \vdash a \text{ SUBS } (\text{SUCC } a)$   
 ⟨proof⟩

**lemma** *All2-Subset-lemma*:  $\text{atom } l \# (k', k) \implies \{P\} \vdash P' \implies \{ \text{All2 } l \ k \ P, k' \ \text{SUBS } k \} \vdash \text{All2 } l \ k' \ P'$   
 ⟨proof⟩

**lemma** *All2-Subset*:  $\llbracket H \vdash \text{All2 } l \ k \ P; H \vdash k' \ \text{SUBS } k; \{P\} \vdash P'; \text{atom } l \# (k', k) \rrbracket \implies H \vdash \text{All2 } l \ k' \ P'$   
 ⟨proof⟩

## 3.2 Extensionality

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

**lemma** *Equality-I*:  $H \vdash y \text{ SUBS } x \implies H \vdash x \text{ SUBS } y \implies H \vdash x \text{ EQ } y$   
 ⟨proof⟩

**lemma** *EQ-imp-SUBS*:  $\text{insert } (t \text{ EQ } u) \ H \vdash (t \text{ SUBS } u)$   
 ⟨proof⟩

**lemma** *EQ-imp-SUBS2*:  $\text{insert } (u \text{ EQ } t) \ H \vdash (t \text{ SUBS } u)$   
 ⟨proof⟩

**lemma** *Equality-E*:  $\text{insert } (t \text{ SUBS } u) \ (\text{insert } (u \text{ SUBS } t) \ H) \vdash A \implies \text{insert } (t \text{ EQ } u) \ H \vdash A$   
 ⟨proof⟩

## 3.3 The Disjointness Relation

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

**nominal-function** *Disjoint* ::  $tm \Rightarrow tm \Rightarrow fm$   
**where**  $\text{atom } z \# (t, u) \implies \text{Disjoint } t \ u = \text{All2 } z \ t \ (\text{Neg } ((\text{Var } z) \ \text{IN } u))$   
 ⟨proof⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**lemma** *Disjoint-Eats-E2*:

$insert (Disjoint z x) (insert (Neg(y IN z)) H) \vdash A \implies insert (Disjoint z (Eats x y)) H \vdash A$   
 ⟨proof⟩

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

**lemma** *Disjoint-Eats-I* [intro!]:  $H \vdash Disjoint x z \implies insert (y IN z) H \vdash Fls \implies H \vdash Disjoint (Eats x y) z$   
 ⟨proof⟩

**lemma** *Disjoint-Eats-I2* [intro!]:  $H \vdash Disjoint z x \implies insert (y IN z) H \vdash Fls \implies H \vdash Disjoint z (Eats x y)$   
 ⟨proof⟩

### 3.4 The Foundation Theorem

**lemma** *Foundation-lemma*:

**assumes**  $i: atom\ i \# z$   
**shows**  $\{ All2\ i\ z\ (Neg\ (Disjoint\ (Var\ i)\ z)) \} \vdash Neg\ (Var\ i\ IN\ z)\ AND\ Disjoint\ (Var\ i)\ z$   
 ⟨proof⟩

**theorem** *Foundation*:  $atom\ i \# z \implies \{\} \vdash All2\ i\ z\ (Neg\ (Disjoint\ (Var\ i)\ z))$   
*IMP z EQ Zero*  
 ⟨proof⟩

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

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

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

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

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

### 3.5 The Ordinal Property

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

**where**  $\llbracket atom\ y \# (x, z); atom\ z \# x \rrbracket \implies$

$OrdP\ x = All2\ y\ x\ ((Var\ y)\ SUBS\ x\ AND\ All2\ z\ (Var\ y)\ ((Var\ z)\ SUBS\ (Var$

y)))  
(proof)

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

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

**lemma** *subst-fm-OrdP* [simp]:  $(\text{OrdP } t)(i::=x) = \text{OrdP } (\text{subst } i \ x \ t)$   
(proof)

**lemma** *OrdP-cong*:  $H \vdash x \text{ EQ } x' \implies H \vdash \text{OrdP } x \text{ IFF } \text{OrdP } x'$   
(proof)

**lemma** *OrdP-Mem-lemma*:  
**assumes**  $z$ : *atom*  $z \# (k,l)$  **and**  $l$ : *insert*  $(\text{OrdP } k) \ H \vdash l \text{ IN } k$   
**shows** *insert*  $(\text{OrdP } k) \ H \vdash l \text{ SUBS } k \text{ AND } \text{All2 } z \ l \ (\text{Var } z \ \text{SUBS } l)$   
(proof)

**lemma** *OrdP-Mem-E*:  
**assumes** *atom*  $z \# (k,l)$   
*insert*  $(\text{OrdP } k) \ H \vdash l \text{ IN } k$   
*insert*  $(l \ \text{SUBS } k) \ (\text{insert } (\text{All2 } z \ l \ (\text{Var } z \ \text{SUBS } l)) \ H) \vdash A$   
**shows** *insert*  $(\text{OrdP } k) \ H \vdash A$   
(proof)

**lemma** *OrdP-Mem-imp-Subset*:  
**assumes**  $k$ :  $H \vdash k \text{ IN } l$  **and**  $l$ :  $H \vdash \text{OrdP } l$  **shows**  $H \vdash k \text{ SUBS } l$   
(proof)

**lemma** *SUCC-Subset-Ord-lemma*:  $\{ k' \text{ IN } k, \text{OrdP } k \} \vdash \text{SUCC } k' \text{ SUBS } k$   
(proof)

**lemma** *SUCC-Subset-Ord*:  $H \vdash k' \text{ IN } k \implies H \vdash \text{OrdP } k \implies H \vdash \text{SUCC } k' \text{ SUBS } k$   
(proof)

**lemma** *OrdP-Trans-lemma*:  $\{ \text{OrdP } k, i \text{ IN } j, j \text{ IN } k \} \vdash i \text{ IN } k$   
(proof)

**lemma** *OrdP-Trans*:  $H \vdash \text{OrdP } k \implies H \vdash i \text{ IN } j \implies H \vdash j \text{ IN } k \implies H \vdash i \text{ IN } k$   
(proof)

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

**shows**  $\text{insert } (\text{OrdP } k) H \vdash \text{OrdP } l$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{Ord-IN-Ord}$ :  $H \vdash l \text{ IN } k \implies H \vdash \text{OrdP } k \implies H \vdash \text{OrdP } l$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{OrdP-I}$ :

**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**  $\text{atom } y \# (x, z) \vee B \in H. \text{atom } y \# B \text{ atom } z \# x \vee B \in H. \text{atom } z \# B$   
**shows**  $H \vdash \text{OrdP } x$   
 $\langle \text{proof} \rangle$

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

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

**lemma**  $\text{OrdP-SUCC-I}$ :  $H \vdash \text{OrdP } k \implies H \vdash \text{OrdP } (\text{SUCC } k)$   
 $\langle \text{proof} \rangle$

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

**lemma**  $\text{OrdP-HPairE}$ :  $\text{insert } (\text{OrdP } (\text{HPair } x y)) H \vdash A$   
 $\langle \text{proof} \rangle$

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

**lemma**  $\text{Zero-Eq-HPairE}$ :  $\text{insert } (\text{Zero} \text{ EQ } \text{HPair } x y) H \vdash A$   
 $\langle \text{proof} \rangle$

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

**lemma**  $\text{HPair-Eq-ZeroE}$ :  $\text{insert } (\text{HPair } x y \text{ EQ } \text{Zero}) H \vdash A$   
 $\langle \text{proof} \rangle$

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

### 3.6 Induction on Ordinals

**lemma** *OrdInd-lemma*:

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

**lemma** *OrdInd*:

**assumes**  $j$ : *atom* ( $j::name$ )  $\#$  ( $i,A$ )  
**and**  $x$ :  $H \vdash OrdP (Var i)$  **and**  $step$ :  $H \vdash All i (OrdP (Var i) IMP (All2 j (Var i) (A(i::= Var j)) IMP A))$   
**shows**  $H \vdash A$   
 $\langle proof \rangle$

**lemma** *OrdIndH*:

**assumes** *atom* ( $j::name$ )  $\#$  ( $i,A$ )  
**and**  $H \vdash All i (OrdP (Var i) IMP (All2 j (Var i) (A(i::= Var j)) IMP A))$   
**shows**  $insert (OrdP (Var i)) H \vdash A$   
 $\langle proof \rangle$

### 3.7 Linearity of Ordinals

**lemma** *OrdP-linear-lemma*:

**assumes**  $j$ : *atom*  $j \# i$   
**shows**  $\{ OrdP (Var i) \} \vdash All j (OrdP (Var j) IMP (Var i IN Var j OR Var i EQ Var j OR Var j IN Var i))$   
(is -  $\vdash$  ?*scheme*)  
 $\langle proof \rangle$

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

**lemma** *OrdP-linear*:

**assumes**  $H \vdash OrdP x$   $H \vdash OrdP y$   
 $insert (x IN y) H \vdash A$   $insert (x EQ y) H \vdash A$   $insert (y IN x) H \vdash A$   
**shows**  $H \vdash A$   
 $\langle proof \rangle$



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

### 3.8 The predicate *OrdNotEqP*

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

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

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

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

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

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

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

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

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

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

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

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

### 3.9 Predecessor of an Ordinal

**lemma** *OrdP-set-max-lemma*:

**assumes**  $j$ : *atom* ( $j::name$ )  $\#$   $i$  **and**  $k$ : *atom* ( $k::name$ )  $\#$  ( $i,j$ )  
**shows**  $\{ \} \vdash (\text{Neg } (Var\ i\ EQ\ Zero)\ AND\ (All2\ j\ (Var\ i)\ (OrdP\ (Var\ j))))\ IMP$   
 $(Ex\ j\ (Var\ j\ IN\ Var\ i\ AND\ (All2\ k\ (Var\ i)\ (Var\ k\ SUBS\ Var\ j))))$   
 $\langle proof \rangle$

**lemma** *OrdP-max-imp*:

**assumes**  $j$ : *atom*  $j$   $\#$  ( $x$ ) **and**  $k$ : *atom*  $k$   $\#$  ( $x,j$ )  
**shows**  $\{ OrdP\ x,\ Neg\ (x\ EQ\ Zero)\ } \vdash Ex\ j\ (Var\ j\ IN\ x\ AND\ (All2\ k\ x\ (Var\ k\ SUBS\ Var\ j)))$   
 $\langle proof \rangle$

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

### 3.10 Case Analysis and Zero/SUCC Induction

**lemma** *OrdP-cases-lemma*:

**assumes**  $p$ : *atom*  $p$   $\#$   $x$   
**shows**  $\{ OrdP\ x,\ Neg\ (x\ EQ\ Zero)\ } \vdash Ex\ p\ (OrdP\ (Var\ p)\ AND\ x\ EQ\ SUCC$   
 $(Var\ p))$   
 $\langle proof \rangle$

**lemma** *OrdP-cases-disj*:

**assumes**  $p$ : *atom*  $p$   $\#$   $x$   
**shows**  $insert\ (OrdP\ x)\ H \vdash x\ EQ\ Zero\ OR\ Ex\ p\ (OrdP\ (Var\ p)\ AND\ x\ EQ\ SUCC$   
 $(Var\ p))$   
 $\langle proof \rangle$

**lemma** *OrdP-cases-E*:

$\llbracket insert\ (x\ EQ\ Zero)\ H \vdash A;$   
 $insert\ (x\ EQ\ SUCC\ (Var\ k))\ (insert\ (OrdP\ (Var\ k))\ H) \vdash A;$   
 $atom\ k\ \# (x,A); \ \forall C \in H. atom\ k\ \# C \rrbracket$   
 $\implies insert\ (OrdP\ x)\ H \vdash A$   
 $\langle proof \rangle$

**lemma** *OrdInd2-lemma*:

$\{ OrdP\ (Var\ i),\ A(i::= Zero),\ (All\ i\ (OrdP\ (Var\ i)\ IMP\ A\ IMP\ (A(i::= SUCC$   
 $(Var\ i)))) \} \vdash A$   
 $\langle proof \rangle$

**lemma** *OrdInd2*:

**assumes**  $H \vdash OrdP\ (Var\ i)$   
**and**  $H \vdash A(i::= Zero)$   
**and**  $H \vdash All\ i\ (OrdP\ (Var\ i)\ IMP\ A\ IMP\ (A(i::= SUCC\ (Var\ i))))$   
**shows**  $H \vdash A$   
 $\langle proof \rangle$

**lemma** *OrdInd2H*:

**assumes**  $H \vdash A(i ::= \text{Zero})$

**and**  $H \vdash \text{All } i \text{ (OrdP (Var } i \text{) IMP } A \text{ IMP (} A(i ::= \text{SUCC (Var } i \text{))})$

**shows**  $\text{insert (OrdP (Var } i \text{)) } H \vdash A$

*<proof>*

### 3.11 The predicate *HFun-Sigma*

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

See the note after the proof of Lemma 2.3.

**definition** *hfun-sigma* **where**

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

**definition** *hfun-sigma-ord* **where**

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

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

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

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

$\implies$

$\text{HFun-Sigma } r =$

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

$y')$

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

*<proof>*

**nominal-termination** (*eqvt*)

*<proof>*

**lemma**

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

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

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

*<proof>*

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

*<proof>*

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

*<proof>*

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

*<proof>*

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

**lemma** *HFun-Sigma-Mem-imp-HPair*:

**assumes**  $H \vdash \text{HFun-Sigma } r \ H \vdash a \text{ IN } r$

**and**  $xy$ :  $\text{atom } x \# (y, a, r) \ \text{atom } y \# (a, r)$

**shows**  $H \vdash (\text{Ex } x (\text{Ex } y (a \text{ EQ } \text{HPair } (\text{Var } x) (\text{Var } y)))) \ (\text{is } - \vdash ?\text{concl})$

*<proof>*

### 3.12 The predicate *HDomain-Incl*

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

**nominal-function** *HDomain-Incl* ::  $tm \Rightarrow tm \Rightarrow fm$

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

$\text{HDomain-Incl } r \ d = \text{All2 } x \ d (\text{Ex } y (\text{Ex } z (\text{Var } z \text{ IN } r \ \text{AND } \text{Var } z \text{ EQ } \text{HPair } (\text{Var } x) (\text{Var } y))))$

*<proof>*

**nominal-termination** (*eqvt*)

*<proof>*

**lemma**

**shows** *HDomain-Incl-fresh-iff* [*simp*]:

$a \# \text{HDomain-Incl } r \ d \longleftrightarrow a \# r \wedge a \# d \ (\text{is } ?\text{thesis1})$

**and** *eval-fm-HDomain-Incl* [*simp*]:

$\text{eval-fm } e (\text{HDomain-Incl } r \ d) \longleftrightarrow \llbracket d \rrbracket e \leq \text{hdomain } \llbracket r \rrbracket e \ (\text{is } ?\text{thesis2})$

*<proof>*

**lemma** *HDomain-Incl-subst* [*simp*]:

$(\text{HDomain-Incl } r \ d)(i ::= t) = \text{HDomain-Incl } (\text{subst } i \ t \ r) (\text{subst } i \ t \ d)$

*<proof>*

**lemma** *HDomain-Incl-Subset-lemma*:  $\{ \text{HDomain-Incl } r \ k, k' \text{ SUBS } k \} \vdash \text{HDomain-Incl } r \ k'$

*<proof>*

**lemma** *HDomain-Incl-Subset*:  $H \vdash \text{HDomain-Incl } r \ k \Longrightarrow H \vdash k' \text{ SUBS } k \Longrightarrow H \vdash \text{HDomain-Incl } r \ k'$

*<proof>*

**lemma** *HDomain-Incl-Mem-Ord*:  $H \vdash \text{HDomain-Incl } r \ k \Longrightarrow H \vdash k' \text{ IN } k \Longrightarrow H \vdash \text{OrdP } k \Longrightarrow H \vdash \text{HDomain-Incl } r \ k'$

*<proof>*

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

*<proof>*

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

**lemma** *HDomain-Incl-Eats-I*:  $H \vdash \text{HDomain-Incl } r \ d \implies H \vdash \text{HDomain-Incl } (\text{Eats } r \ (\text{HPair } d \ d')) \ (\text{SUCC } d)$   
 $\langle \text{proof} \rangle$

### 3.13 *HPair* is Provably Injective

**lemma** *Doubleton-E*:

**assumes**  $\text{insert } (a \ \text{EQ } c) \ (\text{insert } (b \ \text{EQ } d) \ H) \vdash A$   
 $\text{insert } (a \ \text{EQ } d) \ (\text{insert } (b \ \text{EQ } c) \ H) \vdash A$

**shows**  $\text{insert } ((\text{Eats } (\text{Eats } \text{Zero } b) \ a) \ \text{EQ } (\text{Eats } (\text{Eats } \text{Zero } d) \ c)) \ H \vdash A$   
 $\langle \text{proof} \rangle$

**lemma** *HFST*:  $\{ \text{HPair } a \ b \ \text{EQ } \text{HPair } c \ d \} \vdash a \ \text{EQ } c$   
 $\langle \text{proof} \rangle$

**lemma** *b-EQ-d-1*:  $\{ a \ \text{EQ } c, \ a \ \text{EQ } d, \ b \ \text{EQ } c \} \vdash b \ \text{EQ } d$   
 $\langle \text{proof} \rangle$

**lemma** *HSND*:  $\{ \text{HPair } a \ b \ \text{EQ } \text{HPair } c \ d \} \vdash b \ \text{EQ } d$   
 $\langle \text{proof} \rangle$

**lemma** *HPair-E* [*intro!*]:

**assumes**  $\text{insert } (a \ \text{EQ } c) \ (\text{insert } (b \ \text{EQ } d) \ H) \vdash A$   
**shows**  $\text{insert } (\text{HPair } a \ b \ \text{EQ } \text{HPair } c \ d) \ H \vdash A$

$\langle \text{proof} \rangle$

**declare** *HPair-E* [*THEN rotate2, intro!*]

**declare** *HPair-E* [*THEN rotate3, intro!*]

**declare** *HPair-E* [*THEN rotate4, intro!*]

**declare** *HPair-E* [*THEN rotate5, intro!*]

**declare** *HPair-E* [*THEN rotate6, intro!*]

**declare** *HPair-E* [*THEN rotate7, intro!*]

**declare** *HPair-E* [*THEN rotate8, intro!*]

**lemma** *HFun-Sigma-E*:

**assumes**  $r: H \vdash \text{HFun-Sigma } r$

**and**  $b: H \vdash \text{HPair } a \ b \ \text{IN } r$

**and**  $b': H \vdash \text{HPair } a \ b' \ \text{IN } r$

**shows**  $H \vdash b \ \text{EQ } b'$

$\langle \text{proof} \rangle$

### 3.14 *SUCC* is Provably Injective

**lemma** *SUCC-SUBS-lemma*:  $\{ \text{SUCC } x \ \text{SUBS } \text{SUCC } y \} \vdash x \ \text{SUBS } y$

*<proof>*

**lemma** *SUCC-SUBS*: *insert (SUCC x SUBS SUCC y) H*  $\vdash$  *x SUBS y*  
*<proof>*

**lemma** *SUCC-inject*: *insert (SUCC x EQ SUCC y) H*  $\vdash$  *x EQ y*  
*<proof>*

**lemma** *SUCC-inject-E* [*intro!*]: *insert (x EQ y) H*  $\vdash$  *A*  $\implies$  *insert (SUCC x EQ SUCC y) H*  $\vdash$  *A*  
*<proof>*

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

**lemma** *OrdP-IN-SUCC-lemma*:  $\{ \text{OrdP } x, y \text{ IN } x \} \vdash \text{SUCC } y \text{ IN } \text{SUCC } x$   
*<proof>*

**lemma** *OrdP-IN-SUCC*:  $H \vdash \text{OrdP } x \implies H \vdash y \text{ IN } x \implies H \vdash \text{SUCC } y \text{ IN } \text{SUCC } x$   
*<proof>*

**lemma** *OrdP-IN-SUCC-D-lemma*:  $\{ \text{OrdP } x, \text{SUCC } y \text{ IN } \text{SUCC } x \} \vdash y \text{ IN } x$   
*<proof>*

**lemma** *OrdP-IN-SUCC-D*:  $H \vdash \text{OrdP } x \implies H \vdash \text{SUCC } y \text{ IN } \text{SUCC } x \implies H \vdash y \text{ IN } x$   
*<proof>*

**lemma** *OrdP-IN-SUCC-Iff*:  $H \vdash \text{OrdP } y \implies H \vdash \text{SUCC } x \text{ IN } \text{SUCC } y \text{ IFF } x \text{ IN } y$   
*<proof>*

### 3.15 The predicate *LstSeqP*

**lemma** *hfun-sigma-ord-iff*:  $\text{hfun-sigma-ord } s \iff \text{OrdDom } s \wedge \text{hfun-sigma } s$   
*<proof>*

**lemma** *hfun-sigma-iff*:  $\text{hfun-sigma } r \iff \text{hfunction } r \wedge \text{hrelation } r$   
*<proof>*

**lemma** *Seq-iff*:  $\text{Seq } r \text{ } d \iff d \leq \text{hdomain } r \wedge \text{hfun-sigma } r$   
*<proof>*

**lemma** *LstSeq-iff*:  $LstSeq\ s\ k\ y \longleftrightarrow succ\ k \leq hdomain\ s \wedge \langle k, y \rangle \in s \wedge hfun\text{-sigma}\text{-ord}\ s$

*<proof>*

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

**where**

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

*<proof>*

**nominal-termination** (*eqvt*)

*<proof>*

**lemma**

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

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

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

$eval\text{-}fm\ e\ (LstSeqP\ s\ k\ y) \longleftrightarrow LstSeq\ [s]e\ [k]e\ [y]e$  (**is** *?thesis2*)

*<proof>*

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

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

*<proof>*

**lemma** *LstSeqP-E*:

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

$(insert\ (OrdP\ k)\ (insert\ (HFun\text{-}Sigma\ s)$

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

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

*<proof>*

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

**lemma** *LstSeqP-cong*:

**assumes**  $H \vdash s\ EQ\ s'\ H \vdash k\ EQ\ k'\ H \vdash y\ EQ\ y'$

**shows**  $H \vdash LstSeqP\ s\ k\ y\ IFF\ LstSeqP\ s'\ k'\ y'$

*<proof>*

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

*<proof>*

**lemma** *LstSeqP-Mem-lemma*:  $\{ LstSeqP\ r\ k\ y, HPair\ k'\ z\ IN\ r, k'\ IN\ k \} \vdash LstSeqP\ r\ k'\ z$

*<proof>*

**lemma** *LstSeqP-Mem*:  $H \vdash LstSeqP\ r\ k\ y \Longrightarrow H \vdash HPair\ k'\ z\ IN\ r \Longrightarrow H \vdash k'\ IN\ k \Longrightarrow H \vdash LstSeqP\ r\ k'\ z$

*<proof>*

**lemma** *LstSeqP-imp-Mem*:  $H \vdash \text{LstSeqP } s \ k \ y \implies H \vdash \text{HPair } k \ y \ \text{IN } s$   
*<proof>*

**lemma** *LstSeqP-SUCC*:  $H \vdash \text{LstSeqP } r \ (\text{SUCC } d) \ y \implies H \vdash \text{HPair } d \ z \ \text{IN } r \implies$   
 $H \vdash \text{LstSeqP } r \ d \ z$   
*<proof>*

**lemma** *LstSeqP-EQ*:  $\llbracket H \vdash \text{LstSeqP } s \ k \ y; H \vdash \text{HPair } k \ y' \ \text{IN } s \rrbracket \implies H \vdash y \ \text{EQ}$   
 $y'$   
*<proof>*

**end**



# Kapitel 4

## Sigma-Formulas and Theorem 2.5

```
theory Sigma
imports Predicates
begin
```

### 4.1 Ground Terms and Formulas

```
definition ground-aux :: tm  $\Rightarrow$  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)
<proof>
```

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

$ground\text{-}fm\text{-}aux (Neg A) S = (ground\text{-}fm\text{-}aux A S)$   
 $ground\text{-}fm\text{-}aux (Ex x A) S = (ground\text{-}fm\text{-}aux A (S \cup \{atom\ x\}))$   
 $\langle proof \rangle$

**lemma** *ground-fresh*[*simp*]:  
 $ground\ t \implies atom\ i \# t$   
 $ground\text{-}fm\ A \implies atom\ i \# A$   
 $\langle proof \rangle$

## 4.2 Sigma Formulas

Section 2 material

### 4.2.1 Strict Sigma Formulas

Definition 2.1

**inductive** *ss-fm* :: *fm*  $\Rightarrow$  *bool* **where**  
 $MemI: ss\text{-}fm (Var\ i\ IN\ Var\ j)$   
 $| DisjI: ss\text{-}fm\ A \implies ss\text{-}fm\ B \implies ss\text{-}fm (A\ OR\ B)$   
 $| ConjI: ss\text{-}fm\ A \implies ss\text{-}fm\ B \implies ss\text{-}fm (A\ AND\ B)$   
 $| ExI: ss\text{-}fm\ A \implies ss\text{-}fm (Ex\ i\ A)$   
 $| All2I: ss\text{-}fm\ A \implies atom\ j \# (i,A) \implies ss\text{-}fm (All2\ i\ (Var\ j)\ A)$

**equivariance** *ss-fm*

**nominal-inductive** *ss-fm*  
**avoids** *ExI*: *i* | *All2I*: *i*  
 $\langle proof \rangle$

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

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

**lemma** *Sigma-fm-Iff*:  $\{\} \vdash B\ IFF\ A; supp\ A \subseteq supp\ B; Sigma\text{-}fm\ A \implies Sigma\text{-}fm\ B$   
 $\langle proof \rangle$

**lemma** *ss-fm-imp-Sigma-fm* [*intro*]:  $ss\text{-}fm\ A \implies Sigma\text{-}fm\ A$   
 $\langle proof \rangle$

**lemma** *Sigma-fm-Fls* [*iff*]:  $Sigma\text{-}fm\ Fls$   
 $\langle proof \rangle$

### 4.2.2 Closure properties for Sigma-formulas

**lemma**

**assumes**  $\text{Sigma-fm } A \text{ Sigma-fm } B$   
**shows**  $\text{Sigma-fm-AND [intro!]: Sigma-fm } (A \text{ AND } B)$   
**and**  $\text{Sigma-fm-OR [intro!]: Sigma-fm } (A \text{ OR } B)$   
**and**  $\text{Sigma-fm-Ex [intro!]: Sigma-fm } (Ex \ i \ A)$   
 <proof>

**lemma**  $\text{Sigma-fm-All2-Var:}$   
**assumes**  $H0: \text{Sigma-fm } A \text{ and } ij: \text{atom } j \ \# \ (i, A)$   
**shows**  $\text{Sigma-fm } (All2 \ i \ (Var \ j) \ A)$   
 <proof>

### 4.3 Lemma 2.2: Atomic formulas are Sigma-formulas

**lemma**  $\text{Eq-Eats-Iff:}$   
**assumes**  $[\text{unfolded fresh-Pair, simp}]: \text{atom } i \ \# \ (z, x, y)$   
**shows**  $\{\} \vdash z \ EQ \ Eats \ x \ y \ IFF \ (All2 \ i \ z \ (Var \ i \ IN \ x \ OR \ Var \ i \ EQ \ y)) \ AND \ x$   
 $SUBS \ z \ AND \ y \ IN \ z$   
 <proof>

**lemma**  $\text{Subset-Zero-sf: Sigma-fm } (Var \ i \ SUBS \ Zero)$   
 <proof>

**lemma**  $\text{Eq-Zero-sf: Sigma-fm } (Var \ i \ EQ \ Zero)$   
 <proof>

**lemma**  $\text{theorem-sf: assumes } \{\} \vdash A \text{ shows Sigma-fm } A$   
 <proof>

The subset relation

**lemma**  $\text{Var-Subset-sf: Sigma-fm } (Var \ i \ SUBS \ Var \ j)$   
 <proof>

**lemma**  $\text{Zero-Mem-sf: Sigma-fm } (Zero \ IN \ Var \ i)$   
 <proof>

**lemma**  $ijk: i + k < Suc \ (i + j + k)$   
 <proof>

**lemma**  $\text{All2-term-Iff-fresh: } i \neq j \implies \text{atom } j' \ \# \ (i, j, A) \implies$   
 $\{\} \vdash (All2 \ i \ (Var \ j) \ A) \ IFF \ Ex \ j' \ (Var \ j \ EQ \ Var \ j' \ AND \ All2 \ i \ (Var \ j') \ A)$   
 <proof>

