Appendix

A LF

The syntactic classes of the type theory LF are the following:

- Kinds $K \ ::= \ type \mid \Pi x : A.K$
- Families of types $A \ ::= \ a \mid \Pi x : A.B \mid \lambda x : A.B \mid A\ M$
- Objects $M \ ::= \ c \mid x \mid \lambda x : A.M \mid M\ N$
- Signatures $\Sigma \ ::= \ < \ > \mid \Sigma, a : K \mid \Sigma, c : A$
- Contexts $\Gamma \ ::= \ < \ > \mid \Gamma, x : A$

The sequents which can be derived in the type theory LF are the following:

- $\vdash \ \Sigma\ Sign$ which means that Σ is a valid signature.
- $\vdash_\Sigma \ \Gamma\ Ctxt$ which means Γ is a valid context in Σ.
- $\Gamma \ \vdash_\Sigma \ K\ Kind$ which means that K is a valid kind in context Γ with signature Σ.
- $\Gamma \ \vdash_\Sigma \ A \ : \ K$ which asserts that A has kind K in context Γ with signature Σ.
- $\Gamma \ \vdash_\Sigma \ M \ : \ A$ which asserts that M has type A in context Γ with signature Σ.

The rules can be divided in signature validity rules, context validity rules, kind formation rules, family rules and object rules:

- **Signature validity rules:**

$$\frac{}{< \ > \ Sign} \qquad (Empty - Sig)$$

$$\frac{\vdash \ \Sigma\ Sign \quad \vdash_\Sigma \ K\ kind}{\vdash \ \Sigma, a : K\ Sign}\ a \notin dom(\Sigma) \qquad (AddKindSign)$$

$$\frac{\vdash \ \Sigma\ Sign \quad \vdash_\Sigma \ A \ : \ type}{\vdash \ \Sigma, c : A\ Sign} \qquad (AddobjSign)$$

- **Context validity rules:**

$$\frac{\vdash \ \Sigma \ Sign}{\vdash_\Sigma < \ > \ Ctxt} \quad (B-Empty-Ctxt)$$

$$\frac{\vdash_\Sigma \ \Gamma \ Ctxt \quad \Gamma \ \vdash_\Sigma \ A : Type \quad \Gamma \ \vdash_\Sigma \ x : A}{\vdash_\Sigma \ \Gamma, x : A \ Ctxt} \quad (B-Type-Ctxt)$$

- **Kind formation rules:**

$$\frac{\vdash_\Sigma \ \Gamma \ Ctxt}{\Gamma \ \vdash_\Sigma \ \mathbf{Type} \, Kind} \quad (B-Type-Kind)$$

$$\frac{\Gamma, x : A \ \vdash_\Sigma \ K \ Kind}{\Gamma \ \vdash_\Sigma \ \Pi x : A.K \ Kind} \quad (B-Pi-Kind)$$

- **Family rules:**

$$\frac{\vdash_\Sigma \Gamma \ Ctxt \quad c : K \in \Sigma}{\Gamma \vdash_\Sigma c : K} \quad (B-Const-Fam)$$

$$\frac{\Gamma \ \vdash \ A : Type \quad \Gamma, x : A \ \vdash_\Sigma \ B : Type}{\Gamma \ \vdash_\Sigma \ \Pi x : A.B : Type} \quad (B-Pi-Fam)$$

$$\frac{\Gamma, x : A \ \vdash_\Sigma \ B : K}{\Gamma \ \vdash_\Sigma \ \lambda x : A.B : \Pi x : A.K} \quad (B-Abs-Fam)$$

$$\frac{\Gamma \ \vdash_\Sigma \ A \ : \ \Pi x : B.K \quad \Gamma \ \vdash_\Sigma \ M : B}{\Gamma \ \vdash_\Sigma \ A \ M : K\{M/x\}} \quad (B-App-Fam)$$

$$\frac{\Gamma \ \vdash_\Sigma A : K \quad \Gamma \ \vdash_\Sigma ; K' Kind \quad \Gamma \ \vdash_\Sigma \ K \equiv K'}{\Gamma \ \vdash_\Sigma A : K'} \quad (B-Conv-Fam)$$

- **Object rules:**

$$\frac{\vdash_\Sigma \Gamma \ Ctxt \quad c : A \in \Sigma}{\Gamma \vdash_\Sigma c : A} \quad (B-Const-Obj)$$

$$\frac{\vdash_\Sigma \Gamma \ Ctxt \quad x : A \in \Gamma}{\Gamma \vdash_\Sigma x : A} \quad (B-Var-Obj)$$

$$\frac{\Gamma, x : A \ \vdash_\Sigma \ M : B}{\Gamma \ \vdash_\Sigma \ \lambda x : A.M : \Pi x : A.B} \quad (B-Abs-Obj)$$

$$\frac{\Gamma \ \vdash_\Sigma \ M \ : \ \Pi x : A.B \quad \Gamma \ \vdash_\Sigma \ N : B}{\Gamma \ \vdash_\Sigma \ M \ N : B\{N/x\}} \quad (B-App-Obj)$$

$$\frac{\Gamma\ \vdash_\Sigma M : A \quad \Gamma\ \vdash_\Sigma ; A' Type \quad \Gamma\ \vdash_\Sigma\ A \equiv A'}{\Gamma\ \vdash_\Sigma M : A'} \qquad (B - Conv - Obj)$$

where the definitional equality $\equiv$ at all the three levels (objects, families and kinds) is defined as the reflexive,symmetric and transitive closure of the following parallel reduction relation:

- **Parallel Reduction:**

$$\frac{M\ \rightarrow\ M' \quad N\ \rightarrow\ N'}{(\lambda x : A.M)\ N\ \rightarrow\ \{N'/x\}\ M')} \qquad R - Beta - Obj$$

$$\frac{B\ \rightarrow ; B' \quad N\ \rightarrow\ N'}{(\lambda x : B.M)\ N\ \rightarrow\ \{N'/x\}\ B')} \qquad R - Beta - Fam$$

$$\frac{M\ \rightarrow\ M' \quad N\ \rightarrow\ N';}{M\ N\ \rightarrow\ M'\ N'} \qquad R - App - Obj$$

$$\frac{A\ \rightarrow\ A' \quad N\ \rightarrow\ N'}{A\ N\ \rightarrow\ A'\ N'} \qquad R - App - Fam$$

$$\frac{A\ \rightarrow\ A' \quad M\ \rightarrow\ M'}{\lambda x : A.M\ \rightarrow \lambda x : A'.M'} \qquad R - Abs - Obj$$

$$\frac{A\ \rightarrow\ A' \quad B\ \rightarrow\ B'}{\lambda x : A.B\ \rightarrow \lambda x : A'.B'} \qquad R - Abs - Fam$$

$$\frac{A\ \rightarrow\ A' \quad B\ \rightarrow\ B'}{\Pi x : A.B\ \rightarrow \Pi x : A'.B'} \qquad R - Pi - Fam$$

$$\frac{A\ \rightarrow\ A' \quad K\ \rightarrow\ K'}{\Pi x : A.K\ \rightarrow \Pi x : A'.K'} \qquad R - Pi - Kind$$

B ECC

The terms of the type theory ECC are the following:

- Prop, $Type_i\ i \in \omega$ which are also referred as universes.

- $x \mid \Pi x : M.N \mid \lambda x : M.N \mid M\ N \mid \Sigma x : M.N \mid\ < M, N >_A$ $\mid \pi_1(M) \mid \pi_2(M)$ where M,N and A are terms

The formal definition of this type theory has just got the following two sequents:

- $\vdash\ \Gamma\ valid$ which means that the context Γ is valid.
- $\Gamma \vdash M : A$ which asserts that M inhabits A in context Γ.

and it is formally defined by the following rules:

$$\frac{}{\vdash <\ >\ \textbf{valid}} \quad (Empty - Ctxt)$$

$$\frac{\Gamma \vdash\ A\ :\ Type_j \quad x\ \notin\ FV(\Gamma)}{\vdash\ \Gamma, x : A\ \textbf{valid}} \quad (Ctxt - Form)$$

$$\frac{\Gamma, x : A, \Gamma'\ \vdash\ x : A}{\vdash\ \Gamma, x : A, \Gamma'\ \textbf{valid}} \quad (Ass)$$

$$\frac{\Gamma\ \vdash Prop\ :\ Type_0}{\vdash\ \Gamma valid} \quad (AxC)$$

$$\frac{\Gamma\ \vdash Type_i\ :\ Type_{i+1}}{\vdash\ \Gamma valid} \quad (AxT)$$

$$\frac{\Gamma, x : A\ \vdash\ P\ :\ Prop}{\Gamma\ \vdash\ \Pi x : A : P\ :\ Prop} \quad (\Pi_1)$$

$$\frac{\Gamma\ \vdash\ A : Type_j \quad \Gamma, x : A\ \vdash\ B\ :\ Type_j}{\Gamma\ \vdash\ \Pi x : A : B\ :\ Type_j} \quad (\Pi_2)$$

$$\frac{\Gamma, x : A\ \vdash_\Sigma\ M : B}{\Gamma\ \vdash\ \lambda x : A.M : \Pi x : A.B} \quad (\lambda)$$

$$\frac{\Gamma\ \vdash_\Sigma\ M\ :\ \Pi x : A.B \quad \Gamma\ \vdash_\Sigma\ N : B}{\Gamma\ \vdash_\Sigma\ M\ N : B\{N/x\}} \quad (app)$$

$$\frac{\Gamma\ \vdash\ A : Type_j \quad \Gamma, x : A\ \vdash\ B\ :\ Type_j}{\Gamma\ \vdash\ \Sigma x : A : B\ :\ Type_j} \quad (\Sigma)$$