**lemma**  $\text{Sigma-fm-All2-fresh:}$   
**assumes**  $\text{Sigma-fm } A \ i \neq j$   
**shows**  $\text{Sigma-fm } (All2 \ i \ (Var \ j) \ A)$   
 <proof>

**lemma**  $\text{Subset-Eats-sf:}$   
**assumes**  $\bigwedge j::\text{name. Sigma-fm } (Var \ j \ IN \ t)$

**and**  $\bigwedge k::name. \text{Sigma-fm } (Var\ k\ EQ\ u)$   
**shows**  $\text{Sigma-fm } (Var\ i\ SUBS\ Eats\ t\ u)$   
 $\langle proof \rangle$

**lemma** *Eq-Eats-sf*:  
**assumes**  $\bigwedge j::name. \text{Sigma-fm } (Var\ j\ EQ\ t)$   
**and**  $\bigwedge k::name. \text{Sigma-fm } (Var\ k\ EQ\ u)$   
**shows**  $\text{Sigma-fm } (Var\ i\ EQ\ Eats\ t\ u)$   
 $\langle proof \rangle$

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

**lemma** *Subset-Mem-sf-lemma*:  
 $size\ t + size\ u < n \implies \text{Sigma-fm } (t\ SUBS\ u) \wedge \text{Sigma-fm } (t\ IN\ u)$   
 $\langle proof \rangle$

**lemma** *Subset-sf [iff]*:  $\text{Sigma-fm } (t\ SUBS\ u)$   
 $\langle proof \rangle$

**lemma** *Mem-sf [iff]*:  $\text{Sigma-fm } (t\ IN\ u)$   
 $\langle proof \rangle$

The equality relation is a Sigma-Formula

**lemma** *Equality-sf [iff]*:  $\text{Sigma-fm } (t\ EQ\ u)$   
 $\langle proof \rangle$

## 4.4 Universal Quantification Bounded by an Arbitrary Term

**lemma** *All2-term-Iff*:  $atom\ i \# t \implies atom\ j \# (i, t, A) \implies$   
 $\{\} \vdash (All2\ i\ t\ A) \text{ IFF } Ex\ j\ (Var\ j\ EQ\ t\ AND\ All2\ i\ (Var\ j)\ A)$   
 $\langle proof \rangle$

**lemma** *Sigma-fm-All2 [intro!]*:  
**assumes**  $\text{Sigma-fm } A\ atom\ i \# t$   
**shows**  $\text{Sigma-fm } (All2\ i\ t\ A)$   
 $\langle proof \rangle$

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

**lemma** *OrdP-sf [iff]*:  $\text{Sigma-fm } (OrdP\ t)$   
 $\langle proof \rangle$

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

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

**lemma** *HFun-Sigma-Iff*:

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

**shows**

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

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

*y'*)

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

⟨proof⟩

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

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

## 4.6 A Key Result: Theorem 2.5

### 4.6.1 Sigma-Eats Formulas

**inductive** *se-fm* :: *fm*  $\Rightarrow$  *bool* **where**

*MemI*: *se-fm* (*t IN u*)

| *DisjI*: *se-fm* *A*  $\Longrightarrow$  *se-fm* *B*  $\Longrightarrow$  *se-fm* (*A OR B*)

| *ConjI*: *se-fm* *A*  $\Longrightarrow$  *se-fm* *B*  $\Longrightarrow$  *se-fm* (*A AND B*)

| *ExI*: *se-fm* *A*  $\Longrightarrow$  *se-fm* (*Ex i A*)

| *All2I*: *se-fm* *A*  $\Longrightarrow$  *atom* *i*  $\#$  *t*  $\Longrightarrow$  *se-fm* (*All2 i t A*)

**equivariance** *se-fm*

**nominal-inductive** *se-fm*

**avoids** *ExI*: *i* | *All2I*: *i*

⟨proof⟩

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

**lemma** *subst-fm-in-se-fm*: *se-fm* *A*  $\Longrightarrow$  *se-fm* (*A*(*k*::=*x*))  
 ⟨proof⟩

## 4.6.2 Preparation

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

**lemma** *obtain-const-tm*: **obtains  $t$  where**  $\llbracket t \rrbracket e = x$  *ground  $t$*   
 $\langle proof \rangle$

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

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

**lemma** *ex-eval-fm-iff-exists-tm'*:  
 $eval\text{-}fm\ e\ (Ex\ k\ A) \longleftrightarrow (\exists t. eval\text{-}fm\ e\ (A(k::=t)))$   
 $\langle proof \rangle$

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

**nominal-function** *elts* ::  $tm \Rightarrow tm$  *set where*  
 $elts\ Zero = \{\}$   
 $| elts\ (Var\ k) = \{\}$   
 $| elts\ (Eats\ t\ u) = insert\ u\ (elts\ t)$   
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)  
 $\langle proof \rangle$

**lemma** *eval-fm-All2-Eats*:  
 $atom\ i\ \sharp\ (t, u) \Longrightarrow$   
 $eval\text{-}fm\ e\ (All2\ i\ (Eats\ t\ u)\ A) \longleftrightarrow eval\text{-}fm\ e\ (A(i::=u)) \wedge eval\text{-}fm\ e\ (All2\ i\ t\ A)$   
 $\langle proof \rangle$

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

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

**lemma** *prove-elts-imp-prove-All2*:  
 $ground\ t \Longrightarrow (\bigwedge u. u \in elts\ t \Longrightarrow \{\} \vdash A(i::=u)) \Longrightarrow \{\} \vdash All2\ i\ t\ A$   
 $\langle proof \rangle$

## 4.6.3 The base cases: ground atomic formulas

**lemma** *ground-prove*:  
 $\llbracket size\ t + size\ u < n; ground\ t; ground\ u \rrbracket$   
 $\Longrightarrow (\llbracket t \rrbracket e \leq \llbracket u \rrbracket e \longrightarrow \{\} \vdash t\ SUBS\ u) \wedge (\llbracket t \rrbracket e \in \llbracket u \rrbracket e \longrightarrow \{\} \vdash t\ IN\ u)$   
 $\langle proof \rangle$

**lemma**

**assumes** *ground t ground u*  
**shows** *ground-prove-SUBS*:  $\llbracket t \rrbracket e \leq \llbracket u \rrbracket e \implies \{\} \vdash t \text{ SUBS } u$   
**and** *ground-prove-IN*:  $\llbracket t \rrbracket e \in \llbracket u \rrbracket e \implies \{\} \vdash t \text{ IN } u$   
**and** *ground-prove-EQ*:  $\llbracket t \rrbracket e = \llbracket u \rrbracket e \implies \{\} \vdash t \text{ EQ } u$   
 $\langle \text{proof} \rangle$

**lemma** *ground-subst*:  
 $\text{ground-aux } tm \text{ (insert (atom } i) S) \implies \text{ground } t \implies \text{ground-aux (subst } i \ t \ tm) S$   
 $\langle \text{proof} \rangle$

**lemma** *ground-subst-fm*:  
 $\text{ground-fm-aux } A \text{ (insert (atom } i) S) \implies \text{ground } t \implies \text{ground-fm-aux (A(i::=t)) } S$   
 $\langle \text{proof} \rangle$

**lemma** *elts-imp-ground*:  $u \in \text{elts } t \implies \text{ground-aux } t \ S \implies \text{ground-aux } u \ S$   
 $\langle \text{proof} \rangle$

**lemma** *ground-se-fm-induction*:  
 $\text{ground-fm } \alpha \implies \text{size } \alpha < n \implies \text{se-fm } \alpha \implies \text{eval-fm } e \ \alpha \implies \{\} \vdash \alpha$   
 $\langle \text{proof} \rangle$

**lemma** *ss-imp-se-fm*:  $\text{ss-fm } A \implies \text{se-fm } A$   
 $\langle \text{proof} \rangle$

**lemma** *se-fm-imp-thm*:  $\llbracket \text{se-fm } A; \text{ground-fm } A; \text{eval-fm } e \ A \rrbracket \implies \{\} \vdash A$   
 $\langle \text{proof} \rangle$

Theorem 2.5

**theorem** *Sigma-fm-imp-thm*:  $\llbracket \text{Sigma-fm } A; \text{ground-fm } A; \text{eval-fm } e \ 0 \ A \rrbracket \implies \{\} \vdash A$   
 $\langle \text{proof} \rangle$

**end**

# Kapitel 5

## Predicates for Terms, Formulas and Substitution

```
theory Coding-Predicates
imports Coding Sigma
begin
```

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

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

### 5.1 Predicates for atomic terms

#### 5.1.1 Free Variables

```
definition is-Var :: hf  $\Rightarrow$  bool where is-Var  $x \equiv \text{Ord } x \wedge 0 \in x$ 
```

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

```
lemma VarP-eqvt [eqvt]:  $(p \cdot \text{VarP } x) = \text{VarP } (p \cdot x)$   
<proof>
```

```
lemma VarP-fresh-iff [simp]:  $a \# \text{VarP } x \longleftrightarrow a \# x$   
<proof>
```

```
lemma eval-fm-VarP [simp]:  $\text{eval-fm } e (\text{VarP } x) \longleftrightarrow \text{is-Var } \llbracket x \rrbracket e$   
<proof>
```

```
lemma VarP-sf [iff]:  $\text{Sigma-fm } (\text{VarP } x)$   
<proof>
```

```
lemma VarP-subst [simp]:  $(\text{VarP } x)(i ::= t) = \text{VarP } (\text{subst } i \ t \ x)$   
<proof>
```



**lemma** *VarP-cong*:  $H \vdash x \text{ EQ } x' \implies H \vdash \text{VarP } x \text{ IFF } \text{VarP } x'$   
 ⟨proof⟩

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

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

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

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

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

**lemma** *is-Var-imp-decode-Var*:  $\text{is-Var } x \implies x = \llbracket \ll \text{Var } (\text{decode-Var } x) \gg \rrbracket e$   
 ⟨proof⟩

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

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

### 5.1.2 De Bruijn Indexes

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

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

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

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

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

$\langle proof \rangle$

**lemma** *IndP-Q-Ind*:  $H \vdash OrdP\ x \implies H \vdash IndP\ (Q-Ind\ x)$   
 $\langle proof \rangle$

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

**lemma** *IndP-cong*:  $H \vdash x\ EQ\ x' \implies H \vdash IndP\ x\ IFF\ IndP\ x'$   
 $\langle proof \rangle$

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

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

### 5.1.3 Various syntactic lemmas

**lemma** *eval-Var-q*:  $\llbracket \langle Var\ i \rangle \rrbracket e = q-Var\ i$   
 $\langle proof \rangle$

**lemma** *is-Var-eval-Var* [*simp*]:  $is-Var\ \llbracket \langle Var\ i \rangle \rrbracket e$   
 $\langle proof \rangle$

## 5.2 The predicate *SeqCTermP*, for Terms and Constants

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

**nominal-function** *SeqCTermP* ::  $bool \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$   
**where**  $\llbracket atom\ l\ \# (s, k, sl, m, n, sm, sn); atom\ sl\ \# (s, m, n, sm, sn);$   
 $atom\ m\ \# (s, n, sm, sn); atom\ n\ \# (s, sm, sn);$   
 $atom\ sm\ \# (s, sn); atom\ sn\ \# (s) \rrbracket \implies$   
*SeqCTermP*  $vf\ s\ k\ t =$   
 $LstSeqP\ s\ k\ t\ AND$   
 $All2\ l\ (SUCC\ k)\ (Ex\ sl\ (HPair\ (Var\ l)\ (Var\ sl)\ IN\ s\ AND$   
 $(Var\ sl\ EQ\ Zero\ OR\ (if\ vf\ then\ VarP\ (Var\ sl)\ else\ Fls)\ OR$   
 $Ex\ m\ (Ex\ n\ (Ex\ sm\ (Ex\ sn\ (Var\ m\ IN\ Var\ l\ AND\ Var\ n\ IN\ Var\ l$   
 $AND$   
 $HPair\ (Var\ m)\ (Var\ sm)\ IN\ s\ AND\ HPair\ (Var\ n)\ (Var\ sn)$   
 $IN\ s\ AND$   
 $Var\ sl\ EQ\ Q-Eats\ (Var\ sm)\ (Var\ sn))))))$   
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)

*<proof>*

**lemma**

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

$a \# \text{SeqCTermP } vf \ s \ k \ t \longleftrightarrow a \# \ s \wedge a \# \ k \wedge a \# \ t$  (**is** *?thesis1*)

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

$eval\text{-}fm \ e \ (\text{SeqCTermP } vf \ s \ k \ t) \longleftrightarrow \text{SeqCTerm } vf \ \llbracket s \rrbracket e \ \llbracket k \rrbracket e \ \llbracket t \rrbracket e$  (**is** *?thesis2*)

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

$\text{Sigma}\text{-}fm \ (\text{SeqCTermP } vf \ s \ k \ t)$  (**is** *?thsf*)

**and** *SeqCTermP-imp-LstSeqP*:

$\{ \text{SeqCTermP } vf \ s \ k \ t \} \vdash \text{LstSeqP } s \ k \ t$  (**is** *?thlstseq*)

**and** *SeqCTermP-imp-OrdP* [*simp*]:

$\{ \text{SeqCTermP } vf \ s \ k \ t \} \vdash \text{OrdP } k$  (**is** *?thord*)

*<proof>*

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

$(\text{SeqCTermP } vf \ s \ k \ t)(j::=w) = \text{SeqCTermP } vf \ (\text{subst } j \ w \ s) \ (\text{subst } j \ w \ k)$   
 $(\text{subst } j \ w \ t)$

*<proof>*

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

**abbreviation** *SeqTerm* ::  $hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$

**where**  $\text{SeqTerm} \equiv \text{SeqCTerm } True$

**abbreviation** *SeqTermP* ::  $tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$

**where**  $\text{SeqTermP} \equiv \text{SeqCTermP } True$

**abbreviation** *SeqConst* ::  $hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$

**where**  $\text{SeqConst} \equiv \text{SeqCTerm } False$

**abbreviation** *SeqConstP* ::  $tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$

**where**  $\text{SeqConstP} \equiv \text{SeqCTermP } False$

**lemma** *SeqConst-imp-SeqTerm*:  $\text{SeqConst } s \ k \ x \Longrightarrow \text{SeqTerm } s \ k \ x$

*<proof>*

**lemma** *SeqConstP-imp-SeqTermP*:  $\{ \text{SeqConstP } s \ k \ t \} \vdash \text{SeqTermP } s \ k \ t$

*<proof>*

## 5.3 The predicates *TermP* and *ConstP*

### 5.3.1 Definition

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

**where**  $\text{CTerm } vf \ t \equiv (\exists s \ k. \text{SeqCTerm } vf \ s \ k \ t)$

**nominal-function** *CTermP* ::  $bool \Rightarrow tm \Rightarrow fm$

**where**  $\llbracket \text{atom } k \# (s,t); \text{atom } s \# t \rrbracket \implies$   
 $C\text{TermP } \text{vf } t = \text{Ex } s (\text{Ex } k (\text{SeqCTermP } \text{vf } (\text{Var } s) (\text{Var } k) t))$   
 $\langle \text{proof} \rangle$

**nominal-termination** (*eqvt*)  
 $\langle \text{proof} \rangle$

**lemma**

**shows**  $C\text{TermP-fresh-iff}$  [*simp*]:  $a \# C\text{TermP } \text{vf } t \longleftrightarrow a \# t$  (**is** *?thesis1*)  
**and**  $\text{eval-fm-CTermP}$  [*simp*]:  $\text{eval-fm } e (C\text{TermP } \text{vf } t) \longleftrightarrow C\text{Term } \text{vf } \llbracket t \rrbracket e$  (**is** *?thesis2*)  
**and**  $C\text{TermP-sf}$  [*iff*]:  $\text{Sigma-fm } (C\text{TermP } \text{vf } t)$  (**is** *?thsf*)  
 $\langle \text{proof} \rangle$

**lemma**  $C\text{TermP-subst}$  [*simp*]:  $(C\text{TermP } \text{vf } i)(j::=w) = C\text{TermP } \text{vf } (\text{subst } j \ w \ i)$   
 $\langle \text{proof} \rangle$

**abbreviation**  $\text{Term} :: \text{hf} \Rightarrow \text{bool}$   
**where**  $\text{Term} \equiv C\text{Term } \text{True}$

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

**abbreviation**  $\text{Const} :: \text{hf} \Rightarrow \text{bool}$   
**where**  $\text{Const} \equiv C\text{Term } \text{False}$

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

### 5.3.2 Correctness: It Corresponds to Quotations of Real Terms

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

**corollary**  $\text{Term-quot-tm}$  [*iff*]: **fixes**  $t :: \text{tm}$  **shows**  $\text{Term } \llbracket \langle t \rangle \rrbracket e$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{SeqCTerm-imp-wf-dbtm}$ :  
**assumes**  $\text{SeqCTerm } \text{vf } s \ k \ x$   
**shows**  $\exists t::\text{dbtm}. \text{wf-dbtm } t \wedge x = \llbracket \text{quot-dbtm } t \rrbracket e$   
 $\langle \text{proof} \rangle$

**corollary**  $\text{Term-imp-wf-dbtm}$ :  
**assumes**  $\text{Term } x$  **obtains**  $t$  **where**  $\text{wf-dbtm } t \ x = \llbracket \text{quot-dbtm } t \rrbracket e$   
 $\langle \text{proof} \rangle$

**corollary**  $\text{Term-imp-is-tm}$ : **assumes**  $\text{Term } x$  **obtains**  $t::\text{tm}$  **where**  $x = \llbracket \langle t \rangle \rrbracket e$   
 $\langle \text{proof} \rangle$

**lemma** *Term-Var*: *Term* (*q-Var* *i*)  
 ⟨*proof*⟩

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

### 5.3.3 Correctness properties for constants

**lemma** *Const-imp-Term*: *Const* *x*  $\implies$  *Term* *x*  
 ⟨*proof*⟩

**lemma** *Const-0*: *Const* 0  
 ⟨*proof*⟩

**lemma** *ConstP-imp-TermP*:  $\{ConstP\ t\} \vdash TermP\ t$   
 ⟨*proof*⟩

## 5.4 Abstraction over terms

**definition** *SeqStTerm* :: *hf*  $\Rightarrow$  *hf*  $\Rightarrow$  *hf*  $\Rightarrow$  *hf*  $\Rightarrow$  *hf*  $\Rightarrow$  *hf*  $\Rightarrow$  *bool*  
**where** *SeqStTerm* *v* *u* *x* *x'* *s* *k*  $\equiv$   
 (*is-Var* *v*  $\wedge$  *BuildSeq2* ( $\lambda y\ y'$ . (*is-Ind* *y*  $\vee$  *Ord* *y*)  $\wedge$  *y'* = (*if* *y=v* *then* *u* *else* *y*))  
 ( $\lambda u\ u'\ v\ v'\ w\ w'$ . *u* = *q-Eats* *v* *w*  $\wedge$  *u'* = *q-Eats* *v'* *w'*) *s* *k* *x* *x'*)

**definition** *AbstTerm* :: *hf*  $\Rightarrow$  *hf*  $\Rightarrow$  *hf*  $\Rightarrow$  *hf*  $\Rightarrow$  *bool*  
**where** *AbstTerm* *v* *i* *x* *x'*  $\equiv$  *Ord* *i*  $\wedge$  ( $\exists s\ k$ . *SeqStTerm* *v* (*q-Ind* *i*) *x* *x'* *s* *k*)

### 5.4.1 Defining the syntax: quantified body

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

$\langle \text{proof} \rangle$

**nominal-termination** (*eqvt*)

$\langle \text{proof} \rangle$

**lemma**

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

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

(**is** *?thesis1*)

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

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

(**is** *?thesis2*)

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

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

**and** *SeqStTermP-imp-OrdP*:

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

**and** *SeqStTermP-imp-VarP*:

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

**and** *SeqStTermP-imp-LstSeqP*:

$$\{ \text{SeqStTermP } v \ i \ t \ u \ s \ k \} \vdash \text{LstSeqP } s \ k \ (\text{HPair } t \ u) \ (\text{is } ?thlstseq)$$

$\langle \text{proof} \rangle$

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

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

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

$\langle \text{proof} \rangle$

**lemma** *SeqStTermP-cong*:

$$\llbracket H \vdash t \text{ EQ } t'; H \vdash u \text{ EQ } u'; H \vdash s \text{ EQ } s'; H \vdash k \text{ EQ } k' \rrbracket$$

$$\implies H \vdash \text{SeqStTermP } v \ i \ t \ u \ s \ k \text{ IFF } \text{SeqStTermP } v \ i \ t' \ u' \ s' \ k'$$

$\langle \text{proof} \rangle$

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

## 5.4.2 Defining the syntax: main predicate

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

**where**  $\llbracket \text{atom } s \# (v, i, t, u, k); \text{atom } k \# (v, i, t, u) \rrbracket \implies$

$$\text{AbstTermP } v \ i \ t \ u =$$

$$\text{OrdP } i \ \text{AND } \text{Ex } s \ (\text{Ex } k \ (\text{SeqStTermP } v \ (Q\text{-Ind } i) \ t \ u \ (\text{Var } s) \ (\text{Var } k)))$$

$\langle \text{proof} \rangle$

**nominal-termination** (*eqvt*)

$\langle \text{proof} \rangle$

**lemma**

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

$$a \# \text{AbstTermP } v \ i \ t \ u \longleftrightarrow a \# v \wedge a \# i \wedge a \# t \wedge a \# u \ (\text{is } ?thesis1)$$

**and** *eval-fm-AbstTermP* [*simp*]:  
 $eval\text{-}fm\ e\ (AbstTermP\ v\ i\ t\ u) \longleftrightarrow AbstTerm\ \llbracket v \rrbracket e\ \llbracket i \rrbracket e\ \llbracket t \rrbracket e\ \llbracket u \rrbracket e$  (**is** *?thesis2*)  
**and** *AbstTermP-sf* [*iff*]:  
 $Sigma\text{-}fm\ (AbstTermP\ v\ i\ t\ u)$  (**is** *?thsf*)  
 $\langle proof \rangle$

**lemma** *AbstTermP-subst* [*simp*]:  
 $(AbstTermP\ v\ i\ t\ u)(j::=w) = AbstTermP\ (subst\ j\ w\ v)\ (subst\ j\ w\ i)\ (subst\ j\ w\ t)\ (subst\ j\ w\ u)$   
 $\langle proof \rangle$

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

### 5.4.3 Correctness: It Coincides with Abstraction over real terms

**lemma** *not-is-Var-is-Ind*:  $is\text{-}Var\ v \implies \neg is\text{-}Ind\ v$   
 $\langle proof \rangle$

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

**lemma** *AbstTerm-abst-dbtm*:  
 $AbstTerm\ (q\text{-}Var\ i)\ (ord\text{-}of\ n)\ \llbracket quot\text{-}dbtm\ t \rrbracket e$   
 $\llbracket quot\text{-}dbtm\ (abst\text{-}dbtm\ i\ n\ t) \rrbracket e$   
 $\langle proof \rangle$

## 5.5 Substitution over terms

**definition** *SubstTerm* ::  $hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$   
**where**  $SubstTerm\ v\ u\ x\ x' \equiv Term\ u \wedge (\exists s\ k. SeqStTerm\ v\ u\ x\ x'\ s\ k)$

### 5.5.1 Defining the syntax

**nominal-function** *SubstTermP* ::  $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$   
**where**  $\llbracket atom\ s\ \#\ (v,i,t,u,k); atom\ k\ \#\ (v,i,t,u) \rrbracket \implies$   
 $SubstTermP\ v\ i\ t\ u = TermP\ i\ AND\ Ex\ s\ (Ex\ k\ (SeqStTermP\ v\ i\ t\ u\ (Var\ s)\ (Var\ k)))$   
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)  
 $\langle proof \rangle$

**lemma**  
**shows** *SubstTermP-fresh-iff* [*simp*]:  
 $a\ \#\ SubstTermP\ v\ i\ t\ u \longleftrightarrow a\ \#\ v \wedge a\ \#\ i \wedge a\ \#\ t \wedge a\ \#\ u$  (**is** *?thesis1*)

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

**lemma** *SubstTermP-subst* [*simp*]:  
 $(SubstTermP\ v\ i\ t\ u)(j::=w) = SubstTermP\ (subst\ j\ w\ v)\ (subst\ j\ w\ i)\ (subst\ j\ w\ t)\ (subst\ j\ w\ u)$   
*<proof>*

**lemma** *SubstTermP-cong*:  
 $\llbracket H \vdash v\ EQ\ v'; H \vdash i\ EQ\ i'; H \vdash t\ EQ\ t'; H \vdash u\ EQ\ u' \rrbracket$   
 $\implies H \vdash SubstTermP\ v\ i\ t\ u\ IFF\ SubstTermP\ v'\ i'\ t'\ u'$   
*<proof>*

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

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

**corollary** *SubstTerm-imp-subst-dbtm'*:  
**assumes** *SubstTerm*  $v\ y\ x\ x'$   
**obtains**  $t::dbtm$  **and**  $u::dbtm$   
**where**  $y = \llbracket quot\text{-}dbtm\ u \rrbracket e$   
 $x = \llbracket quot\text{-}dbtm\ t \rrbracket e$   
 $x' = \llbracket quot\text{-}dbtm\ (subst\text{-}dbtm\ u\ (decode\text{-}Var\ v)\ t) \rrbracket e$   
*<proof>*

**lemma** *SubstTerm-subst-dbtm*:  
**assumes** *Term*  $\llbracket quot\text{-}dbtm\ u \rrbracket e$   
**shows** *SubstTerm*  $(q\text{-}Var\ v)\ \llbracket quot\text{-}dbtm\ u \rrbracket e\ \llbracket quot\text{-}dbtm\ t \rrbracket e\ \llbracket quot\text{-}dbtm\ (subst\text{-}dbtm\ u\ v\ t) \rrbracket e$   
*<proof>*

## 5.6 Abstraction over formulas

### 5.6.1 The predicate *AbstAtomicP*

**definition** *AbstAtomic*  $:: hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$   
**where** *AbstAtomic*  $v\ i\ y\ y' \equiv$



$(\exists t u t' u'. \text{AbstTerm } v i t t' \wedge \text{AbstTerm } v i u u' \wedge$   
 $((y = q\text{-Eq } t u \wedge y' = q\text{-Eq } t' u') \vee (y = q\text{-Mem } t u \wedge y' = q\text{-Mem } t'$   
 $u'))))$

**nominal-function**  $\text{AbstAtomicP} :: tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$

**where**  $[\text{atom } t \# (v, i, y, y', t', u, u'); \text{atom } t' \# (v, i, y, y', u, u');$

$\text{atom } u \# (v, i, y, y', u'); \text{atom } u' \# (v, i, y, y')]] \Longrightarrow$

$\text{AbstAtomicP } v i y y' =$

$\text{Ex } t (\text{Ex } u (\text{Ex } t' (\text{Ex } u'$

$(\text{AbstTermP } v i (\text{Var } t) (\text{Var } t') \text{ AND } \text{AbstTermP } v i (\text{Var } u) (\text{Var } u')$

$\text{AND}$

$((y \text{ EQ } Q\text{-Eq } (\text{Var } t) (\text{Var } u) \text{ AND } y' \text{ EQ } Q\text{-Eq } (\text{Var } t') (\text{Var}$

$u')) \text{ OR}$

$(y \text{ EQ } Q\text{-Mem } (\text{Var } t) (\text{Var } u) \text{ AND } y' \text{ EQ } Q\text{-Mem } (\text{Var } t')$

$(\text{Var } u')))))))$

$\langle \text{proof} \rangle$

**nominal-termination** ( $\text{eqvt}$ )

$\langle \text{proof} \rangle$

**lemma**

**shows**  $\text{AbstAtomicP-fresh-iff}$  [ $\text{simp}$ ]:

$a \# \text{AbstAtomicP } v i y y' \longleftrightarrow a \# v \wedge a \# i \wedge a \# y \wedge a \# y'$  (**is**

$?thesis1$ )