$$\frac{\Gamma\ \vdash\ M\ :\ A \quad \Gamma\ \vdash\ N\ :\ B\ \{\ M\ /\ x\} \quad \Gamma, x : A\ \vdash\ B\ :\ Type_j}{\Gamma\ \vdash < M, N >_{\Sigma x:A.B} : \Sigma x : A.B} \quad (pair)$$

$$\frac{\Gamma\ \vdash\ M : \Sigma x : A.B}{\Gamma\ \vdash\ \pi_1(\ M\ ) : A} \quad (\pi_1) \frac{\Gamma\ \vdash\ M : \Sigma x : A.B}{\Gamma\ \vdash\ \pi_2(\ M\ ) : B\{\pi_1(\ M\ )\ /\ x\}} \quad (\pi_2)$$

$$\frac{\Gamma\ \vdash M : A \quad \Gamma\ \vdash\ A' : Type_j}{\Gamma\ \vdash M : A'}\ A\ \leq\ A' \quad (Conv)$$

where the cumulativity relation $\leq$ is defined as

$$\leq = \bigcup_{i \in \omega} \leq_i$$

where $\leq_i$ is inductively defined as follows:

- $A \leq_0 B$ if and only if one of the following holds:
  - $A \leq_i B$ or
  - $A \equiv Prop$ and $B \equiv Type_j$ for some $j \in \omega$ or
  - $A \equiv Type_j$ and $B \equiv Type_k$ for some $j < k$.
- $A \leq_{i+1} B$ if and only if one of the following holds:
  - $A \equiv_i B$ or
  - $A \equiv \Pi x : A_1.A_2$ and $B \equiv \Pi x : B_1.B_2$ for some $A_1 \equiv B_1$ and $A_2 \leq_i B_2$ or
  - $A \equiv \Sigma x : A_1.A_2$ and $B \equiv \Sigma x : B_1.B_2$ for some $A_1 \leq_i B_1$ and $A_2 \leq_i B_2$

and the conversion relation $\equiv$ is defined in a similar way as in LF using a similar parallel reduction relation with additionally the following specific rules for Σ-types:

$$\frac{M_1 \rightarrow M_1'}{\pi_1(< M_1, M_2 >_A) \rightarrow M_1'} \quad R - Pi1$$

$$\frac{M_2 \rightarrow M_2'}{\pi_2(< M_1, M_2 >_A) \rightarrow M_2'} \quad R - Pi2$$

C Martin Löf Logical Framework

The sequents which are used in the definition of the logical framework are the following:

- A kind which means that we know the elements of A and an equality on the elements of A which must be decidable.
- $A = B$ which means that A and B have the same elements with the same equality on elements.
- $M : A.$ which asserts that M is of kind A.
- $M = N : A.$ which asserts that elements M and N are of kind A and they are equal by the equality of A.
- $\vdash \Gamma$ which means that the context Γ is valid.

The rules which define this logical framework is divided in context and substitution rules, rules which define the Kind type, equality rules (general ones and equality typing rules) and rules which define dependent product kinds. The rules are the following:

- **Context and substitution rules:**
  - **Context rules:**

$$\frac{}{\vdash ()}\ (Emp) \qquad \frac{\Gamma \vdash A\ kind \quad x \notin dom(\Gamma)}{\vdash \Gamma, x : A}\ (Weak) \qquad \frac{\vdash \Gamma_0, x : A, \Gamma_1}{\Gamma_0, x : A, \Gamma_1 \vdash x : A}\ (Ass)$$

  - **Substitution rules:**

$$\frac{\vdash\ \Gamma_0, z : C, \Gamma_1 \quad \Gamma_0\ \vdash\ P : C}{\vdash\ \Gamma_0, \Gamma_1\{P/z\}} \qquad (Ctxt\ Subst)$$

$$\frac{\Gamma_0, z : C, \Gamma_1\ \vdash\ A\ kind \quad \Gamma_0\ \vdash\ P : C}{\Gamma_0, \Gamma_1\{P/z\}\ \vdash\ A\{P/z\}\ kind} \qquad (Kind\ Subst)$$

$$\frac{\Gamma_0, z : C, \Gamma_1\ \vdash\ M : A \quad \Gamma_0\ \vdash\ P : C}{\Gamma_0, \Gamma_1\{P/z\}\ \vdash\ M\{P/z\} : A\{P/z\}} \qquad (Term\ Subst)$$

$$\frac{\Gamma_0, z : C, \Gamma_1\ \vdash\ A\ kind \quad \Gamma_0\ \vdash\ P\ =\ Q : C}{\Gamma_0, \Gamma_1\{P/z\}\ \vdash\ A\{P/z\}\ =\ A\{Q/z\}} \qquad (Kind\ Subst - Eq)$$

$$\frac{\Gamma_0, z : C, \Gamma_1\ \vdash\ M : A \quad \Gamma_0\ \vdash\ P\ =\ Q : C}{\Gamma_0, \Gamma_1\{P/z\}\ \vdash\ M\{P/z\}\ =\ M\{Q/z\} : A\{P/z\}} \qquad (Term\ Subst - Eq)$$

$$\frac{\Gamma_0, z : C, \Gamma_1\ \vdash\ A\ =\ B \quad \Gamma_0\ \vdash\ P : C}{\Gamma_0, \Gamma_1\{P/z\}\ \vdash\ A\{P/z\}\ =\ B\{P/z\}} \qquad (Kind - eq\ Subst)$$

$$\frac{\Gamma_0, z : C, \Gamma_1\ \vdash\ M\ =\ N : A \quad \Gamma_0\ \vdash\ P : C}{\Gamma_0, \Gamma_1\{P/z\}\ \vdash\ M\{P/z\}\ =\ N\{P/z\} : A\{P/z\}} \qquad (Term - eq\ Subst)$$

- **Kind type rules:**

$$\frac{\vdash\ \Gamma}{\Gamma\ \vdash\ Type\ kind} \quad (Type)$$

$$\frac{\Gamma\ \vdash\ A : Type}{\Gamma\ \vdash\ El(A)\ Kind}\ (El) \qquad \frac{\Gamma\ \vdash\ A\ =\ B : Type}{\Gamma\ \vdash\ El(A)\ =\ El(B)}\ (El - Eq)$$

- **Equality rules:**
  - **General equality rules:**

$$\frac{\Gamma \vdash A\ kind}{\Gamma \vdash A = A}\ (KRefl) \qquad \frac{\Gamma \vdash A = B}{\Gamma \vdash B = A}\ (Ksym)$$

$$\frac{\Gamma \vdash A = B \quad \Gamma \vdash B = C}{\Gamma \vdash A = C}\ (KTrans)$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M = M : A}\ (Refl) \qquad \frac{\Gamma \vdash N = M : A}{\Gamma \vdash M = N : A}\ (Sym)$$

$$\frac{\Gamma \vdash M = N : A \quad \Gamma \vdash N = P : A}{\Gamma \vdash M = P : A}\ (Trans)$$

  - **Equality typing rules**

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A = B}{\Gamma \vdash M : B}\ (= T) \qquad \frac{\Gamma \vdash M = N : A \quad \Gamma \vdash A = B}{\Gamma \vdash M = N : B}\ (= R)$$

- **Dependent product kind rules:**

$$\frac{\Gamma, x : A_1 \vdash A_2\ kind}{\Gamma \vdash (x : A_1)A_2\ kind}\ (\Pi)$$

$$\frac{\Gamma \vdash A_1 = B_1 \quad \Gamma, x : A_1 \vdash A_2 = B_2}{\Gamma \vdash (x : A_1)A_2 = (x : B_1)B_2}\ (\Pi - Eq)$$

$$\frac{\Gamma, x : A_1 \vdash M : A_2}{\Gamma \vdash [x : A_1]M : (x : A_1)A_2}\ (\lambda)$$

$$\frac{\Gamma \vdash A = B \quad \Gamma, x : A \vdash M = N : C}{\Gamma \vdash [x : A]M = [x : B]N : (x : A) : C}\ (\lambda - Eq)$$

$$\frac{\Gamma \vdash M : (x : A_1)A_2 \quad \Gamma \vdash N : A_1}{\Gamma \vdash M\ N : A_2\{A_1/x\}}\ (appl)$$

$$\frac{\Gamma \vdash M = M' : (x : A_1)A_2 \quad \Gamma \vdash N = N' : A_1}{\Gamma \vdash M\ N = M'\ N' : A_2\{A_1/x\}}\ (appl - Eq)$$

$$\frac{\Gamma, x : A \vdash M : B \quad \Gamma \vdash N : A}{\Gamma \vdash [x : A]M\ N = M\ \{N/x\} : B\{N/x\}}\ (\beta)$$

$$\frac{\Gamma \vdash [x : A](M\ x) = M : [x : A]B}{\Gamma \vdash M : [x : A]B}\ (\eta)$$

D UTT

In this appendix we present the formal definition of UTT as in [?] First we will present the formal definition of the universe of propositions, and then we will give the formal definition of the schema which will be used to automatically generate safe induction principles and computational rules from a definition of an inductive type by a set of constructors. Finally, we will instantiate the general schema with some basic inductive relations and we will explain the notation that we will use in the rest of the chapters. See [?] for a formal definition of the hierarchy of types of UTT in the Martin Löf logical framework which has a similar structure as the predicative hierarchy of universes of ECC.