**and**  $\text{eval-fm-AbstAtomicP}$  [ $\text{simp}$ ]:

$\text{eval-fm } e (\text{AbstAtomicP } v i y y') \longleftrightarrow \text{AbstAtomic } [v]e [i]e [y]e [y']e$  (**is**

$?thesis2$ )

**and**  $\text{AbstAtomicP-sf}$  [ $\text{iff}$ ]:  $\text{Sigma-fm } (\text{AbstAtomicP } v i y y')$  (**is**  $?thsf$ )

$\langle \text{proof} \rangle$

**lemma**  $\text{AbstAtomicP-subst}$  [ $\text{simp}$ ]:

$(\text{AbstAtomicP } v tm y y')(i ::= w) = \text{AbstAtomicP } (\text{subst } i w v) (\text{subst } i w tm)$

$(\text{subst } i w y) (\text{subst } i w y')$

$\langle \text{proof} \rangle$

**declare**  $\text{AbstAtomicP.simps}$  [ $\text{simp del}$ ]

## 5.6.2 The predicate $\text{AbsMakeForm}$

**definition**  $\text{AbstMakeForm} :: hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf$   
 $\Rightarrow \text{bool}$

**where**  $\text{AbstMakeForm } k y y' i u u' j w w' \equiv$

$\text{Ord } k \wedge$

$((k = i \wedge k = j \wedge y = q\text{-Disj } u w \wedge y' = q\text{-Disj } u' w') \vee$

$(k = i \wedge y = q\text{-Neg } u \wedge y' = q\text{-Neg } u') \vee$

$(\text{succ } k = i \wedge y = q\text{-Ex } u \wedge y' = q\text{-Ex } u'))$

**definition**  $\text{SeqAbstForm} :: hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow \text{bool}$

**where**  $\text{SeqAbstForm } v i x x' s k \equiv$

*BuildSeq3 (AbstAtomic v) AbstMakeForm s k i x x'*

**nominal-function** *SeqAbstFormP* :: *tm*  $\Rightarrow$  *tm*  $\Rightarrow$  *tm*  $\Rightarrow$  *tm*  $\Rightarrow$  *tm*  $\Rightarrow$  *tm*  $\Rightarrow$  *fm*  
**where**  $\llbracket$  *atom l*  $\#$  (*s,k,v,sli,sl,sl',m,n,smi,sm,sm',sni,sn,sn'*);  
*atom sli*  $\#$  (*s,v,sl,sl',m,n,smi,sm,sm',sni,sn,sn'*);  
*atom sl*  $\#$  (*s,v,sl',m,n,smi,sm,sm',sni,sn,sn'*);  
*atom sl'*  $\#$  (*s,v,m,n,smi,sm,sm',sni,sn,sn'*);  
*atom m*  $\#$  (*s,n,smi,sm,sm',sni,sn,sn'*);  
*atom n*  $\#$  (*s,smi,sm,sm',sni,sn,sn'*); *atom smi*  $\#$  (*s,sm,sm',sni,sn,sn'*);  
*atom sm*  $\#$  (*s,sm',sni,sn,sn'*); *atom sm'*  $\#$  (*s,sni,sn,sn'*);  
*atom sni*  $\#$  (*s,sn,sn'*); *atom sn*  $\#$  (*s,sn'*); *atom sn'*  $\#$  (*s*)  $\rrbracket \implies$   
*SeqAbstFormP v i x x' s k* =  
*LstSeqP s k (HPair i (HPair x x')) AND*  
*All2 l (SUCC k) (Ex sli (Ex sl (Ex sl' (HPair (Var l) (HPair (Var sli) (HPair*  
*(Var sl) (Var sl')) IN s AND*  
*(AbstAtomicP v (Var sli) (Var sl) (Var sl')) OR*  
*OrdP (Var sli) AND*  
*Ex m (Ex n (Ex smi (Ex sm (Ex sm' (Ex sni (Ex sn (Ex sn'*  
*(Var m IN Var l AND Var n IN Var l AND*  
*HPair (Var m) (HPair (Var smi) (HPair (Var sm) (Var sm'))*  
*IN s AND*  
*HPair (Var n) (HPair (Var sni) (HPair (Var sn) (Var sn'))*  
*IN s AND*  
*((Var sli EQ Var smi AND Var sli EQ Var sni AND*  
*Var sl EQ Q-Disj (Var sm) (Var sn) AND*  
*Var sl' EQ Q-Disj (Var sm') (Var sn')) OR*  
*(Var sli EQ Var smi AND*  
*Var sl EQ Q-Neg (Var sm) AND Var sl' EQ Q-Neg (Var sm'))*  
*OR*  
*(SUCC (Var sli) EQ Var smi AND*  
*Var sl EQ Q-Ex (Var sm) AND Var sl' EQ Q-Ex (Var*  
*sm'))))))))))))))))*  
*<proof>*

**nominal-termination** (*eqvt*)  
*<proof>*

**lemma**

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

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

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

*eval-fm e (SeqAbstFormP v i x x' s k)*  $\longleftrightarrow$  *SeqAbstForm*  $\llbracket$  *v*  $\rrbracket$  *e*  $\llbracket$  *i*  $\rrbracket$  *e*  $\llbracket$  *x*  $\rrbracket$  *e*  $\llbracket$  *x'*  $\rrbracket$  *e*  
 $\llbracket$  *s*  $\rrbracket$  *e*  $\llbracket$  *k*  $\rrbracket$  *e* (**is** *?thesis2*)

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

*Sigma-fm (SeqAbstFormP v i x x' s k)* (**is** *?thsf*)

*<proof>*

**lemma** *SeqAbstFormP-subst* [simp]:  
 $(SeqAbstFormP\ v\ u\ x\ x'\ s\ k)(i::=t) =$   
 $SeqAbstFormP\ (subst\ i\ t\ v)\ (subst\ i\ t\ u)\ (subst\ i\ t\ x)\ (subst\ i\ t\ x')\ (subst\ i\ t\ s)\ (subst\ i\ t\ k)$   
 ⟨proof⟩

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

### 5.6.3 Defining the syntax: the main AbstForm predicate

**definition** *AbstForm* ::  $hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$   
**where**  $AbstForm\ v\ i\ x\ x' \equiv is-Var\ v \wedge Ord\ i \wedge (\exists s\ k.\ SeqAbstForm\ v\ i\ x\ x'\ s\ k)$

**nominal-function** *AbstFormP* ::  $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$   
**where**  $\llbracket atom\ s\ \#\ (v,i,x,x',k);$   
 $atom\ k\ \#\ (v,i,x,x') \rrbracket \Longrightarrow$   
 $AbstFormP\ v\ i\ x\ x' = VarP\ v\ AND\ OrdP\ i\ AND\ Ex\ s\ (Ex\ k\ (SeqAbstFormP\ v$   
 $i\ x\ x'\ (Var\ s)\ (Var\ k)))$   
 ⟨proof⟩

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

**lemma**  
**shows** *AbstFormP-fresh-iff* [simp]:  
 $a\ \#\ AbstFormP\ v\ i\ x\ x' \longleftrightarrow a\ \#\ v \wedge a\ \#\ i \wedge a\ \#\ x \wedge a\ \#\ x' \text{ (is ?thesis1)}$   
**and** *eval-fm-AbstFormP* [simp]:  
 $eval-fm\ e\ (AbstFormP\ v\ i\ x\ x') \longleftrightarrow AbstForm\ \llbracket v \rrbracket e\ \llbracket i \rrbracket e\ \llbracket x \rrbracket e\ \llbracket x' \rrbracket e \text{ (is$   
*?thesis2)*  
**and** *AbstFormP-sf* [iff]:  
 $Sigma-fm\ (AbstFormP\ v\ i\ x\ x') \text{ (is ?thsf)}$   
 ⟨proof⟩

**lemma** *AbstFormP-subst* [simp]:  
 $(AbstFormP\ v\ i\ x\ x')(j::=t) = AbstFormP\ (subst\ j\ t\ v)\ (subst\ j\ t\ i)\ (subst\ j\ t\ x)\ (subst\ j\ t\ x')$   
 ⟨proof⟩

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

### 5.6.4 Correctness: It Coincides with Abstraction over real Formulas

**lemma** *AbstForm-imp-Ord*:  $AbstForm\ v\ u\ x\ x' \Longrightarrow Ord\ v$   
 ⟨proof⟩

**lemma** *AbstForm-imp-abst-dbfm*:  
**assumes**  $AbstForm\ v\ i\ x\ x'$   
**shows**  $\exists A.\ x = \llbracket quot-dbfm\ A \rrbracket e \wedge$

$x' = \llbracket \text{quot-dbfm} (\text{abst-dbfm} (\text{decode-Var } v) (\text{nat-of-ord } i) A) \rrbracket e$   
 ⟨proof⟩

**lemma** *AbstForm-abst-dbfm*:

$\text{AbstForm } (q\text{-Var } i) (\text{ord-of } n) \llbracket \text{quot-dbfm } fm \rrbracket e \llbracket \text{quot-dbfm} (\text{abst-dbfm } i \ n \ fm) \rrbracket e$   
 ⟨proof⟩

## 5.7 Substitution over formulas

### 5.7.1 The predicate *SubstAtomicP*

**definition** *SubstAtomic* ::  $hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow \text{bool}$

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

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

**where**  $\llbracket \text{atom } t \ \sharp (v, tm, y, y', t', u, u');$   
 $\text{atom } t' \ \sharp (v, tm, y, y', u, u');$   
 $\text{atom } u \ \sharp (v, tm, y, y', u');$   
 $\text{atom } u' \ \sharp (v, tm, y, y') \rrbracket \implies$   
 $\text{SubstAtomicP } v \ tm \ y \ y' =$   
 $\text{Ex } t \ (\text{Ex } u \ (\text{Ex } t' \ (\text{Ex } u' \ (\text{SubstTermP } v \ tm \ (\text{Var } t) \ (\text{Var } t') \ \text{AND } \text{SubstTermP } v \ tm \ (\text{Var } u) \ (\text{Var } u') \ \text{AND}$   
 $((y \ \text{EQ } Q\text{-Eq } (\text{Var } t) \ (\text{Var } u) \ \text{AND } y' \ \text{EQ } Q\text{-Eq } (\text{Var } t') \ (\text{Var } u')) \ \text{OR}$   
 $(y \ \text{EQ } Q\text{-Mem } (\text{Var } t) \ (\text{Var } u) \ \text{AND } y' \ \text{EQ } Q\text{-Mem } (\text{Var } t') \ (\text{Var } u'))))))))$   
 ⟨proof⟩

**nominal-termination** (*eqvt*)

⟨proof⟩

**lemma**

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

$a \ \sharp \text{SubstAtomicP } v \ tm \ y \ y' \longleftrightarrow a \ \sharp v \wedge a \ \sharp tm \wedge a \ \sharp y \wedge a \ \sharp y'$  (is  
 ?thesis1)

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

$\text{eval-fm } e \ (\text{SubstAtomicP } v \ tm \ y \ y') \longleftrightarrow \text{SubstAtomic} \llbracket v \rrbracket e \llbracket tm \rrbracket e \llbracket y \rrbracket e \llbracket y' \rrbracket e$   
 (is ?thesis2)

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

?thsf)

⟨proof⟩

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

$(\text{SubstAtomicP } v \ tm \ y \ y')(i ::= w) = \text{SubstAtomicP} (\text{subst } i \ w \ v) (\text{subst } i \ w \ tm)$   
 $(\text{subst } i \ w \ y) (\text{subst } i \ w \ y')$

$\langle \text{proof} \rangle$

**lemma** *SubstAtomicP-cong*:

$\llbracket H \vdash v \text{ EQ } v'; H \vdash tm \text{ EQ } tm'; H \vdash x \text{ EQ } x'; H \vdash y \text{ EQ } y' \rrbracket$   
 $\implies H \vdash \text{SubstAtomicP } v \text{ tm } x \text{ y IFF } \text{SubstAtomicP } v' \text{ tm}' x' \text{ y}'$   
 $\langle \text{proof} \rangle$

### 5.7.2 The predicate *SubstMakeForm*

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

**where** *SubstMakeForm*  $y \text{ y}' u \text{ u}' w \text{ w}' \equiv$   
 $((y = q\text{-Disj } u \text{ w} \wedge y' = q\text{-Disj } u' \text{ w}') \vee$   
 $(y = q\text{-Neg } u \wedge y' = q\text{-Neg } u') \vee$   
 $(y = q\text{-Ex } u \wedge y' = q\text{-Ex } u'))$

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

**where** *SeqSubstForm*  $v \text{ u } x \text{ x}' s \text{ k} \equiv \text{BuildSeq2 } (\text{SubstAtomic } v \text{ u}) \text{SubstMakeForm}$   
 $s \text{ k } x \text{ x}'$

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

**where**  $\llbracket \text{atom } l \# (s, k, v, u, sl, sl', m, n, sm, sm', sn, sn');$   
 $\text{atom } sl \# (s, v, u, sl', m, n, sm, sm', sn, sn');$   
 $\text{atom } sl' \# (s, v, u, m, n, sm, sm', sn, sn');$   
 $\text{atom } m \# (s, n, sm, sm', sn, sn'); \text{atom } n \# (s, sm, sm', sn, sn');$   
 $\text{atom } sm \# (s, sm', sn, sn'); \text{atom } sm' \# (s, sn, sn');$   
 $\text{atom } sn \# (s, sn'); \text{atom } sn' \# s \rrbracket \implies$

*SeqSubstFormP*  $v \text{ u } x \text{ x}' s \text{ k} =$

$\text{LstSeqP } s \text{ k } (\text{HPair } x \text{ x}') \text{ AND}$

$\text{All2 } l \text{ (SUCC } k) \text{ (Ex } sl \text{ (Ex } sl' \text{ (HPair } (\text{Var } l) \text{ (HPair } (\text{Var } sl) \text{ (Var } sl')) \text{ IN}$   
 $s \text{ AND}$

$(\text{SubstAtomicP } v \text{ u } (\text{Var } sl) \text{ (Var } sl')) \text{ OR}$

$\text{Ex } m \text{ (Ex } n \text{ (Ex } sm \text{ (Ex } sm' \text{ (Ex } sn \text{ (Ex } sn' \text{ (Var } m \text{ IN Var } l \text{ AND}$   
 $\text{Var } n \text{ IN Var } l \text{ AND}$

$\text{HPair } (\text{Var } m) \text{ (HPair } (\text{Var } sm) \text{ (Var } sm')) \text{ IN } s \text{ AND}$

$\text{HPair } (\text{Var } n) \text{ (HPair } (\text{Var } sn) \text{ (Var } sn')) \text{ IN } s \text{ AND}$

$((\text{Var } sl \text{ EQ } Q\text{-Disj } (\text{Var } sm) \text{ (Var } sn) \text{ AND}$

$\text{Var } sl' \text{ EQ } Q\text{-Disj } (\text{Var } sm') \text{ (Var } sn')) \text{ OR}$

$(\text{Var } sl \text{ EQ } Q\text{-Neg } (\text{Var } sm) \text{ AND Var } sl' \text{ EQ } Q\text{-Neg } (\text{Var } sm'))$

$\text{OR}$

$(\text{Var } sl \text{ EQ } Q\text{-Ex } (\text{Var } sm) \text{ AND Var } sl' \text{ EQ } Q\text{-Ex } (\text{Var}$

$sm'))))))))))))$

$\langle \text{proof} \rangle$

**nominal-termination** (*eqvt*)

$\langle \text{proof} \rangle$

**lemma**

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

$a \# \text{SeqSubstFormP } v \text{ u } x \text{ x}' s \text{ k} \iff a \# v \wedge a \# u \wedge a \# x \wedge a \# x' \wedge a \# s$

$\wedge a \# k$  (is ?thesis1)  
**and** eval-fm-SeqSubstFormP [simp]:  
 $eval\text{-}fm\ e\ (SeqSubstFormP\ v\ u\ x\ x'\ s\ k) \longleftrightarrow$   
 $SeqSubstForm\ \llbracket v \rrbracket e\ \llbracket u \rrbracket e\ \llbracket x \rrbracket e\ \llbracket x' \rrbracket e\ \llbracket s \rrbracket e\ \llbracket k \rrbracket e$  (is ?thesis2)  
**and** SeqSubstFormP-sf [iff]:  
 $Sigma\text{-}fm\ (SeqSubstFormP\ v\ u\ x\ x'\ s\ k)$  (is ?thsf)  
**and** SeqSubstFormP-imp-OrdP:  
 $\{ SeqSubstFormP\ v\ u\ x\ x'\ s\ k \} \vdash OrdP\ k$  (is ?thOrd)  
**and** SeqSubstFormP-imp-LstSeqP:  
 $\{ SeqSubstFormP\ v\ u\ x\ x'\ s\ k \} \vdash LstSeqP\ s\ k$  (HPair x x') (is ?thLstSeq)  
 <proof>

**lemma** SeqSubstFormP-subst [simp]:  
 $(SeqSubstFormP\ v\ u\ x\ x'\ s\ k)(i::=t) =$   
 $SeqSubstFormP\ (subst\ i\ t\ v)\ (subst\ i\ t\ u)\ (subst\ i\ t\ x)\ (subst\ i\ t\ x')\ (subst\ i\ t\ s)\ (subst\ i\ t\ k)$   
 <proof>

**lemma** SeqSubstFormP-cong:  
 $\llbracket H \vdash t\ EQ\ t'; H \vdash u\ EQ\ u'; H \vdash s\ EQ\ s'; H \vdash k\ EQ\ k' \rrbracket$   
 $\implies H \vdash SeqSubstFormP\ v\ i\ t\ u\ s\ k\ IFF\ SeqSubstFormP\ v\ i\ t'\ u'\ s'\ k'$   
 <proof>

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

### 5.7.3 Defining the syntax: the main SubstForm predicate

**definition** SubstForm :: hf  $\Rightarrow$  hf  $\Rightarrow$  hf  $\Rightarrow$  hf  $\Rightarrow$  bool  
**where** SubstForm v u x x'  $\equiv$  is-Var v  $\wedge$  Term u  $\wedge$  ( $\exists s\ k.$  SeqSubstForm v u x x' s k)

**nominal-function** SubstFormP :: tm  $\Rightarrow$  tm  $\Rightarrow$  tm  $\Rightarrow$  tm  $\Rightarrow$  fm  
**where**  $\llbracket atom\ s\ \# (v,i,x,x',k); atom\ k\ \# (v,i,x,x') \rrbracket \implies$   
 $SubstFormP\ v\ i\ x\ x' =$   
 $VarP\ v\ AND\ TermP\ i\ AND\ Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ x\ x'\ (Var\ s)\ (Var\ k)))$   
 <proof>

**nominal-termination** (eqvt)  
 <proof>

**lemma**  
**shows** SubstFormP-fresh-iff [simp]:  
 $a \# SubstFormP\ v\ i\ x\ x' \longleftrightarrow a \# v \wedge a \# i \wedge a \# x \wedge a \# x'$  (is ?thesis1)  
**and** eval-fm-SubstFormP [simp]:  
 $eval\text{-}fm\ e\ (SubstFormP\ v\ i\ x\ x') \longleftrightarrow SubstForm\ \llbracket v \rrbracket e\ \llbracket i \rrbracket e\ \llbracket x \rrbracket e\ \llbracket x' \rrbracket e$  (is ?thesis2)  
**and** SubstFormP-sf [iff]:  
 $Sigma\text{-}fm\ (SubstFormP\ v\ i\ x\ x')$  (is ?thsf)

$\langle proof \rangle$

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

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

$\langle proof \rangle$

**lemma** *SubstFormP-cong*:

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

$\langle proof \rangle$

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

$\langle proof \rangle$

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

## 5.7.4 Correctness of substitution over formulas

**lemma** *SubstForm-imp-subst-dbfm-lemma*:

**assumes** *SubstForm*  $v\ \llbracket quot\text{-}dbtm\ u \rrbracket e\ x\ x'$

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

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

$\langle proof \rangle$

**lemma** *SubstForm-imp-subst-dbfm*:

**assumes** *SubstForm*  $v\ u\ x\ x'$

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

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

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

$\langle proof \rangle$

**lemma** *SubstForm-subst-dbfm*:

**assumes**  $u: wf\text{-}dbtm\ u$

**shows** *SubstForm*  $(q\text{-}Var\ i)\ \llbracket quot\text{-}dbtm\ u \rrbracket e\ \llbracket quot\text{-}dbfm\ A \rrbracket e$

$\llbracket quot\text{-}dbfm\ (subst\text{-}dbfm\ u\ i\ A) \rrbracket e$

$\langle proof \rangle$

**corollary** *SubstForm-subst-dbfm-eq*:

$\llbracket v = q\text{-}Var\ i; Term\ ux; ux = \llbracket quot\text{-}dbtm\ u \rrbracket e; A' = subst\text{-}dbfm\ u\ i\ A \rrbracket$

$\implies SubstForm\ v\ ux\ \llbracket quot\text{-}dbfm\ A \rrbracket e\ \llbracket quot\text{-}dbfm\ A' \rrbracket e$

$\langle proof \rangle$

## 5.8 The predicate *AtomicP*

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

**where** *Atomic*  $y \equiv \exists t\ u.\ Term\ t \wedge Term\ u \wedge (y = q\text{-}Eq\ t\ u \vee y = q\text{-}Mem\ t\ u)$

**nominal-function**  $AtomicP :: tm \Rightarrow fm$   
**where**  $\llbracket atom\ t \ \sharp\ (u,y); atom\ u \ \sharp\ y \rrbracket \Longrightarrow$   
 $AtomicP\ y = Ex\ t\ (Ex\ u\ (TermP\ (Var\ t)\ AND\ TermP\ (Var\ u)\ AND$   
 $(y\ EQ\ Q-Eq\ (Var\ t)\ (Var\ u)\ OR$   
 $y\ EQ\ Q-Mem\ (Var\ t)\ (Var\ u))))$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt)$   
 $\langle proof \rangle$

**lemma**  
**shows**  $AtomicP\text{-fresh-iff}\ [simp]: a \ \sharp\ AtomicP\ y \longleftrightarrow a \ \sharp\ y$  **(is ?thesis1)**  
**and**  $eval\text{-fm-}AtomicP\ [simp]: eval\text{-fm}\ e\ (AtomicP\ y) \longleftrightarrow Atomic\llbracket y \rrbracket e$  **(is ?thesis2)**  
**and**  $AtomicP\text{-sf}\ [iff]: Sigma\text{-fm}\ (AtomicP\ y)$  **(is ?thsf)**  
 $\langle proof \rangle$

## 5.9 The predicate *MakeForm*

**definition**  $MakeForm :: hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$   
**where**  $MakeForm\ y\ u\ w \equiv$   
 $y = q\text{-Disj}\ u\ w \vee y = q\text{-Neg}\ u \vee$   
 $(\exists v\ u'. AbstForm\ v\ 0\ u\ u' \wedge y = q\text{-Ex}\ u')$

**nominal-function**  $MakeFormP :: tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$   
**where**  $\llbracket atom\ v \ \sharp\ (y,u,w,au); atom\ au \ \sharp\ (y,u,w) \rrbracket \Longrightarrow$   
 $MakeFormP\ y\ u\ w =$   
 $y\ EQ\ Q\text{-Disj}\ u\ w\ OR\ y\ EQ\ Q\text{-Neg}\ u\ OR$   
 $Ex\ v\ (Ex\ au\ (AbstFormP\ (Var\ v)\ Zero\ u\ (Var\ au)\ AND\ y\ EQ\ Q\text{-Ex}\ (Var$   
 $au))))$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt)$   
 $\langle proof \rangle$

**lemma**  
**shows**  $MakeFormP\text{-fresh-iff}\ [simp]:$   
 $a \ \sharp\ MakeFormP\ y\ u\ w \longleftrightarrow a \ \sharp\ y \wedge a \ \sharp\ u \wedge a \ \sharp\ w$  **(is ?thesis1)**  
**and**  $eval\text{-fm-}MakeFormP\ [simp]:$   
 $eval\text{-fm}\ e\ (MakeFormP\ y\ u\ w) \longleftrightarrow MakeForm\ \llbracket y \rrbracket e\ \llbracket u \rrbracket e\ \llbracket w \rrbracket e$  **(is ?thesis2)**  
**and**  $MakeFormP\text{-sf}\ [iff]:$   
 $Sigma\text{-fm}\ (MakeFormP\ y\ u\ w)$  **(is ?thsf)**  
 $\langle proof \rangle$

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



## 5.10 The predicate $SeqFormP$

**definition**  $SeqForm :: hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$   
**where**  $SeqForm\ s\ k\ y \equiv BuildSeq\ Atomic\ MakeForm\ s\ k\ y$

**nominal-function**  $SeqFormP :: tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$   
**where**  $\llbracket atom\ l\ \# (s,k,t,sl,m,n,sm,sn); atom\ sl\ \# (s,k,t,m,n,sm,sn);$   
 $atom\ m\ \# (s,k,t,n,sm,sn); atom\ n\ \# (s,k,t,sm,sn);$   
 $atom\ sm\ \# (s,k,t,sn); atom\ sn\ \# (s,k,t) \rrbracket \Longrightarrow$   
 $SeqFormP\ s\ k\ t =$   
 $LstSeqP\ s\ k\ t\ AND$   
 $All2\ n\ (SUCC\ k)\ (Ex\ sn\ (HPair\ (Var\ n)\ (Var\ sn)\ IN\ s\ AND\ (AtomicP\ (Var\ sn)\ OR$   
 $AND$   
 $AND$   
 $s\ AND$   
 $MakeFormP\ (Var\ sn)\ (Var\ sm)\ (Var\ sl))))))$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt)$   
 $\langle proof \rangle$

**lemma**  
**shows**  $SeqFormP\text{-fresh-iff}\ [simp]:$   
 $a\ \#\ SeqFormP\ s\ k\ t \longleftrightarrow a\ \#\ s \wedge a\ \#\ k \wedge a\ \#\ t\ (\mathbf{is}\ ?thesis1)$   
**and**  $eval\text{-fm}\text{-}SeqFormP\ [simp]:$   
 $eval\text{-fm}\ e\ (SeqFormP\ s\ k\ t) \longleftrightarrow SeqForm\ \llbracket s \rrbracket e\ \llbracket k \rrbracket e\ \llbracket t \rrbracket e\ (\mathbf{is}\ ?thesis2)$   
**and**  $SeqFormP\text{-sf}\ [iff]:\ Sigma\text{-fm}\ (SeqFormP\ s\ k\ t)\ (\mathbf{is}\ ?thsf)$   
 $\langle proof \rangle$

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

## 5.11 The predicate $FormP$

### 5.11.1 Definition

**definition**  $Form :: hf \Rightarrow bool$   
**where**  $Form\ y \equiv (\exists\ s\ k. SeqForm\ s\ k\ y)$

**nominal-function**  $FormP :: tm \Rightarrow fm$   
**where**  $\llbracket atom\ k\ \# (s,y); atom\ s\ \# y \rrbracket \Longrightarrow$   
 $FormP\ y = Ex\ k\ (Ex\ s\ (SeqFormP\ (Var\ s)\ (Var\ k)\ y))$   
 $\langle proof \rangle$

**nominal-termination**  $(eqvt)$   
 $\langle proof \rangle$

**lemma**

**shows** *FormP-fresh-iff* [simp]:  $a \# \text{FormP } y \longleftrightarrow a \# y$  (is ?thesis1)  
**and** *eval-fm-FormP* [simp]:  $\text{eval-fm } e (\text{FormP } y) \longleftrightarrow \text{Form } \llbracket y \rrbracket e$  (is ?thesis2)  
**and** *FormP-sf* [iff]:  $\text{Sigma-fm } (\text{FormP } y)$  (is ?thsf)  
(proof)

**lemma** *FormP-subst* [simp]:  $(\text{FormP } y)(j ::= w) = \text{FormP } (\text{subst } j \ w \ y)$   
(proof)

### 5.11.2 Correctness: It Corresponds to Quotations of Real Formulas

**lemma** *AbstForm-trans-fm*:

*AbstForm* (q-Var i) 0  $\llbracket \langle A \rangle \rrbracket e \llbracket \text{quot-dbfm } (\text{trans-fm } [i] \ A) \rrbracket e$   
(proof)