D.1 Universe of propositions

Definition D.1 *The impredicative universe of propositions is defined by the following constant declarations:*

$$Prop : \quad Type$$

$$Prf : \quad (Prop)Type$$

$$\forall : \quad (A : Type)((A)Prop)Prop$$

$$\Lambda : \quad (A : Type)(P : (A)Prop)$$
$$((x : A)Prf(P(x)))Prf(\forall(A, P))$$

$$E_\forall : \quad (A : Type)(P : (A)Prop)$$
$$(R : (Prf(\forall(A, P)))Prop)$$
$$((g : (x : A)Prf(P(x)))Prf(R(\Lambda(A, P, g))))$$
$$(z : Prf(\forall(A, P)))Prf(R(z))$$

with the equality rule

$$E_\forall(A, P, R, f, \Lambda(A, P, g)) \ = \ f(g) : Prf(R(\Lambda(A, P, g)))$$

The kind of the constant $\forall$ can be seen as the encoding of the formation rule of propositions ($\Pi 1$) of ECC,and the kind of the constant Λ can be seen as the encoding of the introduction rule (λ) of ECC when B is the universe **Prop**. The encoding of the elimination rule (*appl*) of ECC when B is the universe

Prop can be encoded using the constant $E_\forall$ as follows:

$$appl(A, P, M, N) =$$

$$E_\forall(A, P, [G : Prf(\forall(A, P))]P(N), [g : (x : A)Prf(P(x))]g(N), M) \ :$$

$$(A : Type)(P : (A)Prop)(Prf(\forall(A, P)))(N : A)Prf(P(N))$$

D.2 Inductive types and inductive relations

D.2.1 Inductive types

Definition D.2 *We say that a kind A is a small kind if it can be inductively defined by the following rules:*

- $A \equiv El(M)$
- $A \equiv (x : A_1)A_2$, *where A_1 and A_2 are small kinds.*

Definition D.3 *Let Γ be a valid context and X be a variable. A kind Φ is a strictly positive operator in Γ with respect to X (denoted by $POS_{\Gamma;X}(\Phi)$) if it can be inductively defined by the following rules:*

- $\Phi \ \equiv \ X.$
- $\Phi \ \equiv \ (x : A)\Phi_0$, *where A is a small kind and $POS_{\Gamma;X}(\Phi_0)$.*

Definition D.4 *Let Γ be a valid context and X be a variable. A kind Θ is an inductive schema in Γ with respect to X (denoted by $SCH_{\Gamma;X}(\Theta)$), if it can be inductively defined by the following rules:*

- $\Theta \ \equiv \ X$
- $\Theta \ \equiv \ (x : A)\Theta_0$, *where A is a small kind and $SCH_{\Gamma;X}(\Theta_0)$*
- $\Theta \ \equiv \ (\Phi)\Theta_0$ *where $POS_{\Gamma;X}(\Phi)$ and $SCH_{\Gamma;X}(\Theta_0)$*

Definition D.5 *A finite sequence of kinds $\Theta_1, \ldots, \Theta_n$ is a schema family in Γ with respect to X (denoted by $SCH_{\Gamma;X}(\Theta_1, \ldots, \Theta_n)$), if $SCH_{\Gamma;X}(\Theta_i)$ for $1 \leq i \leq n$.*

Definition D.6 *Assume that $POS_{\Gamma;X}(\Phi)$ where $\Phi \equiv (x_1 : A_1)\ldots(x_n : A_n)X$, and assume that $\Gamma \ \vdash \ A : Type$, $\Gamma \ \vdash \ C : (A)Type$ and $\Gamma \ \vdash \ z : \Phi(A)$. The kind $\Phi^0[A, C, z]$ is defined as follows:*

$$\Phi^0[A, C, z] \ = \quad (x_1 : A_1)\ldots(x_n : A_n)C(z(x_1, \ldots, x_n))$$

Definition D.7 *Assume that* $SCH_{\Gamma;X}(\Theta)$ *where* $\Theta \equiv (x_1 : M_1)\dots(x_n : M_n)X$, *and assume that* $\Gamma \vdash A : Type$, $\Gamma \vdash C : (A)Type$ *and* $\Gamma \vdash z : \Phi(A)$ *and let* $M_{i_1}, \dots, M_{i_k}$ *be all the strictly positive operators of* $M_1, \dots, M_n$. *The kind* $\Theta^0[A, C, z]$ *is defined as follows:*

$$\Theta^0[A, C, z] = (x_1 : M_1(A))\dots(x_n : M_n(A))$$

$$(\Phi^o_{i_1}[A, C, x_{i_1}])\dots(\Phi^o_{i_k}[A, C, x_{i_k}])C(z(x_1, \dots, x_n))$$

Definition D.8 *Assume that* $POS_{\Gamma;X}(\Phi)$ *where* $\Phi \equiv (x_1 : A_1)\dots(x_n : A_n)X$, *and assume that* $\Gamma \vdash A : Type$, $\Gamma \vdash C : (A)Type$, $\Gamma \vdash z : \Phi(A)$, *and let* $\Gamma \vdash f : (x : A)C\ x$. *The kind* $\Phi^l[A, C, f, z]$ *is defined as follows:*

$$\Phi^1[A, C, f, z] = (x_1 : A_1)\dots(x_n : A_n)f(z(x_1, \dots, x_n))$$

Definition D.9 *Given a valid context* Γ, *a schema family in* Γ *with respect to* X *(*$SCH_{\Gamma,X}(\hat{\Theta})$ *where* $\hat{\Theta} = (\Theta_1, \dots, \Theta_n)$*), the elements* $\mathcal{M}[\hat{\Theta}], i_i[\hat{\Theta}]$ *for every* $i \in [1..n]$ *and* $E[\hat{\Theta}]$ *have the following kinds:*

$$\mathcal{M}[\hat{\Theta}] : Type$$

$$i_i[\hat{\Theta}] : \Theta_i(\mathcal{M}[\hat{\Theta}]) \ \ (1 \leq i \leq n)$$

$$E[\hat{\Theta}] : (C : (\mathcal{M}[\hat{\Theta}]))Type)$$

$$(f_1 : \Theta^o_1[\mathcal{M}[\hat{\Theta}], C, i_1[\hat{\Theta}]])$$

$$\dots$$

$$(f_n : \Theta^o_n[\mathcal{M}[\hat{\Theta}], C, i_n[\hat{\Theta}]])$$

$$(z : \mathcal{M}[\hat{\Theta}])C(z)$$

and the following associated equality rules:

$$E[\hat{\Theta}](C, \hat{f}, i_i[\hat{\Theta}](\hat{a})) =$$

$$f_i(\hat{a}, \phi^1_{i1}[\mathcal{M}[\hat{\Theta}], C, E[\hat{\Theta}](C, \hat{f}), a_{i1}], \dots, \phi^1_{ik}[\mathcal{M}[\hat{\Theta}], C, E[\hat{\Theta}](C, \hat{f}), a_{ik}]) : C(i_i[\hat{\Theta}](\hat{a}))$$

for every $i \in [1..n]$, *where* $\hat{f}$ *stands for* $f_1, \dots, f_n$, $\hat{a}$ *for* $a_1, \dots, a_n$ *and* $\phi^1_{i1}, \dots, \phi^1_{ik}$ *is the sequence of all strictly positive operators in* Γ *with respect to* X *in* Θ_i.

Now we give the concrete instantiations of the kinds of the elements $\mathcal{M}[\hat{\Theta}], i_i[\hat{\Theta}]$ for all $i \in [1..n]$ and $E[\hat{\Theta}]$ and their associated equality rules of different schema families $\hat{\Theta} = (\Theta_1, \dots, \Theta_n)$. For readability reasons, we will rename appropiately the name of these elements for every different schema family.

Definition D.10 *Let Boolsch be the schema family Boolsch* $= [X, X]$. *The elements* $\mathcal{M}[Boolsch], i_1[Boolsch], i_2[Boolsch]$ *and E[Boolsch] renamed respectively to Bool,true,false and E[Bool] are of the following kinds:*

$$\begin{aligned}
Bool &: \quad Type \\
true &: \quad Bool \\
false &: \quad Bool \\
E[Bool] &: \quad (C : (Bool)Type) \\
& \quad (bct : C\ true) \\
& \quad (bcf : C\ false) \\
& \quad (b : Bool)(C\ b)
\end{aligned}$$

The associated equality rules are instantiated as follows:

$$E[Bool]\ C\ bcf\ bct\ false \ = \ bcf$$

$$E[Bool]\ C\ bcf\ bct\ true \ = \ bct$$

Definition D.11 *Let Natsch be the schema family Natsch* $= [X, X(X)]$. *The elements* $\mathcal{M}[Natsch], i_1[Natsch], i_2[Natsch]$ *and E[Natsch] renamed respectively to Nat,zero,succ and E[Nat] are of the following kinds:*

$$\begin{aligned}
Nat &: \quad Type \\
zero &: \quad Nat \\
succ &: \quad (Nat)Nat \\
E[Nat] &: \quad (C : (Nat)Type) \\
& \quad (bcN : C\ zero) \\
& \quad (gcN : (n : Nat)(C\ n)(C\ (succ\ n))) \\
& \quad (n : Nat)(C\ n)
\end{aligned}$$