**corollary** *AbstForm-trans-fm-eq*:

$\llbracket x = \llbracket \langle A \rangle \rrbracket e; \ x' = \llbracket \text{quot-dbfm } (\text{trans-fm } [i] \ A) \rrbracket e \rrbracket \implies \text{AbstForm } (\text{q-Var } i) \ 0 \ x \ x'$   
(proof)

**lemma** *wf-Form-quot-dbfm* [simp]:

**assumes** *wf-dbfm* A **shows** *Form*  $\llbracket \text{quot-dbfm } A \rrbracket e$   
(proof)

**lemma** *Form-quot-fm* [iff]: **fixes** A :: fm **shows** *Form*  $\llbracket \langle A \rangle \rrbracket e$   
(proof)

**lemma** *Atomic-Form-is-wf-dbfm*: *Atomic* x  $\implies \exists A. \text{wf-dbfm } A \wedge x = \llbracket \text{quot-dbfm } A \rrbracket e$   
(proof)

**lemma** *SeqForm-imp-wf-dbfm*:

**assumes** *SeqForm* s k x  
**shows**  $\exists A. \text{wf-dbfm } A \wedge x = \llbracket \text{quot-dbfm } A \rrbracket e$   
(proof)

**lemma** *Form-imp-wf-dbfm*:

**assumes** *Form* x **obtains** A **where** *wf-dbfm* A x =  $\llbracket \text{quot-dbfm } A \rrbracket e$   
(proof)

**lemma** *Form-imp-is-fm*: **assumes** *Form* x **obtains** A::fm **where** x =  $\llbracket \langle A \rangle \rrbracket e$   
(proof)

**lemma** *SubstForm-imp-subst-fm*:

**assumes** *SubstForm* v  $\llbracket \langle u \rangle \rrbracket e \ x \ x' \ \text{Form } x$   
**obtains** A::fm **where** x =  $\llbracket \langle A \rangle \rrbracket e \ x' = \llbracket \langle A(\text{decode-Var } v ::= u) \rangle \rrbracket e$   
(proof)

**lemma** *SubstForm-unique:*

**assumes** *is-Var v and Term y and Form x*

**shows**  $\text{SubstForm } v \ y \ x \ x' \longleftrightarrow$

$(\exists t::tm. y = \llbracket \langle t \rangle \rrbracket e \wedge (\exists A::fm. x = \llbracket \langle A \rangle \rrbracket e \wedge x' = \llbracket \langle A(\text{decode-Var } v::=t) \rangle \rrbracket e))$

*<proof>*

**lemma** *SubstForm-quot-unique:*  $\text{SubstForm } (q\text{-Var } i) \llbracket \langle t \rangle \rrbracket e \llbracket \langle A \rangle \rrbracket e \ x' \longleftrightarrow x' = \llbracket \langle A(i::=t) \rangle \rrbracket e$

*<proof>*

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

*<proof>*

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

**definition**  $\text{VarNonOccForm} :: hf \Rightarrow hf \Rightarrow bool$

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

**nominal-function**  $\text{VarNonOccFormP} :: tm \Rightarrow tm \Rightarrow fm$

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

*<proof>*

**nominal-termination** (*eqvt*)

*<proof>*

**lemma**

**shows** *VarNonOccFormP-fresh-iff* [*simp*]:  $a \# \text{VarNonOccFormP } v \ y \longleftrightarrow a \# v \wedge a \# y$  (*is ?thesis1*)

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

$\text{eval-fm } e \ (\text{VarNonOccFormP } v \ y) \longleftrightarrow \text{VarNonOccForm } \llbracket v \rrbracket e \ \llbracket y \rrbracket e$  (*is ?thesis2*)

**and** *VarNonOccFormP-sf* [*iff*]:  $\text{Sigma-fm } (\text{VarNonOccFormP } v \ y)$  (*is ?thsf*)

*<proof>*

### 5.11.4 Correctness for Real Terms and Formulas

**lemma** *VarNonOccForm-imp-dbfm-fresh:*

**assumes**  $\text{VarNonOccForm } v \ x$

**shows**  $\exists A. \text{wf-dbfm } A \wedge x = \llbracket \text{quot-dbfm } A \rrbracket e \wedge \text{atom } (\text{decode-Var } v) \# A$

*<proof>*

**corollary** *VarNonOccForm-imp-fresh:*

**assumes**  $\text{VarNonOccForm } v \ x$  **obtains**  $A::fm$  **where**  $x = \llbracket \langle A \rangle \rrbracket e$   $\text{atom } (\text{decode-Var } v) \# A$

*<proof>*

**lemma** *VarNonOccForm-dbfm*:  
 $wf\text{-}dbfm\ A \implies atom\ i \# A \implies VarNonOccForm\ (q\text{-}Var\ i)\ [\text{quot}\text{-}dbfm\ A]e$   
 $\langle proof \rangle$

**corollary** *fresh-imp-VarNonOccForm*:  
**fixes**  $A::fm$  **shows**  $atom\ i \# A \implies VarNonOccForm\ (q\text{-}Var\ i)\ [\ll A \gg]e$   
 $\langle proof \rangle$

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

**end**

# Kapitel 6

## Formalizing Provability

```
theory Pf-Predicates
imports Coding-Predicates
begin
```

### 6.1 Section 4 Predicates (Leading up to Pf)

#### 6.1.1 The predicate *SentP*, for the Sentential (Boolean) Axioms

**definition** *Sent-axioms* ::  $hf \Rightarrow hf \Rightarrow hf \Rightarrow hf \Rightarrow bool$  where

```
Sent-axioms x y z w ≡
  x = q-Imp y y ∨
  x = q-Imp y (q-Disj y z) ∨
  x = q-Imp (q-Disj y y) y ∨
  x = q-Imp (q-Disj y (q-Disj z w)) (q-Disj (q-Disj y z) w) ∨
  x = q-Imp (q-Disj y z) (q-Imp (q-Disj (q-Neg y) w) (q-Disj z w))
```

**definition** *Sent* ::  $hf$  set where

```
Sent ≡ {x. ∃ y z w. Form y ∧ Form z ∧ Form w ∧ Sent-axioms x y z w}
```

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

```
where [[atom y ‡ (z,w,x); atom z ‡ (w,x); atom w ‡ x]] ==>
  SentP x = Ex y (Ex z (Ex w (FormP (Var y) AND FormP (Var z) AND
FormP (Var w) AND
  ( (x EQ Q-Imp (Var y) (Var y)) OR
    (x EQ Q-Imp (Var y) (Q-Disj (Var y) (Var z)) OR
    (x EQ Q-Imp (Q-Disj (Var y) (Var y)) (Var y)) OR
    (x EQ Q-Imp (Q-Disj (Var y) (Q-Disj (Var z) (Var w)))
      (Q-Disj (Q-Disj (Var y) (Var z)) (Var w))) OR
    (x EQ Q-Imp (Q-Disj (Var y) (Var z))
      (Q-Imp (Q-Disj (Q-Neg (Var y)) (Var w)) (Q-Disj (Var
z) (Var w))))))))))
  <proof>
```

**nominal-termination** (*eqvt*)

*<proof>*

**lemma**

**shows** *SentP-fresh-iff* [*simp*]:  $a \# \text{SentP } x \longleftrightarrow a \# x$  (is ?thesis1)  
**and** *eval-fm-SentP* [*simp*]:  $\text{eval-fm } e (\text{SentP } x) \longleftrightarrow \llbracket x \rrbracket e \in \text{Sent}$  (is ?thesis2)  
**and** *SentP-sf* [*iff*]:  $\text{Sigma-fm } (\text{SentP } x)$  (is ?thsf)  
*<proof>*

### 6.1.2 The predicate *Equality-axP*, for the Equality Axioms

**definition** *Equality-ax* :: *hf set where*

$\text{Equality-ax} \equiv \{ \llbracket \llcorner \text{refl-ax} \rceil \rrbracket e0, \llbracket \llcorner \text{eq-cong-ax} \rceil \rrbracket e0, \llbracket \llcorner \text{mem-cong-ax} \rceil \rrbracket e0, \llbracket \llcorner \text{eats-cong-ax} \rceil \rrbracket e0 \}$

**function** *Equality-axP* :: *tm*  $\Rightarrow$  *fm*

**where** *Equality-axP*  $x =$   
 $x \text{ EQ } \llcorner \text{refl-ax} \rceil \text{ OR } x \text{ EQ } \llcorner \text{eq-cong-ax} \rceil \text{ OR } x \text{ EQ } \llcorner \text{mem-cong-ax} \rceil \text{ OR } x \text{ EQ } \llcorner \text{eats-cong-ax} \rceil$   
*<proof>*

**termination**

*<proof>*

**lemma** *eval-fm-Equality-axP* [*simp*]:  $\text{eval-fm } e (\text{Equality-axP } x) \longleftrightarrow \llbracket x \rrbracket e \in \text{Equality-ax}$   
*<proof>*

### 6.1.3 The predicate *HF-axP*, for the HF Axioms

**definition** *HF-ax* :: *hf set where*

$\text{HF-ax} \equiv \{ \llbracket \llcorner \text{HF1} \rceil \rrbracket e0, \llbracket \llcorner \text{HF2} \rceil \rrbracket e0 \}$

**function** *HF-axP* :: *tm*  $\Rightarrow$  *fm*

**where** *HF-axP*  $x = x \text{ EQ } \llcorner \text{HF1} \rceil \text{ OR } x \text{ EQ } \llcorner \text{HF2} \rceil$   
*<proof>*

**termination**

*<proof>*

**lemma** *eval-fm-HF-axP* [*simp*]:  $\text{eval-fm } e (\text{HF-axP } x) \longleftrightarrow \llbracket x \rrbracket e \in \text{HF-ax}$   
*<proof>*

**lemma** *HF-axP-sf* [*iff*]:  $\text{Sigma-fm } (\text{HF-axP } t)$

*<proof>*

### 6.1.4 The specialisation axioms

**inductive-set** *Special-ax* :: *hf set where*

$I: \llbracket \text{AbstForm } v \ 0 \ x \ ax; \text{SubstForm } v \ y \ x \ sx; \text{Form } x; \text{is-Var } v; \text{Term } y \rrbracket$   
 $\implies q\text{-Imp } sx \ (q\text{-Ex } ax) \in \text{Special-ax}$

## Defining the syntax

**nominal-function** *Special-axP* :: *tm*  $\Rightarrow$  *fm* **where**

$\llbracket \text{atom } v \# (p, sx, y, ax, x); \text{atom } x \# (p, sx, y, ax);$   
 $\text{atom } ax \# (p, sx, y); \text{atom } y \# (p, sx); \text{atom } sx \# p \rrbracket \Longrightarrow$   
 $\text{Special-axP } p = \text{Ex } v (\text{Ex } x (\text{Ex } ax (\text{Ex } y (\text{Ex } sx$   
 $(\text{FormP } (\text{Var } x) \text{ AND } \text{VarP } (\text{Var } v) \text{ AND } \text{TermP } (\text{Var } y) \text{ AND}$   
 $\text{AbstFormP } (\text{Var } v) \text{ Zero } (\text{Var } x) (\text{Var } ax) \text{ AND}$   
 $\text{SubstFormP } (\text{Var } v) (\text{Var } y) (\text{Var } x) (\text{Var } sx) \text{ AND}$   
 $p \text{ EQ } Q\text{-Imp } (\text{Var } sx) (Q\text{-Ex } (\text{Var } ax))))))$   
 <proof>

**nominal-termination** (*eqvt*)

<proof>

**lemma**

**shows** *Special-axP-fresh-iff* [*simp*]:  $a \# \text{Special-axP } p \longleftrightarrow a \# p$  (**is** *?thesis1*)  
**and** *eval-fm-Special-axP* [*simp*]:  $\text{eval-fm } e (\text{Special-axP } p) \longleftrightarrow \llbracket p \rrbracket e \in \text{Special-ax}$   
**(is** *?thesis2*)  
**and** *Special-axP-sf* [*iff*]:  $\text{Sigma-fm } (\text{Special-axP } p)$  (**is** *?thesis3*)  
 <proof>

## Correctness (or, correspondence)

**lemma** *Special-ax-imp-special-axioms*:

**assumes**  $x \in \text{Special-ax}$  **shows**  $\exists A. x = \llbracket \langle A \rangle \rrbracket e \wedge A \in \text{special-axioms}$   
 <proof>

**lemma** *special-axioms-into-Special-ax*:  $A \in \text{special-axioms} \Longrightarrow \llbracket \langle A \rangle \rrbracket e \in \text{Special-ax}$   
 <proof>

We have precisely captured the codes of the specialisation axioms.

**corollary** *Special-ax-eq-special-axioms*:  $\text{Special-ax} = (\bigcup A \in \text{special-axioms}. \{ \llbracket \langle A \rangle \rrbracket e \})$   
 <proof>

### 6.1.5 The induction axioms

**inductive-set** *Induction-ax* :: *hf set* **where**

$I: \llbracket \text{SubstForm } v \ 0 \ x \ x0;$   
 $\text{SubstForm } v \ w \ x \ xw;$   
 $\text{SubstForm } v \ (q\text{-Eats } v \ w) \ x \ xevw;$   
 $\text{AbstForm } w \ 0 \ (q\text{-Imp } x \ (q\text{-Imp } xw \ xevw)) \ \text{allw};$   
 $\text{AbstForm } v \ 0 \ (q\text{-All } \text{allw}) \ \text{allvw};$   
 $\text{AbstForm } v \ 0 \ x \ ax;$   
 $v \neq w; \text{VarNonOccForm } w \ x \rrbracket$   
 $\Longrightarrow q\text{-Imp } x0 \ (q\text{-Imp } (q\text{-All } \text{allvw}) \ (q\text{-All } ax)) \in \text{Induction-ax}$

## Defining the syntax

**nominal-function** *Induction-axP* :: *tm*  $\Rightarrow$  *fm* **where**

$\llbracket \text{atom } ax \# (p, v, w, x, x0, xw, xevw, allw, allvw);$   
 $\text{atom } allvw \# (p, v, w, x, x0, xw, xevw, allw); \text{atom } allw \# (p, v, w, x, x0, xw, xevw);$   
 $\text{atom } xevw \# (p, v, w, x, x0, xw); \text{atom } xw \# (p, v, w, x, x0);$   
 $\text{atom } x0 \# (p, v, w, x); \text{atom } x \# (p, v, w);$   
 $\text{atom } w \# (p, v); \text{atom } v \# p \rrbracket \implies$   
 $\text{Induction-ax}P \ p = \text{Ex } v \ (\text{Ex } w \ (\text{Ex } x \ (\text{Ex } x0 \ (\text{Ex } xw \ (\text{Ex } xevw \ (\text{Ex } allw \ (\text{Ex } allvw$   
 $(\text{Ex } ax$   
 $((\text{Var } v \ \text{NEQ } \text{Var } w) \ \text{AND } \text{VarNonOccFormP } (\text{Var } w) \ (\text{Var } x) \ \text{AND}$   
 $\text{SubstFormP } (\text{Var } v) \ \text{Zero } (\text{Var } x) \ (\text{Var } x0) \ \text{AND}$   
 $\text{SubstFormP } (\text{Var } v) \ (\text{Var } w) \ (\text{Var } x) \ (\text{Var } xw) \ \text{AND}$   
 $\text{SubstFormP } (\text{Var } v) \ (\text{Q-Eats } (\text{Var } v) \ (\text{Var } w)) \ (\text{Var } x) \ (\text{Var } xevw)$   
 $\text{AND}$   
 $\text{AbstFormP } (\text{Var } w) \ \text{Zero } (\text{Q-Imp } (\text{Var } x) \ (\text{Q-Imp } (\text{Var } xw) \ (\text{Var}$   
 $xevw))) \ (\text{Var } allw) \ \text{AND}$   
 $\text{AbstFormP } (\text{Var } v) \ \text{Zero } (\text{Q-All } (\text{Var } allw)) \ (\text{Var } allvw) \ \text{AND}$   
 $\text{AbstFormP } (\text{Var } v) \ \text{Zero } (\text{Var } x) \ (\text{Var } ax) \ \text{AND}$   
 $p \ \text{EQ } \text{Q-Imp } (\text{Var } x0) \ (\text{Q-Imp } (\text{Q-All } (\text{Var } allvw)) \ (\text{Q-All } (\text{Var}$   
 $ax))))))))))$   
 $\langle \text{proof} \rangle$

**nominal-termination** (*eqvt*)  
 $\langle \text{proof} \rangle$

**lemma**

**shows** *Induction-axP-fresh-iff* [*simp*]:  $a \# \text{Induction-ax}P \ p \longleftrightarrow a \# p$  (**is** *?thesis1*)  
**and** *eval-fm-Induction-axP* [*simp*]:  
 $\text{eval-fm } e \ (\text{Induction-ax}P \ p) \longleftrightarrow \llbracket p \rrbracket e \in \text{Induction-ax}$  (**is** *?thesis2*)  
**and** *Induction-axP-sf* [*iff*]: *Sigma-fm* (*Induction-axP* *p*) (**is** *?thesis3*)  
 $\langle \text{proof} \rangle$

### Correctness (or, correspondence)

**lemma** *Induction-ax-imp-induction-axioms*:

**assumes**  $x \in \text{Induction-ax}$  **shows**  $\exists A. x = \llbracket \langle A \rangle \rrbracket e \wedge A \in \text{induction-axioms}$   
 $\langle \text{proof} \rangle$

**lemma** *induction-axioms-into-Induction-ax*:

$A \in \text{induction-axioms} \implies \llbracket \langle A \rangle \rrbracket e \in \text{Induction-ax}$   
 $\langle \text{proof} \rangle$

We have captured the codes of the induction axioms.

**corollary** *Induction-ax-eq-induction-axioms*:

$\text{Induction-ax} = (\bigcup A \in \text{induction-axioms}. \{\llbracket \langle A \rangle \rrbracket e\})$   
 $\langle \text{proof} \rangle$

### 6.1.6 The predicate *AxiomP*, for any Axioms

**definition** *Extra-ax* :: *hf set* **where**

$\text{Extra-ax} \equiv \{\llbracket \langle \text{extra-axiom} \rangle \rrbracket e0\}$



**definition** *Axiom* :: hf set **where**

$$Axiom \equiv Extra-ax \cup Sent \cup Equality-ax \cup HF-ax \cup Special-ax \cup Induction-ax$$

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

$$\text{where } AxiomP\ x \equiv x\ EQ \ll extra-axiom \gg\ OR\ SentP\ x\ OR\ Equality-axP\ x\ OR\ HF-axP\ x\ OR\ Special-axP\ x\ OR\ Induction-axP\ x$$

**lemma** *AxiomP-eqvt* [eqvt]:  $(p \cdot AxiomP\ x) = AxiomP\ (p \cdot x)$

*<proof>*

**lemma** *AxiomP-fresh-iff* [simp]:  $a \# AxiomP\ x \longleftrightarrow a \# x$

*<proof>*

**lemma** *eval-fm-AxiomP* [simp]:  $eval-fm\ e\ (AxiomP\ x) \longleftrightarrow \llbracket x \rrbracket e \in Axiom$

*<proof>*

**lemma** *AxiomP-sf* [iff]: *Sigma-fm* (*AxiomP* *t*)

*<proof>*

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

**definition** *ModPon* :: hf  $\Rightarrow$  hf  $\Rightarrow$  hf  $\Rightarrow$  bool **where**

$$ModPon\ x\ y\ z \equiv (y = q-Imp\ x\ z)$$

**definition** *ModPonP* :: tm  $\Rightarrow$  tm  $\Rightarrow$  tm  $\Rightarrow$  fm

$$\text{where } ModPonP\ x\ y\ z = (y\ EQ\ Q-Imp\ x\ z)$$

**lemma** *ModPonP-eqvt* [eqvt]:  $(p \cdot ModPonP\ x\ y\ z) = ModPonP\ (p \cdot x)\ (p \cdot y)\ (p \cdot z)$

*<proof>*

**lemma** *ModPonP-fresh-iff* [simp]:  $a \# ModPonP\ x\ y\ z \longleftrightarrow a \# x \wedge a \# y \wedge a \# z$

*<proof>*

**lemma** *eval-fm-ModPonP* [simp]:  $eval-fm\ e\ (ModPonP\ x\ y\ z) \longleftrightarrow ModPon\ \llbracket x \rrbracket e\ \llbracket y \rrbracket e\ \llbracket z \rrbracket e$

*<proof>*

**lemma** *ModPonP-sf* [iff]: *Sigma-fm* (*ModPonP* *t* *u* *v*)

*<proof>*

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

$$(ModPonP\ t\ u\ v)(i ::= w) = ModPonP\ (subst\ i\ w\ t)\ (subst\ i\ w\ u)\ (subst\ i\ w\ v)$$

*<proof>*

### 6.1.8 The predicate $ExistsP$ , for the existential rule

#### Definition

**definition**  $Exists :: hf \Rightarrow hf \Rightarrow bool$  **where**

$$Exists\ p\ q \equiv (\exists x\ x'\ y\ v.\ Form\ x \wedge VarNonOccForm\ v\ y \wedge AbstForm\ v\ 0\ x\ x' \wedge p = q\text{-}Imp\ x\ y \wedge q = q\text{-}Imp\ (q\text{-}Ex\ x')\ y)$$

**nominal-function**  $ExistsP :: tm \Rightarrow tm \Rightarrow fm$  **where**

$$\begin{aligned} & \llbracket atom\ x \# (p, q, v, y, x'); atom\ x' \# (p, q, v, y); \\ & \quad atom\ y \# (p, q, v); atom\ v \# (p, q) \rrbracket \Longrightarrow \\ & ExistsP\ p\ q = Ex\ x\ (Ex\ x'\ (Ex\ y\ (Ex\ v\ (FormP\ (Var\ x)\ AND \\ & \quad VarNonOccFormP\ (Var\ v)\ (Var\ y)\ AND \\ & \quad AbstFormP\ (Var\ v)\ Zero\ (Var\ x)\ (Var\ x')\ AND \\ & \quad p\ EQ\ Q\text{-}Imp\ (Var\ x)\ (Var\ y)\ AND \\ & \quad q\ EQ\ Q\text{-}Imp\ (Q\text{-}Ex\ (Var\ x'))\ (Var\ y)))))) \end{aligned}$$

$\langle proof \rangle$

**nominal-termination** ( $eqvt$ )

$\langle proof \rangle$

**lemma**

**shows**  $ExistsP\text{-fresh}\text{-iff}$  [ $simp$ ]:  $a \# ExistsP\ p\ q \longleftrightarrow a \# p \wedge a \# q$  (**is**  $?thesis1$ )

**and**  $eval\text{-}fm\text{-}ExistsP$  [ $simp$ ]:  $eval\text{-}fm\ e\ (ExistsP\ p\ q) \longleftrightarrow Exists\ \llbracket p \rrbracket e\ \llbracket q \rrbracket e$  (**is**  $?thesis2$ )

**and**  $ExistsP\text{-sf}$  [ $iff$ ]:  $Sigma\text{-}fm\ (ExistsP\ p\ q)$  (**is**  $?thesis3$ )

$\langle proof \rangle$

**lemma**  $ExistsP\text{-subst}$  [ $simp$ ]:  $(ExistsP\ p\ q)(j::=w) = ExistsP\ (subst\ j\ w\ p)\ (subst\ j\ w\ q)$

$\langle proof \rangle$

#### Correctness

**lemma**  $Exists\text{-imp}\text{-exists}$ :

**assumes**  $Exists\ p\ q$

**shows**  $\exists A\ B\ i.\ p = \llbracket \langle A\ IMP\ B \rangle \rrbracket e \wedge q = \llbracket \langle (Ex\ i\ A)\ IMP\ B \rangle \rrbracket e \wedge atom\ i \# B$

$\langle proof \rangle$

**lemma**  $Exists\text{-intro}$ :  $atom\ i \# B \Longrightarrow Exists\ (\llbracket \langle A\ IMP\ B \rangle \rrbracket e)\ \llbracket \langle (Ex\ i\ A)\ IMP\ B \rangle \rrbracket e$

$\langle proof \rangle$

Thus, we have precisely captured the codes of the specialisation axioms.

**corollary**  $Exists\text{-iff}\text{-exists}$ :

$Exists\ p\ q \longleftrightarrow (\exists A\ B\ i.\ p = \llbracket \langle A\ IMP\ B \rangle \rrbracket e \wedge q = \llbracket \langle (Ex\ i\ A)\ IMP\ B \rangle \rrbracket e \wedge atom\ i \# B)$

$\langle proof \rangle$

### 6.1.9 The predicate $SubstP$ , for the substitution rule

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

#### Definition

This is the inference  $H \vdash A \Longrightarrow H \vdash A (i ::= x)$

**definition**  $Subst :: hf \Rightarrow hf \Rightarrow bool$  **where**

$Subst\ p\ q \equiv (\exists v\ u.\ SubstForm\ v\ u\ p\ q)$

**nominal-function**  $SubstP :: tm \Rightarrow tm \Rightarrow fm$  **where**

$\llbracket atom\ u\ \# (p, q, v); atom\ v\ \# (p, q) \rrbracket \Longrightarrow$   
 $SubstP\ p\ q = Ex\ v (Ex\ u (SubstFormP (Var\ v) (Var\ u) p\ q))$   
 $\langle proof \rangle$

**nominal-termination** ( $eqvt$ )

$\langle proof \rangle$

**lemma**

**shows**  $SubstP\text{-fresh-iff}$  [ $simp$ ]:  $a\ \# SubstP\ p\ q \longleftrightarrow a\ \# p \wedge a\ \# q$  (**is**  $?thesis1$ )  
**and**  $eval\text{-fm-SubstP}$  [ $simp$ ]:  $eval\text{-fm}\ e (SubstP\ p\ q) \longleftrightarrow Subst\ \llbracket p \rrbracket e\ \llbracket q \rrbracket e$  (**is**  $?thesis2$ )  
**and**  $SubstP\text{-sf}$  [ $iff$ ]:  $Sigma\text{-fm}\ (SubstP\ p\ q)$  (**is**  $?thesis3$ )  
 $\langle proof \rangle$

**lemma**  $SubstP\text{-subst}$  [ $simp$ ]:  $(SubstP\ p\ q)(j ::= w) = SubstP (subst\ j\ w\ p) (subst\ j\ w\ q)$   
 $\langle proof \rangle$

#### Correctness

**lemma**  $Subst\text{-imp}\text{-subst}$ :

**assumes**  $Subst\ p\ q\ Form\ p$

**shows**  $\exists A :: fm.\ \exists i\ t.\ p = \llbracket \langle A \rangle \rrbracket e \wedge q = \llbracket \langle A(i ::= t) \rangle \rrbracket e$

$\langle proof \rangle$

### 6.1.10 The predicate $PrfP$

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

**where**  $Prf\ s\ k\ y \equiv BuildSeq (\lambda x.\ x \in Axiom) (\lambda u\ v\ w.\ ModPon\ v\ w\ u \vee Exists\ v\ u \vee Subst\ v\ u) s\ k\ y$

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

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

$PrfP\ s\ k\ t =$   
 $LstSeqP\ s\ k\ t\ AND$   
 $All2\ n\ (SUCC\ k)\ (Ex\ sn\ (HPair\ (Var\ n)\ (Var\ sn)\ IN\ s\ AND\ (AxiomP\ (Var\ sn)\ OR$   
 $Ex\ m\ (Ex\ l\ (Ex\ sm\ (Ex\ sl\ (Var\ m\ IN\ Var\ n\ AND\ Var\ l\ IN\ Var\ n$   
 $AND$   
 $HPair\ (Var\ m)\ (Var\ sm)\ IN\ s\ AND\ HPair\ (Var\ l)\ (Var\ sl)\ IN$   
 $s\ AND$   
 $(ModPonP\ (Var\ sm)\ (Var\ sl)\ (Var\ sn)\ OR$   
 $ExistsP\ (Var\ sm)\ (Var\ sn)\ OR$   
 $SubstP\ (Var\ sm)\ (Var\ sn))))))$   
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)  
 $\langle proof \rangle$

**lemma**

**shows**  $PrfP\text{-fresh-iff}$  [*simp*]:  $a\ \#\ PrfP\ s\ k\ t \longleftrightarrow a\ \#\ s \wedge a\ \#\ k \wedge a\ \#\ t$  (**is** *?thesis1*)  
**and**  $eval\text{-fm-PrfP}$  [*simp*]:  $eval\text{-fm}\ e\ (PrfP\ s\ k\ t) \longleftrightarrow Prf\ \llbracket s \rrbracket e\ \llbracket k \rrbracket e\ \llbracket t \rrbracket e$  (**is** *?thesis2*)  
**and**  $PrfP\text{-imp-OrdP}$  [*simp*]:  $\{PrfP\ s\ k\ t\} \vdash OrdP\ k$  (**is** *?thord*)  
**and**  $PrfP\text{-imp-LstSeqP}$  [*simp*]:  $\{PrfP\ s\ k\ t\} \vdash LstSeqP\ s\ k\ t$  (**is** *?thlstseq*)  
**and**  $PrfP\text{-sf}$  [*iff*]:  $Sigma\text{-fm}\ (PrfP\ s\ k\ t)$  (**is** *?thsf*)  
 $\langle proof \rangle$

**lemma**  $PrfP\text{-subst}$  [*simp*]:

$(PrfP\ t\ u\ v)(j::=w) = PrfP\ (subst\ j\ w\ t)\ (subst\ j\ w\ u)\ (subst\ j\ w\ v)$   
 $\langle proof \rangle$

### 6.1.11 The predicate $PfP$

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

**where**  $Pf\ y \equiv (\exists\ s\ k. Prf\ s\ k\ y)$

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

**where**  $\llbracket atom\ k\ \#\ (s,y); atom\ s\ \#\ y \rrbracket \Longrightarrow$

$PfP\ y = Ex\ k\ (Ex\ s\ (PrfP\ (Var\ s)\ (Var\ k)\ y))$

$\langle proof \rangle$

**nominal-termination** (*eqvt*)

$\langle proof \rangle$

**lemma**

**shows**  $PfP\text{-fresh-iff}$  [*simp*]:  $a\ \#\ PfP\ y \longleftrightarrow a\ \#\ y$  (**is** *?thesis1*)  
**and**  $eval\text{-fm-PfP}$  [*simp*]:  $eval\text{-fm}\ e\ (PfP\ y) \longleftrightarrow Pf\ \llbracket y \rrbracket e$  (**is** *?thesis2*)  
**and**  $PfP\text{-sf}$  [*iff*]:  $Sigma\text{-fm}\ (PfP\ y)$  (**is** *?thsf*)  
 $\langle proof \rangle$

**lemma** *PfP-subst [simp]*:  $(PfP\ t)(j::=w) = PfP\ (subst\ j\ w\ t)$   
 ⟨proof⟩

**lemma** *ground-PfP [simp]*:  $ground\text{-}fn\ (PfP\ y) = ground\ y$   
 ⟨proof⟩

## 6.2 Proposition 4.4

### 6.2.1 Left-to-Right Proof

**lemma** *extra-axiom-imp-Pf*:  $Pf\ [\llbracket extra\text{-}axiom \rrbracket]e$   
 ⟨proof⟩

**lemma** *boolean-axioms-imp-Pf*:  
**assumes**  $\alpha \in boolean\text{-}axioms$  **shows**  $Pf\ [\llbracket \alpha \rrbracket]e$   
 ⟨proof⟩

**lemma** *equality-axioms-imp-Pf*:  
**assumes**  $\alpha \in equality\text{-}axioms$  **shows**  $Pf\ [\llbracket \alpha \rrbracket]e$   
 ⟨proof⟩

**lemma** *HF-axioms-imp-Pf*:  
**assumes**  $\alpha \in HF\text{-}axioms$  **shows**  $Pf\ [\llbracket \alpha \rrbracket]e$   
 ⟨proof⟩

**lemma** *special-axioms-imp-Pf*:  
**assumes**  $\alpha \in special\text{-}axioms$  **shows**  $Pf\ [\llbracket \alpha \rrbracket]e$   
 ⟨proof⟩

**lemma** *induction-axioms-imp-Pf*:  
**assumes**  $\alpha \in induction\text{-}axioms$  **shows**  $Pf\ [\llbracket \alpha \rrbracket]e$   
 ⟨proof⟩

**lemma** *ModPon-imp-Pf*:  $\llbracket Pf\ [\llbracket Q\text{-}Imp\ x\ y \rrbracket]e; Pf\ [\llbracket x \rrbracket]e \rrbracket \Longrightarrow Pf\ [\llbracket y \rrbracket]e$   
 ⟨proof⟩

**lemma** *quot-ModPon-imp-Pf*:  $\llbracket Pf\ [\llbracket \alpha\ IMP\ \beta \rrbracket]e; Pf\ [\llbracket \alpha \rrbracket]e \rrbracket \Longrightarrow Pf\ [\llbracket \beta \rrbracket]e$   
 ⟨proof⟩

**lemma** *quot-Exists-imp-Pf*:  $\llbracket Pf\ [\llbracket \alpha\ IMP\ \beta \rrbracket]e; atom\ i\ \#\ \beta \rrbracket \Longrightarrow Pf\ [\llbracket Ex\ i\ \alpha\ IMP\ \beta \rrbracket]e$   
 ⟨proof⟩

**lemma** *proved-imp-Pf*: **assumes**  $H \vdash \alpha$   $H = \{\}$  **shows**  $Pf\ [\llbracket \alpha \rrbracket]e$   
 ⟨proof⟩

**corollary** *proved-imp-proved-PfP*:  $\{\} \vdash \alpha \Longrightarrow \{\} \vdash PfP\ \llbracket \alpha \rrbracket$   
 ⟨proof⟩

## 6.2.2 Right-to-Left Proof

**lemma** *Sent-imp-hfthm*:

**assumes**  $x \in \text{Sent}$  **shows**  $\exists A. x = \llbracket \langle A \rangle \rrbracket e \wedge \{\} \vdash A$   
*<proof>*

**lemma** *Extra-ax-imp-hfthm*:

**assumes**  $x \in \text{Extra-ax}$  **obtains**  $A$  **where**  $x = \llbracket \langle A \rangle \rrbracket e \wedge \{\} \vdash A$   
*<proof>*

**lemma** *Equality-ax-imp-hfthm*:

**assumes**  $x \in \text{Equality-ax}$  **obtains**  $A$  **where**  $x = \llbracket \langle A \rangle \rrbracket e \wedge \{\} \vdash A$   
*<proof>*

**lemma** *HF-ax-imp-hfthm*:

**assumes**  $x \in \text{HF-ax}$  **obtains**  $A$  **where**  $x = \llbracket \langle A \rangle \rrbracket e \wedge \{\} \vdash A$   
*<proof>*

**lemma** *Special-ax-imp-hfthm*:

**assumes**  $x \in \text{Special-ax}$  **obtains**  $A$  **where**  $x = \llbracket \langle A \rangle \rrbracket e \wedge \{\} \vdash A$   
*<proof>*

**lemma** *Induction-ax-imp-hfthm*:

**assumes**  $x \in \text{Induction-ax}$  **obtains**  $A$  **where**  $x = \llbracket \langle A \rangle \rrbracket e \wedge \{\} \vdash A$   
*<proof>*

**lemma** *Exists-imp-hfthm*:  $\llbracket \text{Exists } \llbracket \langle A \rangle \rrbracket e y; \{\} \vdash A \rrbracket \implies \exists B. y = \llbracket \langle B \rangle \rrbracket e \wedge \{\} \vdash B$   
*<proof>*

**lemma** *Subst-imp-hfthm*:  $\llbracket \text{Subst } \llbracket \langle A \rangle \rrbracket e y; \{\} \vdash A \rrbracket \implies \exists B. y = \llbracket \langle B \rangle \rrbracket e \wedge \{\} \vdash B$   
*<proof>*

**lemma** *eval-Neg-imp-Neg*:  $\llbracket \langle \alpha \rangle \rrbracket e = q\text{-Neg } x \implies \exists A. \alpha = \text{Neg } A \wedge \llbracket \langle A \rangle \rrbracket e = x$

*<proof>*

**lemma** *eval-Disj-imp-Disj*:  $\llbracket \langle \alpha \rangle \rrbracket e = q\text{-Disj } x y \implies \exists A B. \alpha = A \text{ OR } B \wedge \llbracket \langle A \rangle \rrbracket e = x \wedge \llbracket \langle B \rangle \rrbracket e = y$

*<proof>*

**lemma** *Prf-imp-proved*: **assumes**  $\text{Prf } s k x$  **shows**  $\exists A. x = \llbracket \langle A \rangle \rrbracket e \wedge \{\} \vdash A$

*<proof>*

**corollary** *Pf-quot-imp-is-proved*:  $\text{Pf } \llbracket \langle \alpha \rangle \rrbracket e \implies \{\} \vdash \alpha$

*<proof>*

Proposition 4.4!

**theorem** *proved-iff-proved-PfP*:  $\{\} \vdash \alpha \longleftrightarrow \{\} \vdash \text{PfP } \langle \alpha \rangle$

*<proof>*

**end**

# Kapitel 7

## Uniqueness Results: Syntactic Relations are Functions

```
theory Functions
imports Coding-Predicates
begin
```

### 7.0.1 SeqStTermP

```
lemma not-IndP-VarP: {IndP x, VarP x} ⊢ A
⟨proof⟩
```

It IS a pair, but not just any pair.

```
lemma IndP-HPairE: insert (IndP (HPair (HPair Zero (HPair Zero Zero)) x))
H ⊢ A
⟨proof⟩
```

```
lemma atom-HPairE:
  assumes H ⊢ x EQ HPair (HPair Zero (HPair Zero Zero)) y
  shows insert (IndP x OR x NEQ v) H ⊢ A
⟨proof⟩
```

```
lemma SeqStTermP-lemma:
  assumes atom m # (v,i,t,u,s,k,n,sm,sm',sn,sn') atom n # (v,i,t,u,s,k,sm,sm',sn,sn')
         atom sm # (v,i,t,u,s,k,sm',sn,sn') atom sm' # (v,i,t,u,s,k,sn,sn')
         atom sn # (v,i,t,u,s,k,sn') atom sn' # (v,i,t,u,s,k)
  shows { SeqStTermP v i t u s k }
        ⊢ ((t EQ v AND u EQ i) OR
          ((IndP t OR t NEQ v) AND u EQ t)) OR
          Ex m (Ex n (Ex sm (Ex sm' (Ex sn (Ex sn' (Var m IN k AND Var n
IN k AND
          SeqStTermP v i (Var sm) (Var sm') s (Var m) AND
          SeqStTermP v i (Var sn) (Var sn') s (Var n) AND
```



$t \text{ EQ } Q\text{-Eats } (Var \ sm) \ (Var \ sn) \ \text{AND}$   
 $u \text{ EQ } Q\text{-Eats } (Var \ sm') \ (Var \ sn'))))$

*<proof>*

**lemma** *SeqStTermP-unique*:  $\{SeqStTermP \ v \ a \ t \ u \ s \ kk, SeqStTermP \ v \ a \ t \ u' \ s' \ kk'\} \vdash u' \text{ EQ } u$

*<proof>*

**theorem** *SubstTermP-unique*:  $\{SubstTermP \ v \ tm \ t \ u, SubstTermP \ v \ tm \ t \ u'\} \vdash u' \text{ EQ } u$

*<proof>*

### 7.0.2 *SubstAtomicP*

**lemma** *SubstTermP-eq*:

$\llbracket H \vdash SubstTermP \ v \ tm \ x \ z; insert \ (SubstTermP \ v \ tm \ y \ z) \ H \vdash A \rrbracket \implies insert \ (x \text{ EQ } y) \ H \vdash A$

*<proof>*

**lemma** *SubstAtomicP-unique*:  $\{SubstAtomicP \ v \ tm \ x \ y, SubstAtomicP \ v \ tm \ x \ y'\} \vdash y' \text{ EQ } y$

*<proof>*

### 7.0.3 *SeqSubstFormP*

**lemma** *SeqSubstFormP-lemma*:

**assumes**  $atom \ m \ \# \ (v, u, x, y, s, k, n, sm, sm', sn, sn')$   $atom \ n \ \# \ (v, u, x, y, s, k, sm, sm', sn, sn')$   
 $atom \ sm \ \# \ (v, u, x, y, s, k, sm', sn, sn')$   $atom \ sm' \ \# \ (v, u, x, y, s, k, sn, sn')$   
 $atom \ sn \ \# \ (v, u, x, y, s, k, sn')$   $atom \ sn' \ \# \ (v, u, x, y, s, k)$

**shows**  $\{ SeqSubstFormP \ v \ u \ x \ y \ s \ k \}$   
 $\vdash SubstAtomicP \ v \ u \ x \ y \ \text{OR}$   
 $Ex \ m \ (Ex \ n \ (Ex \ sm \ (Ex \ sm' \ (Ex \ sn \ (Ex \ sn' \ (Var \ m \ \text{IN } k \ \text{AND } Var \ n \ \text{IN } k \ \text{AND}$   
 $SeqSubstFormP \ v \ u \ (Var \ sm) \ (Var \ sm') \ s \ (Var \ m) \ \text{AND}$   
 $SeqSubstFormP \ v \ u \ (Var \ sn) \ (Var \ sn') \ s \ (Var \ n) \ \text{AND}$   
 $((x \text{ EQ } Q\text{-Disj } (Var \ sm) \ (Var \ sn) \ \text{AND } y \text{ EQ } Q\text{-Disj } (Var \ sm')$   
 $(Var \ sn')) \ \text{OR}$   
 $(x \text{ EQ } Q\text{-Neg } (Var \ sm) \ \text{AND } y \text{ EQ } Q\text{-Neg } (Var \ sm')) \ \text{OR}$   
 $(x \text{ EQ } Q\text{-Ex } (Var \ sm) \ \text{AND } y \text{ EQ } Q\text{-Ex } (Var \ sm'))))$

*<proof>*

**lemma**

**shows** *Neg-SubstAtomicP-Fls*:  $\{y \text{ EQ } Q\text{-Neg } z, SubstAtomicP \ v \ tm \ y \ y'\} \vdash Fls$

(*is ?thesis1*)

**and** *Disj-SubstAtomicP-Fls*:  $\{y \text{ EQ } Q\text{-Disj } z \ w, SubstAtomicP \ v \ tm \ y \ y'\} \vdash Fls$

(*is ?thesis2*)

**and** *Ex-SubstAtomicP-Fls*:  $\{y \text{ EQ } Q\text{-Ex } z, SubstAtomicP \ v \ tm \ y \ y'\} \vdash Fls$

(*is ?thesis3*)

*<proof>*

**lemma** *SeqSubstFormP-eq*:

$$\llbracket H \vdash \text{SeqSubstFormP } v \text{ tm } x \text{ z } s \text{ k}; \text{ insert } (\text{SeqSubstFormP } v \text{ tm } y \text{ z } s \text{ k}) H \vdash A \rrbracket$$

$$\implies \text{insert } (x \text{ EQ } y) H \vdash A$$

*<proof>*

**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>*

#### 7.0.4 *SubstFormP*

**theorem** *SubstFormP-unique*:  $\{\text{SubstFormP } v \text{ tm } x \text{ y}, \text{SubstFormP } v \text{ tm } x \text{ y'}\} \vdash y' \text{ EQ } y$

*<proof>*

**end**

# Kapitel 8

## Section 6 Material and Gdel's First Incompleteness Theorem

```
theory Goedel-I
imports Pf-Predicates Functions
begin
```

### 8.1 The Function W and Lemma 6.1

#### 8.1.1 Predicate form, defined on sequences

```
definition SeqWR :: hf  $\Rightarrow$  hf  $\Rightarrow$  hf  $\Rightarrow$  bool
  where SeqWR s k y  $\equiv$  LstSeq s k y  $\wedge$  app s 0 = 0  $\wedge$ 
    ( $\forall l \in k. \text{app } s \text{ (succ } l) = \text{q-Eats (app } s \text{ } l) \text{ (app } s \text{ } l)$ )
```

```
nominal-function SeqWRP :: tm  $\Rightarrow$  tm  $\Rightarrow$  tm  $\Rightarrow$  fm
  where  $\llbracket \text{atom } l \ \sharp \ (s, k, sl); \text{atom } sl \ \sharp \ (s) \rrbracket \Longrightarrow$ 
    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))
  <proof>
```

```
nominal-termination (eqvt)
  <proof>
```

```
lemma
  shows SeqWRP-fresh-iff [simp]: a  $\sharp$  SeqWRP s k y  $\longleftrightarrow$  a  $\sharp$  s  $\wedge$  a  $\sharp$  k  $\wedge$  a  $\sharp$  y
  (is ?thesis1)
  and eval-fm-SeqWRP [simp]: eval-fm e (SeqWRP s k y)  $\longleftrightarrow$  SeqWR  $\llbracket s \rrbracket e$ 
 $\llbracket k \rrbracket e \llbracket y \rrbracket e$  (is ?thesis2)
  and SeqWRP-sf [iff]: Sigma-fm (SeqWRP s k y) (is ?thsf)
```

*<proof>*

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

$(SeqWRP\ s\ k\ y)(i::=t) = SeqWRP\ (subst\ i\ t\ s)\ (subst\ i\ t\ k)\ (subst\ i\ t\ y)$

*<proof>*

**lemma** *SeqWRP-cong*:

**assumes**  $H \vdash s\ EQ\ s'$  **and**  $H \vdash k\ EQ\ k'$  **and**  $H \vdash y\ EQ\ y'$

**shows**  $H \vdash SeqWRP\ s\ k\ y\ IFF\ SeqWRP\ s'\ k'\ y'$

*<proof>*

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

### 8.1.2 Predicate form of W

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

**where**  $WR\ x\ y \equiv (\exists s. SeqWR\ s\ x\ y)$

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

**where**  $\llbracket atom\ s\ \#\ (x,y) \rrbracket \Longrightarrow$

$WRP\ x\ y = Ex\ s\ (SeqWRP\ (Var\ s)\ x\ y)$

*<proof>*

**nominal-termination** (*eqvt*)

*<proof>*

**lemma**

**shows** *WRP-fresh-iff* [*simp*]:  $a\ \#\ WRP\ x\ y \longleftrightarrow a\ \#\ x \wedge a\ \#\ y$  (**is** *?thesis1*)

**and** *eval-fm-WRP* [*simp*]:  $eval\ fm\ e\ (WRP\ x\ y) \longleftrightarrow WR\ \llbracket x \rrbracket e\ \llbracket y \rrbracket e$  (**is** *?thesis2*)

**and** *sigma-fm-WRP* [*simp*]:  $Sigma\ fm\ (WRP\ x\ y)$  (**is** *?thsf*)

*<proof>*

**lemma** *WRP-subst* [*simp*]:  $(WRP\ x\ y)(i::=t) = WRP\ (subst\ i\ t\ x)\ (subst\ i\ t\ y)$

*<proof>*

**lemma** *WRP-cong*:  $H \vdash t\ EQ\ t' \Longrightarrow H \vdash u\ EQ\ u' \Longrightarrow H \vdash WRP\ t\ u\ IFF\ WRP\ t'\ u'$

*<proof>*

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

**lemma** *WR0-iff*:  $WR\ 0\ y \longleftrightarrow y=0$

*<proof>*

**lemma** *WR0*:  $WR\ 0\ 0$

*<proof>*

**lemma** *WR-succ-iff*: **assumes**  $i: Ord\ i$  **shows**  $WR\ (succ\ i)\ z = (\exists y. z = q\ Eats$

$y \ y \wedge \text{WR } i \ y)$   
 $\langle \text{proof} \rangle$

**lemma** *WR-succ*:  $\text{Ord } i \implies \text{WR } (\text{succ } i) \ (q\text{-Eats } y \ y) = \text{WR } i \ y$   
 $\langle \text{proof} \rangle$

**lemma** *WR-ord-of*:  $\text{WR } (\text{ord-of } i) \ [\ll \text{ORD-OF } i \gg] e$   
 $\langle \text{proof} \rangle$

Lemma 6.1

**lemma** *WR-quot-Var*:  $\text{WR } [\ll \text{Var } x \gg] e \ [\ll\ll \text{Var } x \gg\gg] e$   
 $\langle \text{proof} \rangle$

**lemma** *ground-WRP* [*simp*]:  $\text{ground-fm } (\text{WRP } x \ y) \longleftrightarrow \text{ground } x \wedge \text{ground } y$   
 $\langle \text{proof} \rangle$

**lemma** *prove-WRP*:  $\{\} \vdash \text{WRP } \ll \text{Var } x \gg \ll\ll \text{Var } x \gg\gg$   
 $\langle \text{proof} \rangle$

### 8.1.3 Proving that these relations are functions

**lemma** *SeqWRP-Zero-E*:  
**assumes**  $\text{insert } (y \ \text{EQ } \text{Zero}) \ H \vdash A \quad H \vdash k \ \text{EQ } \text{Zero}$   
**shows**  $\text{insert } (\text{SeqWRP } s \ k \ y) \ H \vdash A$   
 $\langle \text{proof} \rangle$

**lemma** *SeqWRP-SUCC-lemma*:  
**assumes**  $y': \text{atom } y' \ \# \ (s, k, y)$   
**shows**  $\{\text{SeqWRP } s \ (\text{SUCC } k) \ y\} \vdash \text{Ex } y' \ (\text{SeqWRP } s \ k \ (\text{Var } y') \ \text{AND } y \ \text{EQ } \text{Q-Succ } (\text{Var } y'))$   
 $\langle \text{proof} \rangle$

**lemma** *SeqWRP-SUCC-E*:  
**assumes**  $y': \text{atom } y' \ \# \ (s, k, y)$  **and**  $k': H \vdash k' \ \text{EQ } (\text{SUCC } k)$   
**shows**  $\text{insert } (\text{SeqWRP } s \ k' \ y) \ H \vdash \text{Ex } y' \ (\text{SeqWRP } s \ k \ (\text{Var } y') \ \text{AND } y \ \text{EQ } \text{Q-Succ } (\text{Var } y'))$   
 $\langle \text{proof} \rangle$

**lemma** *SeqWRP-unique*:  $\{\text{OrdP } x, \text{SeqWRP } s \ x \ y, \text{SeqWRP } s' \ x \ y'\} \vdash y' \ \text{EQ } y$   
 $\langle \text{proof} \rangle$

**theorem** *WRP-unique*:  $\{\text{OrdP } x, \text{WRP } x \ y, \text{WRP } x \ y'\} \vdash y' \ \text{EQ } y$   
 $\langle \text{proof} \rangle$

### 8.1.4 The equivalent function

**definition**  $W :: hf \Rightarrow tm$   
**where**  $W \equiv \text{hmemrec } (\lambda f \ z. \ \text{if } z=0 \ \text{then } \text{Zero} \ \text{else } \text{Q-Eats } (f \ (\text{pred } z)) \ (f \ (\text{pred } z)))$

**lemma**  $W0$  [simp]:  $W\ 0 = Zero$   
 ⟨proof⟩

**lemma**  $W$ -succ [simp]:  $Ord\ i \implies W\ (succ\ i) = Q\text{-Eats}\ (W\ i)\ (W\ i)$   
 ⟨proof⟩

**lemma**  $W$ -ord-of [simp]:  $W\ (ord\text{-of}\ i) = \ll ORD\text{-OF}\ i \gg$   
 ⟨proof⟩

**lemma**  $WR$ -iff-eq- $W$ :  $Ord\ x \implies WR\ x\ y \longleftrightarrow y = \ll W\ x \gg e$   
 ⟨proof⟩

## 8.2 The Function HF and Lemma 6.2

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

**where**  $SeqHR\ x\ x'\ s\ k \equiv$

$BuildSeq2\ (\lambda y\ y'. Ord\ y \wedge WR\ y\ y')$   
 $(\lambda u\ u'\ v\ v'\ w\ w'. u = \langle v, w \rangle \wedge u' = q\text{-HPair}\ v'\ w')\ s\ k\ x\ x'$

### 8.2.1 Defining the syntax: quantified body

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

**where**  $\ll atom\ l \# (s, k, sl, sl', m, n, sm, sm', sn, sn')$ ;  
 $atom\ sl \# (s, sl', m, n, sm, sm', sn, sn')$ ;  
 $atom\ sl' \# (s, m, n, sm, sm', sn, sn')$ ;  
 $atom\ m \# (s, n, sm, sm', sn, sn')$ ;  
 $atom\ n \# (s, sm, sm', sn, sn')$ ;  
 $atom\ sm \# (s, sm', sn, sn')$ ;  
 $atom\ sm' \# (s, sn, sn')$ ;  
 $atom\ sn \# (s, sn')$ ;  
 $atom\ sn' \# (s) \gg \implies$

$SeqHRP\ x\ x'\ s\ k =$

$LstSeqP\ s\ k\ (HPair\ x\ x')\ AND$

$All2\ l\ (SUCC\ k)\ (Ex\ sl\ (Ex\ sl'\ (HPair\ (Var\ l)\ (HPair\ (Var\ sl)\ (Var\ sl'))\ IN\ s\ AND$

$((OrdP\ (Var\ sl)\ AND\ WRP\ (Var\ sl)\ (Var\ sl'))\ OR$

$Ex\ m\ (Ex\ n\ (Ex\ sm\ (Ex\ sm'\ (Ex\ sn\ (Ex\ sn'\ (Var\ m\ IN\ Var\ l\ AND$

$Var\ n\ IN\ Var\ l\ AND$

$HPair\ (Var\ m)\ (HPair\ (Var\ sm)\ (Var\ sm'))\ IN\ s\ AND$

$HPair\ (Var\ n)\ (HPair\ (Var\ sn)\ (Var\ sn'))\ IN\ s\ AND$

$Var\ sl\ EQ\ HPair\ (Var\ sm)\ (Var\ sn)\ AND$

$Var\ sl'\ EQ\ Q\text{-HPair}\ (Var\ sm')\ (Var\ sn'))))))))$

⟨proof⟩

**nominal-termination** (eqvt)

⟨proof⟩

**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** *eval-fm-SeqHRP* [simp]:  
 $\text{eval-fm } e (\text{SeqHRP } x x' s k) \longleftrightarrow \text{SeqHR } \llbracket x \rrbracket e \llbracket x' \rrbracket e \llbracket s \rrbracket e \llbracket k \rrbracket e$  (**is** ?thesis2)  
**and** *SeqHRP-sf* [iff]: *Sigma-fm* (*SeqHRP*  $x x' s k$ ) (**is** ?thsf)  
**and** *SeqHRP-imp-OrdP*:  $\{ \text{SeqHRP } x y s k \} \vdash \text{OrdP } k$  (**is** ?thord)  
 <proof>

**lemma** *SeqHRP-subst* [simp]:  
 $(\text{SeqHRP } x x' s k)(i::=t) = \text{SeqHRP } (\text{subst } i t x) (\text{subst } i t x') (\text{subst } i t s)$   
 $(\text{subst } i t k)$   
 <proof>

**lemma** *SeqHRP-cong*:  
**assumes**  $H \vdash x \text{EQ } x'$  **and**  $H \vdash y \text{EQ } y'$   $H \vdash s \text{EQ } s'$  **and**  $H \vdash k \text{EQ } k'$   
**shows**  $H \vdash \text{SeqHRP } x y s k \text{ IFF } \text{SeqHRP } x' y' s' k'$   
 <proof>

## 8.2.2 Defining the syntax: main predicate

**definition** *HR* ::  $hf \Rightarrow hf \Rightarrow \text{bool}$   
**where**  $HR x x' \equiv \exists s k. \text{SeqHR } x x' s k$

**nominal-function** *HRP* ::  $tm \Rightarrow tm \Rightarrow fm$   
**where**  $\llbracket \text{atom } s \# (x, x', k); \text{atom } k \# (x, x') \rrbracket \Longrightarrow$   
 $HRP x x' = Ex s (Ex k (\text{SeqHRP } x x' (\text{Var } s) (\text{Var } k)))$   
 <proof>

**nominal-termination** (*eqvt*)  
 <proof>

**lemma**  
**shows** *HRP-fresh-iff* [simp]:  $a \# HRP x x' \longleftrightarrow a \# x \wedge a \# x'$  (**is** ?thesis1)  
**and** *eval-fm-HRP* [simp]:  $\text{eval-fm } e (HRP x x') \longleftrightarrow HR \llbracket x \rrbracket e \llbracket x' \rrbracket e$  (**is** ?thesis2)  
**and** *HRP-sf* [iff]: *Sigma-fm* (*HRP*  $x x'$ ) (**is** ?thsf)  
 <proof>