The associated equality rules are instantiated as follows:

$$E[Nat]\ C\ bcn\ gcn\ zero \ = \ bcn$$

$$E[Nat]\ C\ bcn\ gcn\ (succ\ n) \ = \ gcn\ n\ (E[Nat]\ C\ bcn\ gcn\ n)$$

Definition D.12 *Assume that* $A : Type$. *Let* $Listsch$ *be the schema family* $Listsch = [(A : Type)X, (A : Type)(A)(X)X]$. *The elements* $\mathcal{M}[Listsch]$, $i_1[Listsch], i_2[Listsch]$ *and* $E[Listsch]$ *renamed respectively to* $(A : Type)List\ A, nil, cons$ *and* $E[(A : Type)List\ A]$ *are of the following kind:*

$$List\ A : \quad Type$$

$$nil\ : \quad (A : Type)(List\ A)$$

$$cons\ : \quad (A : Type)(A)(List\ A)(List\ A)$$

$$E[List\ A] : \quad (C : (List\ A)Type)$$

$$(bcL : (A : Type)(C\ (nil\ A)))$$

$$(gcL : (A : Type)(a : A)(l : ListA)(C\ l)(C\ (cons\ A\ a\ l))$$

$$(l : List\ A)(C\ l)$$

The associated equality rules for a given type $A : Type$ *are instantiated as follows:*

$$E[List\ A]\ C\ bcl\ gcl\ (nil\ A)\ =\ (bcl\ A)$$

$$E[List\ A]\ C\ bcl\ gcl\ (cons\ A\ a\ l)\ =\ gcl\ A\ a\ l\ (E[List\ A]\ C\ bcl\ gcl\ l)$$

Definition D.13 *Assume that* $A : Type$ *and* $B : Type$. *Let* $Pairsch$ *be the schema family* $Pairsch = [(A)(B)X]$. *The elements* $\mathcal{M}[Pairsch], i[Pairsch]$ *and* $E[Pairsch]$ *renamed respectively to* $Pair\ A\ B, mkpair$ *and* $E[Pair\ A\ B]$ *are of the following kind:*

$$Pair\ A\ B : Type$$

$$mkpair\ :\ (A : Type)(B : Type)(A)(B)(Pair\ A\ B)$$

$$E[(Pair\ A\ B)]\ :$$

$$(C : (Pair\ A\ B))Type)$$

$$(bcP : (a : A)(b : B)(C\ (mkpair\ A\ B\ a\ b)))$$

$$(p : Pair\ A\ B)(C\ p)$$

The associated equality rules for a given type $A : Type, B : Type$, *are instantiated as follows:*

$$E[Pair\ A\ B]\ C\ bcp\ (mkpair\ A\ B\ a\ b)\ =\ bcp\ a\ b$$

D.2.2 Inductive relations

Since we haven't found a formal way to instantiate the schemata for inductive types to define inductive relations of the form

$$\mathcal{R} \equiv \Pi x_1 : A_1 . \dots . \Pi x_n : A_n . Prop$$

where $A_1 : Type, \dots , A_n : Type$, we give a new schemata to define these relations.

Definition D.14 *Let Γ be a valid context and X be a variable. A kind $\Phi[X]$ is a strictly positive relational operator in Γ with respect to X (denoted by $POSR_{\Gamma;X}(\Phi[X])$) if it can be inductively defined by the following rules:*

- $\Phi[X] \equiv Prf(X(a_1, \dots , a_n))$.
- $\Phi[X] \equiv (x : A)\Phi_0$, *where A is a small kind and $POS_{\Gamma;X}(\Phi_0)$.*

where for any $j \in [1..n]$, $a_j : A_j\{a_1 \,/\, x_1\} \dots \{a_{j-1} \,/\, x_{j-1}\}$

Definition D.15 *Let Γ be a valid context and X be a variable. A kind Θ is an inductive schema in Γ with respect to X (denoted by $SCHR_{\Gamma;X}(\Theta)$), if it can be inductively defined by the following rules:*

- $\Theta \equiv Prf(X(a_1, \dots , a_n))$
- $\Theta \equiv (x : A)\Theta_0$, *where A is a small kind and $SCHR_{\Gamma;X}(\Theta_0)$*
- $\Theta \equiv (\Phi)\Theta_0$ *where $POSR_{\Gamma;X}(\Phi)$ and $SCHR_{\Gamma;X}(\Theta_0)$*

where for any $j \in [1..n]$, $a_j : A_j\{a_1 \,/\, x_1\} \dots \{a_{j-1} \,/\, x_{j-1}\}$

Definition D.16 *Assume that $SCHR_{\Gamma;X}(\Theta_i)$ where $\Theta \equiv (x_{1i} : M_1) \dots (x_{mi} : M_{mi})Prf(X(a_1, \dots , a_n))$, and assume that $\Gamma \vdash C : (y_1 : A_1) \dots (y_n : A_n)Prop$ and let $M_{i_1}, \dots , M_{i_k}$ be all the strictly positive relational operators of $M_{1i}, \dots , M_{mi}$. The kind $\Theta^0[\mathcal{R}, C]$ is defined as follows:*

$$\Theta_i^0[\mathcal{R}, C] = (x_1 : M_1(\mathcal{R})) \dots (x_{mi} : M_{mi}(\mathcal{R}))$$

$$(\Phi_{i_1}[C]) \dots (\Phi_{i_k}[C])Prf(C(a_1, \dots , a_n))$$

where for any $j \in [1..n]$, $a_j : A_j\{a_1 \,/\, x_1\} \dots \{a_{j-1} \,/\, x_{j-1}\}$