**lemma** *HRP-subst* [simp]:  $(HRP x x')(i::=t) = HRP (\text{subst } i t x) (\text{subst } i t x')$   
 <proof>

## 8.2.3 Proving that these relations are functions

**lemma** *SeqHRP-lemma*:  
**assumes**  $\text{atom } m \# (x, x', s, k, n, sm, sm', sn, sn')$   $\text{atom } n \# (x, x', s, k, sm, sm', sn, sn')$   
 $\text{atom } sm \# (x, x', s, k, sm', sn, sn')$   $\text{atom } sm' \# (x, x', s, k, sn, sn')$   
 $\text{atom } sn \# (x, x', s, k, sn')$   $\text{atom } sn' \# (x, x', s, k)$   
**shows**  $\{ \text{SeqHRP } x x' s k \}$   
 $\vdash (\text{OrdP } x \text{ AND } \text{WRP } x x') \text{ OR}$   
 $Ex m (Ex n (Ex sm (Ex sm' (Ex sn (Ex sn' (\text{Var } m \text{ IN } k \text{ AND } \text{Var } n \text{ IN } k \text{ AND}$

$$\text{SeqHRP } (\text{Var } sm) (\text{Var } sm') s (\text{Var } m) \text{ AND}$$

$$\text{SeqHRP } (\text{Var } sn) (\text{Var } sn') s (\text{Var } n) \text{ AND}$$

$$x \text{ EQ } \text{HPair } (\text{Var } sm) (\text{Var } sn) \text{ AND}$$

$$x' \text{ EQ } \text{Q-HPair } (\text{Var } sm') (\text{Var } sn') \text{))))))$$

$\langle \text{proof} \rangle$

**lemma** *SeqHRP-unique*:  $\{\text{SeqHRP } x y s u, \text{SeqHRP } x y' s' u'\} \vdash y' \text{ EQ } y$

$\langle \text{proof} \rangle$

**theorem** *HRP-unique*:  $\{\text{HRP } x y, \text{HRP } x y'\} \vdash y' \text{ EQ } y$

$\langle \text{proof} \rangle$

### 8.2.4 Finally The Function HF Itself

**definition** *HF* ::  $hf \Rightarrow tm$

**where**  $HF \equiv \text{hmemrec } (\lambda f z. \text{if Ord } z \text{ then } W z \text{ else } \text{Q-HPair } (f (\text{hfst } z)) (f (\text{hsnd } z)))$

**lemma** *HF-Ord [simp]*:  $\text{Ord } i \Longrightarrow HF i = W i$

$\langle \text{proof} \rangle$

**lemma** *HF-pair [simp]*:  $HF (\text{hpair } x y) = \text{Q-HPair } (HF x) (HF y)$

$\langle \text{proof} \rangle$

**lemma** *SeqHR-hpair*:  $\text{SeqHR } x1 x3 s1 k1 \Longrightarrow \text{SeqHR } x2 x4 s2 k2 \Longrightarrow \exists s k. \text{SeqHR}$

$\langle x1, x2 \rangle (\text{q-HPair } x3 x4) s k$

$\langle \text{proof} \rangle$

**lemma** *HR-H*:  $\text{coding-hf } x \Longrightarrow \text{HR } x \llbracket HF x \rrbracket e$

$\langle \text{proof} \rangle$

Lemma 6.2

**lemma** *HF-quot-coding-tm*:  $\text{coding-tm } t \Longrightarrow HF \llbracket t \rrbracket e = \langle t \rangle$

$\langle \text{proof} \rangle$

**lemma** *HR-quot-fm*: **fixes**  $A::fm$  **shows**  $\text{HR } \llbracket \langle A \rangle \rrbracket e \llbracket \langle \langle A \rangle \rangle \rrbracket e$

$\langle \text{proof} \rangle$

**lemma** *prove-HRP*: **fixes**  $A::fm$  **shows**  $\{\} \vdash \text{HRP } \langle A \rangle \langle \langle A \rangle \rangle$

$\langle \text{proof} \rangle$

### 8.3 The Function K and Lemma 6.3

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

**where**  $\text{atom } y \# (v, x, x') \Longrightarrow$

$KRP v x x' = \text{Ex } y (\text{HRP } x (\text{Var } y) \text{ AND } \text{SubstFormP } v (\text{Var } y) x x')$

$\langle \text{proof} \rangle$

**nominal-termination** (*eqvt*)



*<proof>*

**lemma** *KRP-fresh-iff* [*simp*]:  $a \# KRP\ v\ x\ x' \longleftrightarrow a \# v \wedge a \# x \wedge a \# x'$   
*<proof>*

**lemma** *KRP-subst* [*simp*]:  $(KRP\ v\ x\ x')(i::=t) = KRP\ (subst\ i\ t\ v)\ (subst\ i\ t\ x)$   
*<proof>*

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

**lemma** *prove-SubstFormP*:  $\{\} \vdash SubstFormP\ \langle Var\ i \rangle \langle\langle A \rangle\rangle \langle A \rangle \langle A(i::=\langle A \rangle) \rangle$   
*<proof>*

**lemma** *prove-KRP*:  $\{\} \vdash KRP\ \langle Var\ i \rangle \langle A \rangle \langle A(i::=\langle A \rangle) \rangle$   
*<proof>*

**lemma** *KRP-unique*:  $\{KRP\ v\ x\ y, KRP\ v\ x\ y'\} \vdash y' EQ\ y$   
*<proof>*

**lemma** *KRP-subst-fm*:  $\{KRP\ \langle Var\ i \rangle \langle\beta\rangle\ (Var\ j)\} \vdash Var\ j EQ\ \langle\beta(i::=\langle\beta\rangle)\rangle$   
*<proof>*

## 8.4 The Diagonal Lemma and Gdel's Theorem

**lemma** *diagonal*:

**obtains**  $\delta$  **where**  $\{\} \vdash \delta$  *IFF*  $\alpha(i::=\langle\delta\rangle)$   $supp\ \delta = supp\ \alpha - \{atom\ i\}$   
*<proof>*

Gdel's first incompleteness theorem: Our theory is incomplete. NB it is provably consistent

**theorem** *Goedel-I*:

**obtains**  $\delta$  **where**  $\{\} \vdash \delta$  *IFF*  $Neg\ (Pfp\ \langle\delta\rangle) \neg \{\} \vdash \delta \neg \{\} \vdash Neg\ \delta$   
*eval-fm e  $\delta$  ground-fm  $\delta$*

*<proof>*

**end**

## Kapitel 9

# Syntactic Preliminaries for the Second Incompleteness Theorem

```
theory II-Prelims
imports Pf-Predicates
begin
```

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

```
lemma VarP-Var [intro]:  $H \vdash \text{VarP} \ll \text{Var } i \gg$ 
⟨proof⟩
```

```
lemma VarP-neg-IndP:  $\{t \text{ EQ } v, \text{VarP } v, \text{IndP } t\} \vdash \text{Fls}$ 
⟨proof⟩
```

```
lemma OrdP-ORD-OF [intro]:  $H \vdash \text{OrdP} (\text{ORD-OF } n)$ 
⟨proof⟩
```

```
lemma Mem-HFun-Sigma-OrdP:  $\{\text{HPair } t \text{ u IN } f, \text{HFun-Sigma } f\} \vdash \text{OrdP } t$ 
⟨proof⟩
```

### 9.1 NotInDom

```
nominal-function NotInDom ::  $tm \Rightarrow tm \Rightarrow fm$ 
  where  $\text{atom } z \# (t, r) \Longrightarrow \text{NotInDom } t \text{ r} = \text{All } z (\text{Neg } (\text{HPair } t (\text{Var } z) \text{ IN } r))$ 
⟨proof⟩
```

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

```
lemma NotInDom-fresh-iff [simp]:  $a \# \text{NotInDom } t \text{ r} \longleftrightarrow a \# (t, r)$ 
⟨proof⟩
```

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

**lemma** *NotInDom-cong*:  $H \vdash t \ EQ \ t' \implies H \vdash r \ EQ \ r' \implies H \vdash \text{NotInDom } t \ r$   
 IFF  $\text{NotInDom } t' \ r'$   
 ⟨proof⟩

**lemma** *NotInDom-Zero*:  $H \vdash \text{NotInDom } t \ \text{Zero}$   
 ⟨proof⟩

**lemma** *NotInDom-Fls*:  $\{\text{HPair } d \ d' \ IN \ r, \ \text{NotInDom } d \ r\} \vdash A$   
 ⟨proof⟩

**lemma** *NotInDom-Contra*:  $H \vdash \text{NotInDom } d \ r \implies H \vdash \text{HPair } x \ y \ IN \ r \implies \text{insert}$   
 $(x \ EQ \ d) \ H \vdash A$   
 ⟨proof⟩

## 9.2 Restriction of a Sequence to a Domain

**nominal-function** *RestrictedP* ::  $tm \Rightarrow tm \Rightarrow tm \Rightarrow fn$   
**where**  $\llbracket \text{atom } x \ \sharp \ (y, f, k, g); \ \text{atom } y \ \sharp \ (f, k, g) \rrbracket \implies$   
 $\text{RestrictedP } f \ k \ g =$   
 $g \ \text{SUBS } f \ \text{AND}$   
 $\text{All } x \ (\text{All } y \ (\text{HPair } (\text{Var } x) \ (\text{Var } y) \ IN \ g \ \text{IFF}$   
 $(\text{Var } x) \ IN \ k \ \text{AND } \text{HPair } (\text{Var } x) \ (\text{Var } y) \ IN \ f))$   
 ⟨proof⟩

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

**lemma** *RestrictedP-fresh-iff* [simp]:  $a \ \sharp \ \text{RestrictedP } f \ k \ g \longleftrightarrow a \ \sharp \ f \wedge a \ \sharp \ k \wedge a \ \sharp \ g$   
 ⟨proof⟩

**lemma** *subst-fm-RestrictedP* [simp]:  
 $(\text{RestrictedP } f \ k \ g)(i ::= u) = \text{RestrictedP } (\text{subst } i \ u \ f) \ (\text{subst } i \ u \ k) \ (\text{subst } i \ u \ g)$   
 ⟨proof⟩

**lemma** *RestrictedP-cong*:  
 $\llbracket H \vdash f \ EQ \ f'; \ H \vdash k \ EQ \ A'; \ H \vdash g \ EQ \ g' \rrbracket$   
 $\implies H \vdash \text{RestrictedP } f \ k \ g \ \text{IFF } \text{RestrictedP } f' \ A' \ g'$   
 ⟨proof⟩

**lemma** *RestrictedP-Zero*:  $H \vdash \text{RestrictedP } \text{Zero } k \ \text{Zero}$   
 ⟨proof⟩

**lemma** *RestrictedP-Mem*:  $\{\text{RestrictedP } s \ k \ s', \ \text{HPair } a \ b \ IN \ s, \ a \ IN \ k\} \vdash \text{HPair}$

$a \ b \ IN \ s'$   
 $\langle proof \rangle$

**lemma** *RestrictedP-imp-Subset*:  $\{ \text{RestrictedP } s \ k \ s' \} \vdash s' \ \text{SUBS } s$   
 $\langle proof \rangle$

**lemma** *RestrictedP-Mem2*:  
 $\{ \text{RestrictedP } s \ k \ s', \ \text{HPair } a \ b \ IN \ s' \} \vdash \text{HPair } a \ b \ IN \ s \ \text{AND } a \ IN \ k$   
 $\langle proof \rangle$

**lemma** *RestrictedP-Mem-D*:  $H \vdash \text{RestrictedP } s \ k \ t \implies H \vdash a \ IN \ t \implies \text{insert } (a \ IN \ s) \ H \vdash A \implies H \vdash A$   
 $\langle proof \rangle$

**lemma** *RestrictedP-Eats*:  
 $\{ \text{RestrictedP } s \ k \ s', \ a \ IN \ k \} \vdash \text{RestrictedP } (\text{Eats } s \ (\text{HPair } a \ b)) \ k \ (\text{Eats } s' \ (\text{HPair } a \ b)) \ \langle proof \rangle$

**lemma** *exists-RestrictedP*:  
**assumes**  $s$ :  $\text{atom } s \ \sharp \ (f, k)$   
**shows**  $H \vdash \text{Ex } s \ (\text{RestrictedP } f \ k \ (\text{Var } s)) \ \langle proof \rangle$

**lemma** *cut-RestrictedP*:  
**assumes**  $s$ :  $\text{atom } s \ \sharp \ (f, k, A)$  **and**  $\forall C \in H. \text{atom } s \ \sharp \ C$   
**shows**  $\text{insert } (\text{RestrictedP } f \ k \ (\text{Var } s)) \ H \vdash A \implies H \vdash A$   
 $\langle proof \rangle$

**lemma** *RestrictedP-NotInDom*:  $\{ \text{RestrictedP } s \ k \ s', \ \text{Neg } (j \ IN \ k) \} \vdash \text{NotInDom } j \ s'$   
 $\langle proof \rangle$

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

### 9.3 Applications to LstSeqP

**lemma** *HFun-Sigma-Eats*:  
**assumes**  $H \vdash \text{HFun-Sigma } r \ H \vdash \text{NotInDom } d \ r \ H \vdash \text{OrdP } d$   
**shows**  $H \vdash \text{HFun-Sigma } (\text{Eats } r \ (\text{HPair } d \ d')) \ \langle proof \rangle$

**lemma** *HFun-Sigma-single [iff]*:  $H \vdash \text{OrdP } d \implies H \vdash \text{HFun-Sigma } (\text{Eats } \text{Zero} \ (\text{HPair } d \ d'))$   
 $\langle proof \rangle$

**lemma** *LstSeqP-single [iff]*:  $H \vdash \text{LstSeqP } (\text{Eats } \text{Zero} \ (\text{HPair } \text{Zero } x)) \ \text{Zero } x$   
 $\langle proof \rangle$

**lemma** *NotInDom-LstSeqP-Eats*:  
 $\{ \text{NotInDom } (\text{SUCC } k) \ s, \ \text{LstSeqP } s \ k \ y \} \vdash \text{LstSeqP } (\text{Eats } s \ (\text{HPair } (\text{SUCC } k) \ z)) \ (\text{SUCC } k) \ z$   
 $\langle proof \rangle$

**lemma** *RestrictedP-HDomain-Incl*:  $\{ \text{HDomain-Incl } s \ k, \ \text{RestrictedP } s \ k \ s' \} \vdash \text{HDomain-Incl}$

$s' k$   
 $\langle proof \rangle$

**lemma** *RestrictedP-HFun-Sigma*:  $\{HFun\text{-Sigma } s, RestrictedP\ s\ k\ s'\} \vdash HFun\text{-Sigma } s'$   
 $\langle proof \rangle$

**lemma** *RestrictedP-LstSeqP*:  
 $\{RestrictedP\ s\ (SUCC\ k)\ s', LstSeqP\ s\ k\ y\} \vdash LstSeqP\ s'\ k\ y$   
 $\langle proof \rangle$

**lemma** *RestrictedP-LstSeqP-Eats*:  
 $\{RestrictedP\ s\ (SUCC\ k)\ s', LstSeqP\ s\ k\ y\}$   
 $\vdash LstSeqP\ (Eats\ s'\ (HPair\ (SUCC\ k)\ z))\ (SUCC\ k)\ z$   
 $\langle proof \rangle$

## 9.4 Ordinal Addition

### 9.4.1 Predicate form, defined on sequences

**nominal-function** *SeqHaddP* ::  $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$   
**where**  $\llbracket atom\ l\ \#\ (sl, s, k, j); atom\ sl\ \#\ (s, j) \rrbracket \implies$   
 $SeqHaddP\ s\ j\ k\ y = LstSeqP\ s\ k\ y\ AND$   
 $HPair\ Zero\ j\ IN\ s\ AND$   
 $All2\ l\ k\ (Ex\ sl\ (HPair\ (Var\ l)\ (Var\ sl)\ IN\ s\ AND$   
 $HPair\ (SUCC\ (Var\ l))\ (SUCC\ (Var\ sl))\ IN\ s))$   
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)  
 $\langle proof \rangle$

**lemma** *SeqHaddP-fresh-iff* [*simp*]:  $a\ \#\ SeqHaddP\ s\ j\ k\ y \longleftrightarrow a\ \#\ s \wedge a\ \#\ j \wedge a\ \#\ k \wedge a\ \#\ y$   
 $\langle proof \rangle$

**lemma** *SeqHaddP-subst* [*simp*]:  
 $(SeqHaddP\ s\ j\ k\ y)(i::=t) = SeqHaddP\ (subst\ i\ t\ s)\ (subst\ i\ t\ j)\ (subst\ i\ t\ k)$   
 $(subst\ i\ t\ y)$   
 $\langle proof \rangle$

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

**nominal-function** *HaddP* ::  $tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$   
**where**  $\llbracket atom\ s\ \#\ (x, y, z) \rrbracket \implies$   
 $HaddP\ x\ y\ z = Ex\ s\ (SeqHaddP\ (Var\ s)\ x\ y\ z)$   
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)  
 $\langle proof \rangle$

**lemma** *HaddP-fresh-iff* [simp]:  $a \# \text{HaddP } x \ y \ z \longleftrightarrow a \# x \wedge a \# y \wedge a \# z$   
 ⟨proof⟩

**lemma** *HaddP-subst* [simp]:  $(\text{HaddP } x \ y \ z)(i::=t) = \text{HaddP } (\text{subst } i \ t \ x) \ (\text{subst } i \ t \ y) \ (\text{subst } i \ t \ z)$   
 ⟨proof⟩

**lemma** *HaddP-cong*:  $\llbracket H \vdash t \ \text{EQ} \ t'; H \vdash u \ \text{EQ} \ u'; H \vdash v \ \text{EQ} \ v' \rrbracket \Longrightarrow H \vdash \text{HaddP } t \ u \ v \ \text{IFF} \ \text{HaddP } t' \ u' \ v'$   
 ⟨proof⟩

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

**lemma** *HaddP-Zero2*:  $H \vdash \text{HaddP } x \ \text{Zero } x$   
 ⟨proof⟩

**lemma** *HaddP-imp-OrdP*:  $\{\text{HaddP } x \ y \ z\} \vdash \text{OrdP } y$   
 ⟨proof⟩

**lemma** *HaddP-SUCC2*:  $\{\text{HaddP } x \ y \ z\} \vdash \text{HaddP } x \ (\text{SUCC } y) \ (\text{SUCC } z)$  ⟨proof⟩

## 9.4.2 Proving that these relations are functions

**lemma** *SeqHaddP-Zero-E*:  $\{\text{SeqHaddP } s \ w \ \text{Zero } z\} \vdash w \ \text{EQ} \ z$   
 ⟨proof⟩

**lemma** *SeqHaddP-SUCC-lemma*:  
**assumes**  $y': \text{atom } y' \ \# \ (s, j, k, y)$   
**shows**  $\{\text{SeqHaddP } s \ j \ (\text{SUCC } k) \ y\} \vdash \text{Ex } y' \ (\text{SeqHaddP } s \ j \ k \ (\text{Var } y')) \ \text{AND } y \ \text{EQ} \ \text{SUCC } (\text{Var } y')$   
 ⟨proof⟩

**lemma** *SeqHaddP-SUCC*:  
**assumes**  $H \vdash \text{SeqHaddP } s \ j \ (\text{SUCC } k) \ y \ \text{atom } y' \ \# \ (s, j, k, y)$   
**shows**  $H \vdash \text{Ex } y' \ (\text{SeqHaddP } s \ j \ k \ (\text{Var } y')) \ \text{AND } y \ \text{EQ} \ \text{SUCC } (\text{Var } y')$   
 ⟨proof⟩

**lemma** *SeqHaddP-unique*:  $\{\text{OrdP } x, \ \text{SeqHaddP } s \ w \ x \ y, \ \text{SeqHaddP } s' \ w \ x \ y'\} \vdash y' \ \text{EQ} \ y$  ⟨proof⟩

**lemma** *HaddP-unique*:  $\{\text{HaddP } w \ x \ y, \ \text{HaddP } w \ x \ y'\} \vdash y' \ \text{EQ} \ y$   
 ⟨proof⟩

**lemma** *HaddP-Zero1*: **assumes**  $H \vdash \text{OrdP } x$  **shows**  $H \vdash \text{HaddP } \text{Zero } x \ x$   
 ⟨proof⟩

**lemma** *HaddP-Zero-D1*:  $\text{insert } (\text{HaddP } \text{Zero } x \ y) \ H \vdash x \ \text{EQ} \ y$   
 ⟨proof⟩

**lemma** *HaddP-Zero-D2*:  $\text{insert } (\text{HaddP } x \text{ Zero } y) H \vdash x \text{ EQ } y$   
 $\langle \text{proof} \rangle$

**lemma** *HaddP-SUCC-Ex2*:  
**assumes**  $H \vdash \text{HaddP } x (\text{SUCC } y) z \text{ atom } z' \# (x,y,z)$   
**shows**  $H \vdash \text{Ex } z' (\text{HaddP } x y (\text{Var } z') \text{ AND } z \text{ EQ } \text{SUCC } (\text{Var } z'))$   
 $\langle \text{proof} \rangle$

**lemma** *HaddP-SUCC1*:  $\{ \text{HaddP } x y z \} \vdash \text{HaddP } (\text{SUCC } x) y (\text{SUCC } z) \langle \text{proof} \rangle$

**lemma** *HaddP-commute*:  $\{ \text{HaddP } x y z, \text{OrdP } x \} \vdash \text{HaddP } y x z \langle \text{proof} \rangle$

**lemma** *HaddP-SUCC-Ex1*:  
**assumes**  $\text{atom } i \# (x,y,z)$   
**shows**  $\text{insert } (\text{HaddP } (\text{SUCC } x) y z) (\text{insert } (\text{OrdP } x) H)$   
 $\vdash \text{Ex } i (\text{HaddP } x y (\text{Var } i) \text{ AND } z \text{ EQ } \text{SUCC } (\text{Var } i))$   
 $\langle \text{proof} \rangle$

**lemma** *HaddP-inv2*:  $\{ \text{HaddP } x y z, \text{HaddP } x y' z, \text{OrdP } x \} \vdash y' \text{ EQ } y \langle \text{proof} \rangle$

**lemma** *Mem-imp-subtract*:  $\langle \text{proof} \rangle$

**lemma** *HaddP-OrdP*:  
**assumes**  $H \vdash \text{HaddP } x y z H \vdash \text{OrdP } x$  **shows**  $H \vdash \text{OrdP } z \langle \text{proof} \rangle$

**lemma** *HaddP-Mem-cancel-left*:  
**assumes**  $H \vdash \text{HaddP } x y' z' H \vdash \text{HaddP } x y z H \vdash \text{OrdP } x$   
**shows**  $H \vdash z' \text{ IN } z \text{ IFF } y' \text{ IN } y \langle \text{proof} \rangle$

**lemma** *HaddP-Mem-cancel-right-Mem*:  
**assumes**  $H \vdash \text{HaddP } x' y z' H \vdash \text{HaddP } x y z H \vdash x' \text{ IN } x H \vdash \text{OrdP } x$   
**shows**  $H \vdash z' \text{ IN } z$   
 $\langle \text{proof} \rangle$

**lemma** *HaddP-Mem-cases*:  
**assumes**  $H \vdash \text{HaddP } k1 k2 k H \vdash \text{OrdP } k1$   
 $\text{insert } (x \text{ IN } k1) H \vdash A$   
 $\text{insert } (\text{Var } i \text{ IN } k2) (\text{insert } (\text{HaddP } k1 (\text{Var } i) x) H) \vdash A$   
**and**  $i: \text{atom } (i::\text{name}) \# (k1,k2,k,x,A)$  **and**  $\forall C \in H. \text{atom } i \# C$   
**shows**  $\text{insert } (x \text{ IN } k) H \vdash A \langle \text{proof} \rangle$

**lemma** *HaddP-Mem-contra*:  
**assumes**  $H \vdash \text{HaddP } x y z H \vdash z \text{ IN } x H \vdash \text{OrdP } x$   
**shows**  $H \vdash A$   
 $\langle \text{proof} \rangle$

**lemma** *exists-HaddP*:  
**assumes**  $H \vdash \text{OrdP } y \text{ atom } j \# (x,y)$   
**shows**  $H \vdash \text{Ex } j (\text{HaddP } x y (\text{Var } j))$   
 $\langle \text{proof} \rangle$

**lemma** *HaddP-Mem-I*:  
**assumes**  $H \vdash \text{HaddP } x y z H \vdash \text{OrdP } x$  **shows**  $H \vdash x \text{ IN } \text{SUCC } z$   
 $\langle \text{proof} \rangle$

## 9.5 A Shifted Sequence

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

**where**  $\llbracket atom\ x \# (x',y,z,f,del,k); atom\ x' \# (y,z,f,del,k); atom\ y \# (z,f,del,k); atom\ z \# (f,del,g,k) \rrbracket \implies$

$ShiftP\ f\ k\ del\ g =$

$All\ z\ (Var\ z\ IN\ g\ IFF$

$(Ex\ x\ (Ex\ x'\ (Ex\ y\ ((Var\ z)\ EQ\ HPair\ (Var\ x')\ (Var\ y)\ AND$

$HaddP\ del\ (Var\ x)\ (Var\ x')\ AND$

$HPair\ (Var\ x)\ (Var\ y)\ IN\ f\ AND\ Var\ x\ IN\ k))))))$

$\langle proof \rangle$

**nominal-termination**  $(eqvt)$

$\langle proof \rangle$

**lemma**  $ShiftP\text{-fresh-iff}$   $[simp]: a \# ShiftP\ f\ k\ del\ g \longleftrightarrow a \# f \wedge a \# k \wedge a \# del \wedge a \# g$

$\langle proof \rangle$

**lemma**  $subst\text{-fm-ShiftP}$   $[simp]:$

$(ShiftP\ f\ k\ del\ g)(i::=u) = ShiftP\ (subst\ i\ u\ f)\ (subst\ i\ u\ k)\ (subst\ i\ u\ del)\ (subst\ i\ u\ g)$

$\langle proof \rangle$

**lemma**  $ShiftP\text{-Zero}: \{\} \vdash ShiftP\ Zero\ k\ d\ Zero$

$\langle proof \rangle$

**lemma**  $ShiftP\text{-Mem1}:$

$\{ShiftP\ f\ k\ del\ g, HPair\ a\ b\ IN\ f, HaddP\ del\ a\ a', a\ IN\ k\} \vdash HPair\ a'\ b\ IN\ g$

$\langle proof \rangle$

**lemma**  $ShiftP\text{-Mem2}:$

**assumes**  $atom\ u \# (f,k,del,a,b)$

**shows**  $\{ShiftP\ f\ k\ del\ g, HPair\ a\ b\ IN\ g\} \vdash Ex\ u\ ((Var\ u)\ IN\ k\ AND\ HaddP\ del\ (Var\ u)\ a\ AND\ HPair\ (Var\ u)\ b\ IN\ f)$

$\langle proof \rangle$

**lemma**  $ShiftP\text{-Mem-D}:$

**assumes**  $H \vdash ShiftP\ f\ k\ del\ g\ H \vdash a\ IN\ g$

$atom\ x \# (x',y,a,f,del,k)\ atom\ x' \# (y,a,f,del,k)\ atom\ y \# (a,f,del,k)$

**shows**  $H \vdash (Ex\ x\ (Ex\ x'\ (Ex\ y\ (a\ EQ\ HPair\ (Var\ x')\ (Var\ y)\ AND$

$HaddP\ del\ (Var\ x)\ (Var\ x')\ AND$

$HPair\ (Var\ x)\ (Var\ y)\ IN\ f\ AND\ Var\ x\ IN\ k))))))$

$(is\ -\vdash\ ?concl)$

$\langle proof \rangle$

**lemma**  $ShiftP\text{-Eats-Eats}:$

$\{ShiftP\ f\ k\ del\ g, HaddP\ del\ a\ a', a\ IN\ k\}$



$\vdash \text{ShiftP } (Eats f (HPair a b)) k del (Eats g (HPair a' b)) \langle proof \rangle$   
**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**  
 $\{ \text{ShiftP } f\ k\ del\ g,$   
 $\text{Neg } (Ex\ u\ (Ex\ u'\ (Ex\ v\ (c\ EQ\ HPair\ (Var\ u)\ (Var\ v)\ AND\ Var\ u\ IN\ k\ AND$   
 $\text{HaddP } del\ (Var\ u)\ (Var\ u'))))\}$   
 $\vdash \text{ShiftP } (Eats\ f\ c)\ k\ del\ g\ \langle proof \rangle$   
**lemma** *exists-ShiftP*:  
**assumes**  $t: atom\ t \# (s, k, del)$   
**shows**  $H \vdash Ex\ t\ (\text{ShiftP } s\ k\ del\ (Var\ t)) \langle proof \rangle$

## 9.6 Union of Two Sets

**nominal-function** *UnionP* ::  $tm \Rightarrow tm \Rightarrow tm \Rightarrow fn$   
**where**  $atom\ i \# (x, y, z) \Longrightarrow UnionP\ x\ y\ z = All\ i\ (Var\ i\ IN\ z\ IFF\ (Var\ i\ IN\ x$   
 $OR\ Var\ i\ IN\ y))$   
 $\langle proof \rangle$

**nominal-termination** (*eqvt*)  
 $\langle proof \rangle$

**lemma** *UnionP-fresh-iff* [*simp*]:  $a \# UnionP\ x\ y\ z \longleftrightarrow a \# x \wedge a \# y \wedge a \# z$   
 $\langle proof \rangle$

**lemma** *subst-fn-UnionP* [*simp*]:  
 $(UnionP\ x\ y\ z)(i::=u) = UnionP\ (subst\ i\ u\ x)\ (subst\ i\ u\ y)\ (subst\ i\ u\ z)$   
 $\langle proof \rangle$

**lemma** *Union-Zero1*:  $H \vdash UnionP\ Zero\ x\ x$   
 $\langle proof \rangle$

**lemma** *Union-Eats*:  $\{ UnionP\ x\ y\ z \} \vdash UnionP\ (Eats\ x\ a)\ y\ (Eats\ z\ a)$   
 $\langle proof \rangle$

**lemma** *exists-Union-lemma*:  
**assumes**  $z: atom\ z \# (i, y)$  **and**  $i: atom\ i \# y$   
**shows**  $\{ \} \vdash Ex\ z\ (UnionP\ (Var\ i)\ y\ (Var\ z))$   
 $\langle proof \rangle$

**lemma** *exists-UnionP*:  
**assumes**  $z: atom\ z \# (x, y)$  **shows**  $H \vdash Ex\ z\ (UnionP\ x\ y\ (Var\ z))$   
 $\langle proof \rangle$

**lemma** *UnionP-Mem1*:  $\{ UnionP\ x\ y\ z, a\ IN\ x \} \vdash a\ IN\ z$   
 $\langle proof \rangle$

**lemma** *UnionP-Mem2*:  $\{ UnionP\ x\ y\ z, a\ IN\ y \} \vdash a\ IN\ z$   
 $\langle proof \rangle$

**lemma** *UnionP-Mem*:  $\{ \text{UnionP } x \ y \ z, a \ \text{IN } z \} \vdash a \ \text{IN } x \ \text{OR } a \ \text{IN } y$   
 $\langle \text{proof} \rangle$

**lemma** *UnionP-Mem-E*:  
**assumes**  $H \vdash \text{UnionP } x \ y \ z$   
**and**  $\text{insert } (a \ \text{IN } x) \ H \vdash A$   
**and**  $\text{insert } (a \ \text{IN } y) \ H \vdash A$   
**shows**  $\text{insert } (a \ \text{IN } z) \ H \vdash A$   
 $\langle \text{proof} \rangle$

## 9.7 Append on Sequences

**nominal-function** *SeqAppendP* ::  $tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow tm \Rightarrow fm$   
**where**  $\llbracket \text{atom } g1 \ \sharp (g2, f1, k1, f2, k2, g); \text{atom } g2 \ \sharp (f1, k1, f2, k2, g) \rrbracket \Longrightarrow$   
 $\text{SeqAppendP } f1 \ k1 \ f2 \ k2 \ g =$   
 $(\text{Ex } g1 \ (\text{Ex } g2 \ (\text{RestrictedP } f1 \ k1 \ (\text{Var } g1) \ \text{AND}$   
 $\text{ShiftP } f2 \ k2 \ k1 \ (\text{Var } g2) \ \text{AND}$   
 $\text{UnionP } (\text{Var } g1) \ (\text{Var } g2) \ g)))$   
 $\langle \text{proof} \rangle$

**nominal-termination** (*eqvt*)  
 $\langle \text{proof} \rangle$

**lemma** *SeqAppendP-fresh-iff* [*simp*]:  
 $a \ \sharp \ \text{SeqAppendP } f1 \ k1 \ f2 \ k2 \ g \longleftrightarrow a \ \sharp \ f1 \ \wedge \ a \ \sharp \ k1 \ \wedge \ a \ \sharp \ f2 \ \wedge \ a \ \sharp \ k2 \ \wedge \ a \ \sharp \ g$   
 $\langle \text{proof} \rangle$

**lemma** *subst-fm-SeqAppendP* [*simp*]:  
 $(\text{SeqAppendP } f1 \ k1 \ f2 \ k2 \ g)(i ::= u) =$   
 $\text{SeqAppendP } (\text{subst } i \ u \ f1) \ (\text{subst } i \ u \ k1) \ (\text{subst } i \ u \ f2) \ (\text{subst } i \ u \ k2) \ (\text{subst } i \ u$   
 $g)$   
 $\langle \text{proof} \rangle$

**lemma** *exists-SeqAppendP*:  
**assumes**  $\text{atom } g \ \sharp (f1, k1, f2, k2)$   
**shows**  $H \vdash \text{Ex } g \ (\text{SeqAppendP } f1 \ k1 \ f2 \ k2 \ (\text{Var } g))$   
 $\langle \text{proof} \rangle$

**lemma** *SeqAppendP-Mem1*:  $\{ \text{SeqAppendP } f1 \ k1 \ f2 \ k2 \ g, \ \text{HPair } x \ y \ \text{IN } f1, \ x \ \text{IN}$   
 $k1 \} \vdash \text{HPair } x \ y \ \text{IN } g$   
 $\langle \text{proof} \rangle$

**lemma** *SeqAppendP-Mem2*:  $\{ \text{SeqAppendP } f1 \ k1 \ f2 \ k2 \ g, \ \text{HaddP } k1 \ x \ x', \ x \ \text{IN } k2,$   
 $\text{HPair } x \ y \ \text{IN } f2 \} \vdash \text{HPair } x' \ y \ \text{IN } g$   
 $\langle \text{proof} \rangle$

**lemma** *SeqAppendP-Mem-E*:  
**assumes**  $H \vdash \text{SeqAppendP } f1 \ k1 \ f2 \ k2 \ g$

**and**  $\text{insert } (\text{HPair } x \ y \ \text{IN } f1) \ (\text{insert } (x \ \text{IN } k1) \ H) \vdash A$   
**and**  $\text{insert } (\text{HPair } (\text{Var } u) \ y \ \text{IN } f2) \ (\text{insert } (\text{HaddP } k1 \ (\text{Var } u) \ x) \ (\text{insert } (\text{Var } u \ \text{IN } k2) \ H)) \vdash A$   
**and**  $u: \text{atom } u \ \# \ (f1, k1, f2, k2, x, y, g, A) \ \forall C \in H. \text{atom } u \ \# \ C$   
**shows**  $\text{insert } (\text{HPair } x \ y \ \text{IN } g) \ H \vdash A \ \langle \text{proof} \rangle$

## 9.8 LstSeqP and SeqAppendP

**lemma** *HDomain-Incl-SeqAppendP*: — The And eliminates the need to prove *cut5*

$\{ \text{SeqAppendP } f1 \ k1 \ f2 \ k2 \ g, \ \text{HDomain-Incl } f1 \ k1 \ \text{AND} \ \text{HDomain-Incl } f2 \ k2, \ \text{HaddP } k1 \ k2 \ k, \ \text{OrdP } k1 \} \vdash \text{HDomain-Incl } g \ k \ \langle \text{proof} \rangle$

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

**lemma** *HFun-Sigma-SeqAppendP*:

$\{ \text{SeqAppendP } f1 \ k1 \ f2 \ k2 \ g, \ \text{HFun-Sigma } f1, \ \text{HFun-Sigma } f2, \ \text{OrdP } k1 \} \vdash \text{HFun-Sigma } g \ \langle \text{proof} \rangle$

**lemma** *LstSeqP-SeqAppendP*:

**assumes**  $H \vdash \text{SeqAppendP } f1 \ (\text{SUCC } k1) \ f2 \ (\text{SUCC } k2) \ g$   
 $H \vdash \text{LstSeqP } f1 \ k1 \ y1 \ H \vdash \text{LstSeqP } f2 \ k2 \ y2 \ H \vdash \text{HaddP } k1 \ k2 \ k$

**shows**  $H \vdash \text{LstSeqP } g \ (\text{SUCC } k) \ y2$

$\langle \text{proof} \rangle$

**lemma** *SeqAppendP-NotInDom*:  $\{ \text{SeqAppendP } f1 \ k1 \ f2 \ k2 \ g, \ \text{HaddP } k1 \ k2 \ k, \ \text{OrdP } k1 \} \vdash \text{NotInDom } k \ g$

$\langle \text{proof} \rangle$

**lemma** *LstSeqP-SeqAppendP-Eats*:

**assumes**  $H \vdash \text{SeqAppendP } f1 \ (\text{SUCC } k1) \ f2 \ (\text{SUCC } k2) \ g$   
 $H \vdash \text{LstSeqP } f1 \ k1 \ y1 \ H \vdash \text{LstSeqP } f2 \ k2 \ y2 \ H \vdash \text{HaddP } k1 \ k2 \ k$

**shows**  $H \vdash \text{LstSeqP } (\text{Eats } g \ (\text{HPair } (\text{SUCC } (\text{SUCC } k)) \ z)) \ (\text{SUCC } (\text{SUCC } k))$

$z$

$\langle \text{proof} \rangle$

## 9.9 Substitution and Abstraction on Terms

### 9.9.1 Atomic cases

**lemma** *SeqStTermP-Var-same*:

**assumes**  $\text{atom } s \ \# \ (k, v, i) \ \text{atom } k \ \# \ (v, i)$

**shows**  $\{ \text{VarP } v \} \vdash \text{Ex } s \ (\text{Ex } k \ (\text{SeqStTermP } v \ i \ v \ i \ (\text{Var } s) \ (\text{Var } k)))$

$\langle \text{proof} \rangle$

**lemma** *SeqStTermP-Var-diff*:

**assumes**  $\text{atom } s \ \# \ (k, v, w, i) \ \text{atom } k \ \# \ (v, w, i)$

**shows**  $\{ \text{VarP } v, \ \text{VarP } w, \ \text{Neg } (v \ \text{EQ } w) \} \vdash \text{Ex } s \ (\text{Ex } k \ (\text{SeqStTermP } v \ i \ w \ w \ (\text{Var } s) \ (\text{Var } k)))$

$\langle \text{proof} \rangle$

**lemma** *SeqStTermP-Zero*:

**assumes**  $atom\ s \# (k, v, i)$   $atom\ k \# (v, i)$

**shows**  $\{VarP\ v\} \vdash Ex\ s\ (Ex\ k\ (SeqStTermP\ v\ i\ Zero\ Zero\ (Var\ s)\ (Var\ k)))$

*<proof>*

**corollary** *SubstTermP-Zero*:  $\{TermP\ t\} \vdash SubstTermP\ \ll Var\ v \gg t\ Zero\ Zero$

*<proof>*

**corollary** *SubstTermP-Var-same*:  $\{VarP\ v, TermP\ t\} \vdash SubstTermP\ v\ t\ v\ t$

*<proof>*

**corollary** *SubstTermP-Var-diff*:  $\{VarP\ v, VarP\ w, Neg\ (v\ EQ\ w), TermP\ t\} \vdash SubstTermP\ v\ t\ w\ w$

*<proof>*

**lemma** *SeqStTermP-Ind*:

**assumes**  $atom\ s \# (k, v, t, i)$   $atom\ k \# (v, t, i)$

**shows**  $\{VarP\ v, IndP\ t\} \vdash Ex\ s\ (Ex\ k\ (SeqStTermP\ v\ i\ t\ t\ (Var\ s)\ (Var\ k)))$

*<proof>*

**corollary** *SubstTermP-Ind*:  $\{VarP\ v, IndP\ w, TermP\ t\} \vdash SubstTermP\ v\ t\ w\ w$

*<proof>*

## 9.9.2 Non-atomic cases

**lemma** *SeqStTermP-Eats*:

**assumes**  $sk: atom\ s \# (k, s1, s2, k1, k2, t1, t2, u1, u2, v, i)$

$atom\ k \# (t1, t2, u1, u2, v, i)$

**shows**  $\{SeqStTermP\ v\ i\ t1\ u1\ s1\ k1, SeqStTermP\ v\ i\ t2\ u2\ s2\ k2\}$

$\vdash Ex\ s\ (Ex\ k\ (SeqStTermP\ v\ i\ (Q-Eats\ t1\ t2)\ (Q-Eats\ u1\ u2)\ (Var\ s)$

$(Var\ k)))$  *<proof>*

**theorem** *SubstTermP-Eats*:

$\{SubstTermP\ v\ i\ t1\ u1, SubstTermP\ v\ i\ t2\ u2\} \vdash SubstTermP\ v\ i\ (Q-Eats\ t1\ t2)\ (Q-Eats\ u1\ u2)$

*<proof>*

## 9.9.3 Substitution over a constant

**lemma** *SeqConstP-lemma*:

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

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

**shows**  $\{SeqConstP\ s\ k\ c\}$

$\vdash c\ EQ\ Zero\ OR$

$Ex\ m\ (Ex\ n\ (Ex\ sm\ (Ex\ sn\ (Var\ m\ IN\ k\ AND\ Var\ n\ IN\ k\ AND$

$SeqConstP\ s\ (Var\ m)\ (Var\ sm)\ AND$

$SeqConstP\ s\ (Var\ n)\ (Var\ sn)\ AND$

$c\ EQ\ Q-Eats\ (Var\ sm)\ (Var\ sn))))))$  *<proof>*

**lemma** *SeqConstP-imp-SubstTermP*:  $\{SeqConstP\ s\ k\ c, TermP\ t\} \vdash SubstTermP\ \ll Var\ w \gg t\ c\ c$  *<proof>*

**theorem** *SubstTermP-Const*:  $\{ConstP\ c, TermP\ t\} \vdash SubstTermP \ll Var\ w \gg t\ c$   
 $c$   
 $\langle proof \rangle$

## 9.10 Substitution on Formulas

### 9.10.1 Membership

**lemma** *SubstAtomicP-Mem*:

$\{SubstTermP\ v\ i\ x\ x', SubstTermP\ v\ i\ y\ y'\} \vdash SubstAtomicP\ v\ i\ (Q-Mem\ x\ y)$   
 $(Q-Mem\ x'\ y')$   
 $\langle proof \rangle$

**lemma** *SeqSubstFormP-Mem*:

**assumes**  $atom\ s\ \# (k, x, y, x', y', v, i)$   $atom\ k\ \# (x, y, x', y', v, i)$   
**shows**  $\{SubstTermP\ v\ i\ x\ x', SubstTermP\ v\ i\ y\ y'\}$   
 $\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q-Mem\ x\ y)\ (Q-Mem\ x'\ y')\ (Var\ s)$   
 $(Var\ k)))$   
 $\langle proof \rangle$

**lemma** *SubstFormP-Mem*:

$\{SubstTermP\ v\ i\ x\ x', SubstTermP\ v\ i\ y\ y'\} \vdash SubstFormP\ v\ i\ (Q-Mem\ x\ y)$   
 $(Q-Mem\ x'\ y')$   
 $\langle proof \rangle$

### 9.10.2 Equality

**lemma** *SubstAtomicP-Eq*:

$\{SubstTermP\ v\ i\ x\ x', SubstTermP\ v\ i\ y\ y'\} \vdash SubstAtomicP\ v\ i\ (Q-Eq\ x\ y)\ (Q-Eq\ x'\ y')$   
 $\langle proof \rangle$

**lemma** *SeqSubstFormP-Eq*:

**assumes**  $sk: atom\ s\ \# (k, x, y, x', y', v, i)$   $atom\ k\ \# (x, y, x', y', v, i)$   
**shows**  $\{SubstTermP\ v\ i\ x\ x', SubstTermP\ v\ i\ y\ y'\}$   
 $\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q-Eq\ x\ y)\ (Q-Eq\ x'\ y')\ (Var\ s)\ (Var\ k)))$   
 $\langle proof \rangle$

**lemma** *SubstFormP-Eq*:

$\{SubstTermP\ v\ i\ x\ x', SubstTermP\ v\ i\ y\ y'\} \vdash SubstFormP\ v\ i\ (Q-Eq\ x\ y)\ (Q-Eq\ x'\ y')$   
 $\langle proof \rangle$

### 9.10.3 Negation

**lemma** *SeqSubstFormP-Neg*:

**assumes**  $atom\ s\ \# (k, s1, k1, x, x', v, i)$   $atom\ k\ \# (s1, k1, x, x', v, i)$   
**shows**  $\{SeqSubstFormP\ v\ i\ x\ x'\ s1\ k1, TermP\ i, VarP\ v\}$

$\vdash Ex\ s\ (Ex\ k\ (SeqSubstFormP\ v\ i\ (Q-Neg\ x)\ (Q-Neg\ x')\ (Var\ s)\ (Var\ k)))$   
 ⟨proof⟩  
**theorem** *SubstFormP-Neg*:  $\{SubstFormP\ v\ i\ x\ x'\} \vdash SubstFormP\ v\ i\ (Q-Neg\ x)$   
 $(Q-Neg\ x')$   
 ⟨proof⟩

### 9.10.4 Disjunction

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

### 9.10.5 Existential

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

## 9.11 Constant Terms

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

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

## 9.12 Proofs

**lemma** *PrfP-inference*:  
**assumes**  $atom\ s\ \# (k, s1, s2, k1, k2, \alpha1, \alpha2, \beta)$   $atom\ k\ \# (s1, s2, k1, k2, \alpha1, \alpha2, \beta)$   
**shows**  $\{PrfP\ s1\ k1\ \alpha1, PrfP\ s2\ k2\ \alpha2, ModPonP\ \alpha1\ \alpha2\ \beta\ OR\ ExistsP\ \alpha1\ \beta$   
 $OR\ SubstP\ \alpha1\ \beta\}$

$\vdash \text{Ex } k \ (\text{Ex } s \ (\text{PrfP} \ (\text{Var } s) \ (\text{Var } k) \ \beta)) \ \langle \text{proof} \rangle$

**corollary** *PfP-inference*:  $\{\text{PfP } \alpha 1, \text{PfP } \alpha 2, \text{ModPonP } \alpha 1 \ \alpha 2 \ \beta \ \text{OR} \ \text{ExistsP } \alpha 1 \ \beta \ \text{OR} \ \text{SubstP } \alpha 1 \ \beta\} \vdash \text{PfP } \beta$   
 $\langle \text{proof} \rangle$

**theorem** *PfP-implies-SubstForm-PfP*:  
**assumes**  $H \vdash \text{PfP } y \ H \vdash \text{SubstFormP } x \ t \ y \ z$   
**shows**  $H \vdash \text{PfP } z$   
 $\langle \text{proof} \rangle$

**theorem** *PfP-implies-ModPon-PfP*:  $\llbracket H \vdash \text{PfP} \ (Q\text{-Imp } x \ y); H \vdash \text{PfP } x \rrbracket \implies H \vdash \text{PfP } y$   
 $\langle \text{proof} \rangle$

**corollary** *PfP-implies-ModPon-PfP-quot*:  $\llbracket H \vdash \text{PfP} \ \langle \alpha \ \text{IMP} \ \beta \rangle; H \vdash \text{PfP} \ \langle \alpha \rangle \rrbracket \implies H \vdash \text{PfP} \ \langle \beta \rangle$   
 $\langle \text{proof} \rangle$

**end**

# Kapitel 10

## Pseudo-Coding: Section 7 Material

```
theory Pseudo-Coding
imports II-Prelims
begin
```

### 10.1 General Lemmas

```
lemma Collect-disj-Un: {f i |i. P i ∨ Q i} = {f i |i. P i} ∪ {f i |i. Q i}
⟨proof⟩
```

```
abbreviation Q-Subset :: tm ⇒ tm ⇒ tm
  where Q-Subset t u ≡ (Q-All (Q-Imp (Q-Mem (Q-Ind Zero) t) (Q-Mem (Q-Ind
Zero) u)))
```

```
lemma NEQ-quot-tm: i ≠ j ⇒ { } ⊢ «Var i» NEQ «Var j»
⟨proof⟩
```

```
lemma EQ-quot-tm-Fls: i ≠ j ⇒ insert («Var i» EQ «Var j») H ⊢ Fls
⟨proof⟩
```

```
lemma perm-commute: a # p ⇒ a' # p ⇒ (a ⇔ a') + p = p + (a ⇔ a')
⟨proof⟩
```

```
lemma perm-self-inverseI: [¬p = q; a # p; a' # p] ⇒ ¬((a ⇔ a') + p) = (a ⇔
a') + q
⟨proof⟩
```

```
lemma fresh-image:
  fixes f :: 'a ⇒ 'b::fs shows finite A ⇒ i # f ' A ↔ (∀ x∈A. i # f x)
⟨proof⟩
```

```
lemma atom-in-atom-image [simp]: atom j ∈ atom ' V ↔ j ∈ V
```



*<proof>*

**lemma** *fresh-star-empty* [*simp*]:  $\{\} \#* bs$   
*<proof>*

**declare** *fresh-star-insert* [*simp*]

**lemma** *fresh-star-finite-insert*:  
**fixes**  $S :: ('a::fs) \text{ set}$  **shows**  $\text{finite } S \implies a \#* \text{insert } x \ S \longleftrightarrow a \#* x \wedge a \#* S$   
*<proof>*

**lemma** *fresh-finite-Diff-single* [*simp*]:  
**fixes**  $V :: \text{name set}$  **shows**  $\text{finite } V \implies a \# (V - \{j\}) \longleftrightarrow (a \# j \longrightarrow a \# V)$   
*<proof>*

**lemma** *fresh-image-atom* [*simp*]:  $\text{finite } A \implies i \# \text{atom } ' A \longleftrightarrow i \# A$   
*<proof>*

**lemma** *atom-fresh-star-atom-set-conv*:  $\llbracket \text{atom } i \# bs; \text{finite } bs \rrbracket \implies bs \#* i$   
*<proof>*

**lemma** *notin-V*:  
**assumes**  $p: \text{atom } i \# p$  **and**  $V: \text{finite } V \text{ atom } ' (p \cdot V) \#* V$   
**shows**  $i \notin V \ i \notin p \cdot V$   
*<proof>*

## 10.2 Simultaneous Substitution

**definition** *ssubst* ::  $tm \Rightarrow \text{name set} \Rightarrow (\text{name} \Rightarrow tm) \Rightarrow tm$   
**where**  $\text{ssubst } t \ V \ F = \text{Finite-Set.fold } (\lambda i. \text{subst } i \ (F \ i)) \ t \ V$

**definition** *make-F* ::  $\text{name set} \Rightarrow \text{perm} \Rightarrow \text{name} \Rightarrow tm$   
**where**  $\text{make-F } Vs \ p \equiv \lambda i. \text{if } i \in Vs \text{ then } \text{Var } (p \cdot i) \text{ else } \text{Var } i$

**lemma** *ssubst-empty* [*simp*]:  $\text{ssubst } t \ \{\} \ F = t$   
*<proof>*

Renaming a finite set of variables. Based on the theorem *at-set-avoiding*

**locale** *quote-perm* =  
**fixes**  $p :: \text{perm}$  **and**  $Vs :: \text{name set}$  **and**  $F :: \text{name} \Rightarrow tm$   
**assumes**  $p: \text{atom } ' (p \cdot Vs) \#* Vs$   
**and**  $\text{pinv}: -p = p$   
**and**  $Vs: \text{finite } Vs$   
**defines**  $F \equiv \text{make-F } Vs \ p$   
**begin**

**lemma** *F-unfold*:  $F \ i = (\text{if } i \in Vs \text{ then } \text{Var } (p \cdot i) \text{ else } \text{Var } i)$   
*<proof>*

**lemma** *finite-V [simp]*:  $V \subseteq Vs \implies \text{finite } V$   
*<proof>*

**lemma** *perm-exits-Vs*:  $i \in Vs \implies (p \cdot i) \notin Vs$   
*<proof>*

**lemma** *atom-fresh-perm*:  $\llbracket x \in Vs; y \in Vs \rrbracket \implies \text{atom } x \# p \cdot y$   
*<proof>*

**lemma** *fresh-pj*:  $\llbracket a \# p; j \in Vs \rrbracket \implies a \# p \cdot j$   
*<proof>*

**lemma** *fresh-Vs*:  $a \# p \implies a \# Vs$   
*<proof>*

**lemma** *fresh-pVs*:  $a \# p \implies a \# p \cdot Vs$   
*<proof>*

**lemma** *assumes*  $V \subseteq Vs$   $a \# p$   
**shows** *fresh-pV [simp]*:  $a \# p \cdot V$  **and** *fresh-V [simp]*:  $a \# V$   
*<proof>*

**lemma** *qp-insert*:  
**fixes**  $i::\text{name}$  **and**  $i'::\text{name}$   
**assumes**  $\text{atom } i \# p$   $\text{atom } i' \# (i,p)$   
**shows** *quote-perm*  $((\text{atom } i \iff \text{atom } i') + p)$   $(\text{insert } i \text{ } Vs)$   
*<proof>*

**lemma** *subst-F-left-commute*:  $\text{subst } x (F x) (\text{subst } y (F y) t) = \text{subst } y (F y)$   
 $(\text{subst } x (F x) t)$   
*<proof>*

**lemma**  
**assumes**  $\text{finite } V$   $i \notin V$   
**shows** *ssubst-insert*:  $\text{ssubst } t (\text{insert } i \text{ } V) F = \text{subst } i (F i) (\text{ssubst } t \text{ } V F)$  (**is**  
*?thesis1*)  
**and** *ssubst-insert2*:  $\text{ssubst } t (\text{insert } i \text{ } V) F = \text{ssubst } (\text{subst } i (F i) t) \text{ } V F$  (**is**  
*?thesis2*)  
*<proof>*

**lemma** *ssubst-insert-if*:  
 $\text{finite } V \implies$   
 $\text{ssubst } t (\text{insert } i \text{ } V) F = (\text{if } i \in V \text{ then } \text{ssubst } t \text{ } V F$   
 $\text{else } \text{subst } i (F i) (\text{ssubst } t \text{ } V F))$   
*<proof>*

**lemma** *ssubst-single [simp]*:  $\text{ssubst } t \{i\} F = \text{subst } i (F i) t$   
*<proof>*

**lemma** *ssubst-Var-if* [simp]:

**assumes** *finite V*

**shows**  $ssubst (Var\ i)\ V\ F = (if\ i \in V\ then\ F\ i\ else\ Var\ i)$

*<proof>*

**lemma** *ssubst-Zero* [simp]:  $finite\ V \implies ssubst\ Zero\ V\ F = Zero$

*<proof>*

**lemma** *ssubst-Eats* [simp]:  $finite\ V \implies ssubst\ (Eats\ t\ u)\ V\ F = Eats\ (ssubst\ t\ V\ F)\ (ssubst\ u\ V\ F)$

*<proof>*

**lemma** *ssubst-SUCC* [simp]:  $finite\ V \implies ssubst\ (SUCC\ t)\ V\ F = SUCC\ (ssubst\ t\ V\ F)$

*<proof>*

**lemma** *ssubst-ORD-OF* [simp]:  $finite\ V \implies ssubst\ (ORD-OF\ n)\ V\ F = ORD-OF\ n$

*<proof>*

**lemma** *ssubst-HPair* [simp]:

$finite\ V \implies ssubst\ (HPair\ t\ u)\ V\ F = HPair\ (ssubst\ t\ V\ F)\ (ssubst\ u\ V\ F)$

*<proof>*

**lemma** *ssubst-HTuple* [simp]:  $finite\ V \implies ssubst\ (HTuple\ n)\ V\ F = (HTuple\ n)$

*<proof>*

**lemma** *ssubst-Subset*:

**assumes** *finite V* **shows**  $ssubst\ [t\ SUBS\ u]\ V\ V\ F = Q-Subset\ (ssubst\ [t]\ V\ V\ F)\ (ssubst\ [u]\ V\ V\ F)$

*<proof>*

**lemma** *fresh-ssubst*:

**assumes**  $finite\ V\ a\ \#\ p \cdot V\ a\ \#\ t$

**shows**  $a\ \#\ ssubst\ t\ V\ F$

*<proof>*

**lemma** *fresh-ssubst'*:

**assumes**  $finite\ V\ atom\ i\ \#\ t\ atom\ (p \cdot i)\ \#\ t$

**shows**  $atom\ i\ \#\ ssubst\ t\ V\ F$

*<proof>*

**lemma** *ssubst-vquot-Ex*:

$\llbracket finite\ V; atom\ i\ \#\ p \cdot V \rrbracket$

$\implies ssubst\ [Ex\ i\ A]\ (insert\ i\ V)\ (insert\ i\ V)\ F = ssubst\ [Ex\ i\ A]\ V\ V\ F$

*<proof>*

**lemma** *ground-ssubst-eq*:  $\llbracket finite\ V; supp\ t = \{\} \rrbracket \implies ssubst\ t\ V\ F = t$

*<proof>*

**lemma** *ssubst-quot-tm* [*simp*]:  
**fixes**  $t::tm$  **shows**  $finite\ V \implies ssubst\ \langle t \rangle\ V\ F = \langle t \rangle$   
 $\langle proof \rangle$

**lemma** *ssubst-quot-fm* [*simp*]:  
**fixes**  $A::fm$  **shows**  $finite\ V \implies ssubst\ \langle A \rangle\ V\ F = \langle A \rangle$   
 $\langle proof \rangle$

**lemma** *atom-in-p-Vs*:  $\llbracket i \in p \cdot V; V \subseteq Vs \rrbracket \implies i \in p \cdot Vs$   
 $\langle proof \rangle$

## 10.3 The Main Theorems of Section 7

**lemma** *SubstTermP-vquot-dbtm*:  
**assumes**  $w: w \in Vs - V$  **and**  $V: V \subseteq Vs\ V' = p \cdot V$   
**and**  $s: supp\ dbtm \subseteq atom\ 'Vs$   
**shows**  
 $insert\ (ConstP\ (F\ w))\ \{ConstP\ (F\ i) \mid i. i \in V\}$   
 $\vdash\ SubstTermP\ \langle Var\ w \rangle\ (F\ w)$   
 $(ssubst\ (vquot-dbtm\ V\ dbtm)\ V\ F)$   
 $(subst\ w\ (F\ w)\ (ssubst\ (vquot-dbtm\ (insert\ w\ V)\ dbtm)\ V\ F))$   
 $\langle proof \rangle$

**lemma** *SubstFormP-vquot-dbfm*:  
**assumes**  $w: w \in Vs - V$  **and**  $V: V \subseteq Vs\ V' = p \cdot V$   
**and**  $s: supp\ dbfm \subseteq atom\ 'Vs$   
**shows**  
 $insert\ (ConstP\ (F\ w))\ \{ConstP\ (F\ i) \mid i. i \in V\}$   
 $\vdash\ SubstFormP\ \langle Var\ w \rangle\ (F\ w)$   
 $(ssubst\ (vquot-dbfm\ V\ dbfm)\ V\ F)$   
 $(subst\ w\ (F\ w)\ (ssubst\ (vquot-dbfm\ (insert\ w\ V)\ dbfm)\ V\ F))$   
 $\langle proof \rangle$

Lemmas 7.5 and 7.6

**lemma** *ssubst-SubstFormP*:  
**fixes**  $A::fm$   
**assumes**  $w: w \in Vs - V$  **and**  $V: V \subseteq Vs\ V' = p \cdot V$   
**and**  $s: supp\ A \subseteq atom\ 'Vs$   
**shows**  
 $insert\ (ConstP\ (F\ w))\ \{ConstP\ (F\ i) \mid i. i \in V\}$   
 $\vdash\ SubstFormP\ \langle Var\ w \rangle\ (F\ w)$   
 $(ssubst\ [A]\ V\ V\ F)$   
 $(ssubst\ [A]\ (insert\ w\ V)\ (insert\ w\ V)\ F)$   
 $\langle proof \rangle$

Theorem 7.3

**theorem** *PfP-implies-PfP-ssubst*:  
**fixes**  $\beta::fm$

**assumes**  $\beta: \{\} \vdash PfP \langle \beta \rangle$   
**and**  $V: V \subseteq Vs$   
**and**  $s: supp \beta \subseteq atom \text{ ' } Vs$   
**shows**  $\{ConstP (F i) \mid i. i \in V\} \vdash PfP (ssubst [\beta] V V F)$   
*<proof>*  
**end**  
**end**

# Kapitel 11

## Quotations of the Free Variables

```
theory Quote
imports Pseudo-Coding
begin
```

### 11.1 Sequence version of the “Special p-Function, F\*”

The definition below describes a relation, not a function. This material relates to Section 8, but omits the ordering of the universe.

```
definition SeqQuote :: hf  $\Rightarrow$  hf  $\Rightarrow$  hf  $\Rightarrow$  hf  $\Rightarrow$  bool
where SeqQuote x x' s k  $\equiv$ 
  BuildSeq2 ( $\lambda y y'. y=0 \wedge y' = 0$ )
  ( $\lambda u u' v v' w w'. u = v \triangleleft w \wedge u' = q\text{-Eats } v' w'$ ) s k x x'
```

#### 11.1.1 Defining the syntax: quantified body

```
nominal-function SeqQuoteP :: tm  $\Rightarrow$  tm  $\Rightarrow$  tm  $\Rightarrow$  tm  $\Rightarrow$  fm
where  $\llbracket$  atom l  $\#$  (s,k,sl,sl',m,n,sm,sm',sn,sn');
  atom sl  $\#$  (s,sl',m,n,sm,sm',sn,sn'); atom sl'  $\#$  (s,m,n,sm,sm',sn,sn');
  atom m  $\#$  (s,n,sm,sm',sn,sn'); atom n  $\#$  (s,sm,sm',sn,sn');
  atom sm  $\#$  (s,sm',sn,sn'); atom sm'  $\#$  (s,sn,sn');
  atom sn  $\#$  (s,sn'); atom sn'  $\#$  s  $\rrbracket \implies$ 
  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
```

$HPair (Var n) (HPair (Var sn) (Var sn')) IN s AND$   
 $Var sl EQ Eats (Var sm) (Var sn) AND$   
 $Var sl' EQ Q-Eats (Var sm') (Var sn'))))))))$

$\langle proof \rangle$

**nominal-termination** (*eqvt*)

$\langle proof \rangle$

**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** *eval-fm-SeqQuoteP* [*simp*]:

$eval-fm e (SeqQuoteP t u s k) \longleftrightarrow SeqQuote \llbracket t \rrbracket e \llbracket u \rrbracket e \llbracket s \rrbracket e \llbracket k \rrbracket e$  (**is** *?thesis2*)

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

$Sigma-fm (SeqQuoteP t u s k)$  (**is** *?thsf*)

**and** *SeqQuoteP-imp-OrdP*:

$\{ SeqQuoteP t u s k \} \vdash OrdP k$  (**is** *?thord*)

**and** *SeqQuoteP-imp-LstSeqP*:

$\{ SeqQuoteP t u s k \} \vdash LstSeqP s k (HPair t u)$  (**is** *?thlstseq*)

$\langle proof \rangle$

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

$(SeqQuoteP t u s k)(j ::= w) =$

$SeqQuoteP (subst j w t) (subst j w u) (subst j w s) (subst j w k)$

$\langle proof \rangle$

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

### 11.1.2 Correctness properties

**lemma** *SeqQuoteP-lemma*:

**fixes** *m::name and sm::name and sm'::name and n::name and sn::name and sn'::name*

**assumes**  $atom m \# (t, u, s, k, n, sm, sm', sn, sn')$   $atom n \# (t, u, s, k, sm, sm', sn, sn')$

$atom sm \# (t, u, s, k, sm', sn, sn')$   $atom sm' \# (t, u, s, k, sn, sn')$

$atom sn \# (t, u, s, k, sn')$   $atom sn' \# (t, u, s, k)$

**shows**  $\{ SeqQuoteP t u s k \}$

$\vdash (t EQ Zero AND u EQ Zero) OR$

$Ex m (Ex n (Ex sm (Ex sm' (Ex sn (Ex sn' (Var m IN k AND Var n IN k AND$

$SeqQuoteP (Var sm) (Var sm') s (Var m) AND$

$SeqQuoteP (Var sn) (Var sn') s (Var n) AND$

$t EQ Eats (Var sm) (Var sn) AND$

$u EQ Q-Eats (Var sm') (Var sn'))))))))$

$\langle proof \rangle$

## 11.2 The “special function” itself

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

where  $Quote\ x\ x' \equiv \exists s\ k. SeqQuote\ x\ x'\ s\ k$

### 11.2.1 Defining the syntax

**nominal-function**  $QuoteP :: tm \Rightarrow tm \Rightarrow fm$   
 where  $\llbracket atom\ s\ \# (t,u,k); atom\ k\ \# (t,u) \rrbracket \Longrightarrow$   
 $QuoteP\ t\ u = Ex\ s\ (Ex\ k\ (SeqQuoteP\ t\ u\ (Var\ s)\ (Var\ k)))$   
 $\langle proof \rangle$

**nominal-termination** ( $eqvt$ )  
 $\langle proof \rangle$

**lemma**  
 shows  $QuoteP\text{-fresh-iff}$  [ $simp$ ]:  $a\ \# QuoteP\ t\ u \longleftrightarrow a\ \# t \wedge a\ \# u$  (**is**  $?thesis1$ )  
 and  $eval\text{-fm-QuoteP}$  [ $simp$ ]:  $eval\text{-fm}\ e\ (QuoteP\ t\ u) \longleftrightarrow Quote\ \llbracket t \rrbracket e\ \llbracket u \rrbracket e$  (**is**  $?thesis2$ )  
 and  $QuoteP\text{-sf}$  [ $iff$ ]:  $Sigma\text{-fm}\ (QuoteP\ t\ u)$  (**is**  $?thsf$ )  
 $\langle proof \rangle$

**lemma**  $QuoteP\text{-subst}$  [ $simp$ ]:  
 $(QuoteP\ t\ u)(j::=w) = QuoteP\ (subst\ j\ w\ t)\ (subst\ j\ w\ u)$   
 $\langle proof \rangle$

**declare**  $QuoteP.simps$  [ $simp\ del$ ]

### 11.2.2 Correctness properties

**lemma**  $Quote\ 0$ :  $Quote\ 0\ 0$   
 $\langle proof \rangle$

**lemma**  $QuoteP\text{-Zero}$ :  $\{\} \vdash QuoteP\ Zero\ Zero$   
 $\langle proof \rangle$

**lemma**  $SeqQuoteP\text{-Eats}$ :  
 assumes  $atom\ s\ \# (k,s1,s2,k1,k2,t1,t2,u1,u2)$   $atom\ k\ \# (s1,s2,k1,k2,t1,t2,u1,u2)$   
 shows  $\{SeqQuoteP\ t1\ u1\ s1\ k1, SeqQuoteP\ t2\ u2\ s2\ k2\} \vdash$   
 $Ex\ s\ (Ex\ k\ (SeqQuoteP\ (Eats\ t1\ t2)\ (Q\text{-Eats}\ u1\ u2)\ (Var\ s)\ (Var\ k)))$   
 $\langle proof \rangle$

**lemma**  $QuoteP\text{-Eats}$ :  $\{QuoteP\ t1\ u1, QuoteP\ t2\ u2\} \vdash QuoteP\ (Eats\ t1\ t2)$   
 $(Q\text{-Eats}\ u1\ u2)$   
 $\langle proof \rangle$

**lemma**  $exists\text{-QuoteP}$ :  
 assumes  $j: atom\ j\ \#\ x$  shows  $\{\} \vdash Ex\ j\ (QuoteP\ x\ (Var\ j))$   
 $\langle proof \rangle$

**lemma**  $QuoteP\text{-imp-ConstP}$ :  $\{QuoteP\ x\ y\} \vdash ConstP\ y$



*<proof>*

**lemma** *SeqQuoteP-imp-QuoteP*:  $\{SeqQuoteP\ t\ u\ s\ k\} \vdash QuoteP\ t\ u$   
*<proof>*

**lemmas** *QuoteP-I = SeqQuoteP-imp-QuoteP [THEN cut1]*

## 11.3 The Operator *quote-all*

### 11.3.1 Definition and basic properties

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

**lemma** *quote-all-empty [simp]*:  $quote-all\ p\ \{\} = \{\}$   
*<proof>*

**lemma** *quote-all-insert [simp]*:  
 $quote-all\ p\ (insert\ i\ V) = insert\ (QuoteP\ (Var\ i)\ (Var\ (p \cdot i)))\ (quote-all\ p\ V)$   
*<proof>*

**lemma** *finite-quote-all [simp]*:  $finite\ V \Longrightarrow finite\ (quote-all\ p\ V)$   
*<proof>*

**lemma** *fresh-quote-all [simp]*:  $finite\ V \Longrightarrow i \# quote-all\ p\ V \longleftrightarrow i \# V \wedge i \# p \cdot V$   
*<proof>*

**lemma** *fresh-quote-all-mem*:  $\llbracket A \in quote-all\ p\ V; finite\ V; i \# V; i \# p \cdot V \rrbracket \Longrightarrow i \# A$   
*<proof>*

**lemma** *quote-all-perm-eq*:  
**assumes**  $finite\ V\ atom\ i \# (p, V)\ atom\ i' \# (p, V)$   
**shows**  $quote-all\ ((atom\ i \rightleftharpoons atom\ i') + p)\ V = quote-all\ p\ V$   
*<proof>*

### 11.3.2 Transferring theorems to the level of derivability

**context** *quote-perm*  
**begin**

**lemma** *QuoteP-imp-ConstP-F-hyps*:  
**assumes**  $Us \subseteq Vs\ \{ConstP\ (F\ i) \mid i. i \in Us\} \vdash A$  **shows**  $quote-all\ p\ Us \vdash A$   
*<proof>*

Lemma 8.3

**theorem** *quote-all-PfP-ssubst*:  
**assumes**  $\beta: \{\} \vdash \beta$   
**and**  $V: V \subseteq Vs$

**and**  $s: \text{supp } \beta \subseteq \text{atom } ' Vs$   
**shows**  $\text{quote-all } p \ V \vdash \text{PfP } (\text{ssubst } [\beta] \ V \ V \ F)$   
 $\langle \text{proof} \rangle$

Lemma 8.4

**corollary** *quote-all-MonPon-PfP-ssubst:*

**assumes**  $A: \{\} \vdash \alpha \ \text{IMP} \ \beta$   
**and**  $V: V \subseteq Vs$   
**and**  $s: \text{supp } \alpha \subseteq \text{atom } ' Vs \ \text{supp } \beta \subseteq \text{atom } ' Vs$   
**shows**  $\text{quote-all } p \ V \vdash \text{PfP } (\text{ssubst } [\alpha] \ V \ V \ F) \ \text{IMP} \ \text{PfP } (\text{ssubst } [\beta] \ V \ V \ F)$   
 $\langle \text{proof} \rangle$

Lemma 8.4b

**corollary** *quote-all-MonPon2-PfP-ssubst:*

**assumes**  $A: \{\} \vdash \alpha 1 \ \text{IMP} \ \alpha 2 \ \text{IMP} \ \beta$   
**and**  $V: V \subseteq Vs$   
**and**  $s: \text{supp } \alpha 1 \subseteq \text{atom } ' Vs \ \text{supp } \alpha 2 \subseteq \text{atom } ' Vs \ \text{supp } \beta \subseteq \text{atom } ' Vs$   
**shows**  $\text{quote-all } p \ V \vdash \text{PfP } (\text{ssubst } [\alpha 1] \ V \ V \ F) \ \text{IMP} \ \text{PfP } (\text{ssubst } [\alpha 2] \ V \ V \ F) \ \text{IMP} \ \text{PfP } (\text{ssubst } [\beta] \ V \ V \ F)$   
 $\langle \text{proof} \rangle$

**lemma** *quote-all-Disj-I1-PfP-ssubst:*

**assumes**  $V \subseteq Vs \ \text{supp } \alpha \subseteq \text{atom } ' Vs \ \text{supp } \beta \subseteq \text{atom } ' Vs$   
**and** *prems:*  $H \vdash \text{PfP } (\text{ssubst } [\alpha] \ V \ V \ F) \ \text{quote-all } p \ V \subseteq H$   
**shows**  $H \vdash \text{PfP } (\text{ssubst } [\alpha \ \text{OR} \ \beta] \ V \ V \ F)$   
 $\langle \text{proof} \rangle$

**lemma** *quote-all-Disj-I2-PfP-ssubst:*

**assumes**  $V \subseteq Vs \ \text{supp } \alpha \subseteq \text{atom } ' Vs \ \text{supp } \beta \subseteq \text{atom } ' Vs$   
**and** *prems:*  $H \vdash \text{PfP } (\text{ssubst } [\beta] \ V \ V \ F) \ \text{quote-all } p \ V \subseteq H$   
**shows**  $H \vdash \text{PfP } (\text{ssubst } [\alpha \ \text{OR} \ \beta] \ V \ V \ F)$   
 $\langle \text{proof} \rangle$

**lemma** *quote-all-Conj-I-PfP-ssubst:*

**assumes**  $V \subseteq Vs \ \text{supp } \alpha \subseteq \text{atom } ' Vs \ \text{supp } \beta \subseteq \text{atom } ' Vs$   
**and** *prems:*  $H \vdash \text{PfP } (\text{ssubst } [\alpha] \ V \ V \ F) \ H \vdash \text{PfP } (\text{ssubst } [\beta] \ V \ V \ F) \ \text{quote-all } p \ V \subseteq H$   
**shows**  $H \vdash \text{PfP } (\text{ssubst } [\alpha \ \text{AND} \ \beta] \ V \ V \ F)$   
 $\langle \text{proof} \rangle$

**lemma** *quote-all-Contra-PfP-ssubst:*

**assumes**  $V \subseteq Vs \ \text{supp } \alpha \subseteq \text{atom } ' Vs$   
**shows**  $\text{quote-all } p \ V$   
 $\vdash \text{PfP } (\text{ssubst } [\alpha] \ V \ V \ F) \ \text{IMP} \ \text{PfP } (\text{ssubst } [\text{Neg } \alpha] \ V \ V \ F) \ \text{IMP} \ \text{PfP}$   
 $(\text{ssubst } [\text{Fls}] \ V \ V \ F)$   
 $\langle \text{proof} \rangle$

**lemma** *fresh-ssubst-dbtm:*  $\llbracket \text{atom } i \ \# \ p \cdot V; \ V \subseteq Vs \rrbracket \implies \text{atom } i \ \# \ \text{ssubst } (v\text{quot-dbtm } V \ t) \ V \ F$

$\langle \text{proof} \rangle$

**lemma** *fresh-ssubst-dbfm*:  $\llbracket \text{atom } i \# p \cdot V; V \subseteq Vs \rrbracket \implies \text{atom } i \# \text{ssubst } (\text{vquote-dbfm } V A) V F$   
 $\langle \text{proof} \rangle$

**lemma** *fresh-ssubst-fm*:

**fixes**  $A::fm$  **shows**  $\llbracket \text{atom } i \# p \cdot V; V \subseteq Vs \rrbracket \implies \text{atom } i \# \text{ssubst } (\llbracket A \rrbracket V) V F$   
 $\langle \text{proof} \rangle$

**end**

## 11.4 Star Property. Equality and Membership: Lemmas 9.3 and 9.4

**lemma** *SeqQuoteP-Mem-imp-QMem-and-Subset*:

**assumes**  $\text{atom } i \# (j, j', i', si, ki, sj, kj)$   $\text{atom } i' \# (j, j', si, ki, sj, kj)$   
 $\text{atom } j \# (j', si, ki, sj, kj)$   $\text{atom } j' \# (si, ki, sj, kj)$   
 $\text{atom } si \# (ki, sj, kj)$   $\text{atom } sj \# (ki, kj)$

**shows**  $\{ \text{SeqQuoteP } (Var\ i) (Var\ i') (Var\ si)\ ki, \text{SeqQuoteP } (Var\ j) (Var\ j') (Var\ sj)\ kj \}$

$\vdash (Var\ i\ IN\ Var\ j\ IMP\ Pfp\ (Q\ Mem\ (Var\ i')\ (Var\ j')))\ AND$   
 $(Var\ i\ SUBS\ Var\ j\ IMP\ Pfp\ (Q\ Subset\ (Var\ i')\ (Var\ j')))$

$\langle \text{proof} \rangle$

**lemma**

**assumes**  $\text{atom } i \# (j, j', i')$   $\text{atom } i' \# (j, j')$   $\text{atom } j \# (j')$

**shows** *QuoteP-Mem-imp-QMem*:

$\{ \text{QuoteP } (Var\ i) (Var\ i'), \text{QuoteP } (Var\ j) (Var\ j'), Var\ i\ IN\ Var\ j \}$   
 $\vdash Pfp\ (Q\ Mem\ (Var\ i')\ (Var\ j'))$  **(is ?thesis1)**

**and** *QuoteP-Mem-imp-QSubset*:

$\{ \text{QuoteP } (Var\ i) (Var\ i'), \text{QuoteP } (Var\ j) (Var\ j'), Var\ i\ SUBS\ Var\ j \}$   
 $\vdash Pfp\ (Q\ Subset\ (Var\ i')\ (Var\ j'))$  **(is ?thesis2)**

$\langle \text{proof} \rangle$

## 11.5 Star Property. Universal Quantifier: Lemma 9.7

**lemma** (**in** *quote-perm*) *SeqQuoteP-Mem-imp-All2*:

**assumes**  $IH: \text{insert } (\text{QuoteP } (Var\ i) (Var\ i')) (\text{quote-all } p\ Vs)$   
 $\vdash \alpha\ IMP\ Pfp\ (\text{ssubst } \llbracket \alpha \rrbracket (\text{insert } i\ Vs)\ (\text{insert } i\ Vs)\ Fi)$

**and**  $sp: \text{supp } \alpha - \{ \text{atom } i \} \subseteq \text{atom } ' Vs$

**and**  $j: j \in Vs$  **and**  $j': p \cdot j = j'$

**and**  $pi: pi = (\text{atom } i \rightleftharpoons \text{atom } i') + p$

**and**  $Fi: Fi = \text{make-F } (\text{insert } i\ Vs)\ pi$

**and**  $\text{atoms}: \text{atom } i \# (j, j', s, k, p)$   $\text{atom } i' \# (i, p, \alpha)$

$atom\ j\ \#(j',s,k,\alpha)\ atom\ j'\ \#(s,k,\alpha)$   
 $atom\ s\ \#(k,\alpha)\ atom\ k\ \#(\alpha,p)$

**shows**  $insert\ (SeqQuoteP\ (Var\ j)\ (Var\ j')\ (Var\ s)\ (Var\ k))\ (quote\text{-}all\ p\ (Vs\ -\{j\}))$   
 $\vdash\ All2\ i\ (Var\ j)\ \alpha\ IMP\ Pfp\ (ssubst\ [All2\ i\ (Var\ j)\ \alpha]\ Vs\ Vs\ F)$   
 $\langle proof \rangle$

**lemma** (in *quote-perm*) *quote-all-Mem-imp-All2*:  
**assumes**  $IH: insert\ (QuoteP\ (Var\ i)\ (Var\ i'))\ (quote\text{-}all\ p\ Vs)$   
 $\vdash\ \alpha\ IMP\ Pfp\ (ssubst\ [\alpha]\ (insert\ i\ Vs)\ (insert\ i\ Vs)\ Fi)$   
**and**  $supp\ (All2\ i\ (Var\ j)\ \alpha) \subseteq atom\ 'Vs$   
**and**  $j: atom\ j\ \#(i,\alpha)$  **and**  $i: atom\ i\ \#p$  **and**  $i': atom\ i'\ \#(i,p,\alpha)$   
**and**  $pi: pi = (atom\ i \rightleftharpoons atom\ i') + p$   
**and**  $Fi: Fi = make\text{-}F\ (insert\ i\ Vs)\ pi$   
**shows**  $insert\ (All2\ i\ (Var\ j)\ \alpha)\ (quote\text{-}all\ p\ Vs) \vdash Pfp\ (ssubst\ [All2\ i\ (Var\ j)\ \alpha]\ Vs\ Vs\ F)$   
 $\langle proof \rangle$

## 11.6 The Derivability Condition, Theorem 9.1

**lemma** *SpecI*:  $H \vdash A\ IMP\ Ex\ i\ A$   
 $\langle proof \rangle$

**lemma** *star*:  
**fixes**  $p :: perm$  **and**  $F :: name \Rightarrow tm$   
**assumes**  $C: ss\text{-}fm\ \alpha$   
**and**  $p: atom\ ' (p \cdot V) \#* V -p = p$   
**and**  $V: finite\ V\ supp\ \alpha \subseteq atom\ ' V$   
**and**  $F: F = make\text{-}F\ V\ p$   
**shows**  $insert\ \alpha\ (quote\text{-}all\ p\ V) \vdash Pfp\ (ssubst\ [\alpha]\ V\ V\ F)$   
 $\langle proof \rangle$

**theorem** *Provability*:  
**assumes**  $Sigma\text{-}fm\ \alpha\ ground\text{-}fm\ \alpha$   
**shows**  $\{\alpha\} \vdash Pfp\ \langle \alpha \rangle$   
 $\langle proof \rangle$

**end**

## Kapitel 12

# Gdel's Second Incompleteness Theorem

```
theory Goedel-II
imports Goedel-I Quote
begin
```

The connection between *Quote* and *HR* (for interest only).

```
lemma Quote-q-Eats [intro]:
  Quote y y'  $\implies$  Quote z z'  $\implies$  Quote (y  $\triangleleft$  z) (q-Eats y' z')
  <proof>
```

```
lemma Quote-q-Succ [intro]: Quote y y'  $\implies$  Quote (succ y) (q-Succ y')
  <proof>
```

```
lemma HR-imp-eq-H: HR x z  $\implies$  z =  $\llbracket HF\ x \rrbracket e$ 
  <proof>
```

```
lemma HR-Ord-D: HR x y  $\implies$  Ord x  $\implies$  WR x y
  <proof>
```

```
lemma WR-Quote: WR (ord-of i) y  $\implies$  Quote (ord-of i) y
  <proof>
```

```
lemma [simp]:  $\langle \langle 0, 0, 0 \rangle, x, y \rangle = q\text{-Eats } x\ y$ 
  <proof>
```

```
lemma HR-imp-Quote: coding-hf x  $\implies$  HR x y  $\implies$  Quote x y
  <proof>
```

```
interpretation qp0: quote-perm 0 {} make-F {} 0
  <proof>
```

```
lemma MonPon-PfP-implies-PfP:
```

$\llbracket \{\} \vdash \alpha \text{ IMP } \beta; \text{ground-fm } \alpha; \text{ground-fm } \beta \rrbracket \implies \{PfP \ll \alpha \gg\} \vdash PfP \ll \beta \gg$   
*<proof>*

**lemma** *PfP-quot-contr*:  $\text{ground-fm } \alpha \implies \{\} \vdash PfP \ll \alpha \gg \text{ IMP } PfP \ll \text{Neg } \alpha \gg \text{ IMP } PfP \ll \text{Fls} \gg$   
*<proof>*

Gdel's second incompleteness theorem: Our theory cannot prove its own consistency.

**theorem** *Goedel-II*:  $\neg \{\} \vdash \text{Neg } (PfP \ll \text{Fls} \gg)$   
*<proof>*

**end**

# Literaturverzeichnis

- [1] G. S. Boolos. *The Logic of Provability*. Cambridge University Press, 1993.
- [2] S. Feferman et al., editors. *Kurt Gödel: Collected Works*, volume I. Oxford University Press, 1986.
- [3] S. Świerczkowski. Finite sets and Gödel's incompleteness theorems. *Dissertationes Mathematicae*, 422:1–58, 2003. <http://journals.impan.gov.pl/dm/Inf/422-0-1.html>.