Definition D.17 *Assume that $POSR_{\Gamma;X}(\Phi)$ where $\Phi \equiv (x_1 : A_1) \dots (x_m : A_m)X(a_1, \dots , a_n)$ where for any $j \in [1..n]$, $a_j : A_j\{a_1 \,/\, x_1\} \dots \{a_{j-1} \,/\, x_{j-1}\}$. Assume also that $\Gamma \vdash C : (y_1 : A_1) \dots (y_n : A_n)Prop$ and $\Gamma \vdash F : (y_1 : A_1) \dots (y_n : A_n)Prf(C(y_1, \dots , y_n))$. The kind $\Phi^1[X, C]$ is defined as follows:*

$$\Phi^1[C, F] = (x_1 : A_1) \dots (x_m : A_m)F(a_1, \dots , a_n)$$

Definition D.18 *Given a valid context* Γ*, a schema family in* Γ *with respect to* X *(*$SCHR_{\Gamma;X}(\hat{\Theta})$ *where* $\hat{\Theta} = (\Theta_1, \ldots, \Theta_n)$*), the elements* $\mathcal{R}[\hat{\Theta}], i_i[\hat{\Theta}]$ *for all* $i \in [1..n]$ *and* $E[\hat{\Theta}]$ *have the following kinds:*

$$\mathcal{R}[\hat{\Theta}] : \quad (x_1 : A_1) \ldots (x_n : A_n) Prop$$

$$i_i[\hat{\Theta}] : \quad \Theta_i(\mathcal{R}[\hat{\Theta}]) \ (1 \leq i \leq n)$$

$$E[\hat{\Theta}] : \quad (C : (x_1 : A_1) \ldots (x_n : A_n) : Prop)$$

$$(f_1 : \Theta_1^0[\mathcal{R}, C])$$

$$\ldots$$

$$(f_n : \Theta_n^0[\mathcal{R}, C])$$

$$(x_1 : A_1) \ldots (x_n : A_n) C(x_1, \ldots, x_n)$$

As an example, we give an instantiation of the inductive relations schemata of a schema family which could be used to encode a fragment of the propositional calculus including the following rules:

$$\frac{\Gamma \Rightarrow \phi_1 \wedge \phi_2}{\Gamma \Rightarrow \phi_1} \ (\wedge El) \qquad \frac{\Gamma \Rightarrow \phi_1 \wedge \phi_2}{\Gamma \Rightarrow \phi_2} \ (\wedge Er)$$

$$\frac{\Gamma \Rightarrow \phi_1 \quad \Gamma \Rightarrow \phi_2}{\Gamma \Rightarrow \phi_1 \wedge \phi_2} \ (\wedge I)$$

$$\frac{\Gamma \cup \phi \Rightarrow \phi'}{\Gamma \Rightarrow \phi \supset \phi'} \ (\supset \ i) \qquad \frac{\Gamma \Rightarrow \phi \supset \phi' \quad \Gamma \Rightarrow \phi}{\Gamma \Rightarrow \phi'} \ (\supset \ e)$$

We assume predefined the inductive type $Propos$ which for this fragment would just contain the two connectives described above (conjuction ($and : Propos \rightarrow Propos \rightarrow Propos$) and implication ($implies : Propos \rightarrow Propos \rightarrow Propos$) and the type Env defined as $List\ Propos$ using the inductive type list described above.

Definition D.19 *Let* $\mathcal{R}[PCsch] : (env : List\ Propos)(form : Propos)Prop$

where PCsch is the following schema family:

$$PCsch = \quad (env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$

$$(pr : Prf(X\ (env, and\ \phi_1\ \phi_2)))$$

$$(Prf(X\ (env, \phi_1)))$$

$$(env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$

$$(pr : Prf(X\ (env, and\ \phi_1\ \phi_2)))$$

$$(Prf(X\ (env, \phi_2)))$$

$$(env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$

$$(pr_1 : Prf(X\ (env, \phi_1)))(pr_2 : Prf(X\ (env, \phi_2)))$$

$$(Prf(X\ (env, and\ \phi_1\ \phi_2)))$$

$$(env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$

$$(pr : Prf(X\ (cons\ Propos\ \phi_1\ env, \phi_2)))$$

$$Prf(X\ (env, implies\ \phi_1\ \phi_2))$$

$$(env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$

$$(pr_1 : Prf(X\ (env, implies\ \phi_1\ \phi_2)))(pr_2 : Prf(X\ (env, \phi_1)))$$

$$(Prf(X\ (env, \phi_2)))$$

The elements $\mathcal{R}[PCsch], i_1[PCsch], i_2[PCsch], i_3[PCsch], i_4[PCsch], i_5[PCsch]$ *and* $E[PCsch]$ *renamed respectively to* $PC, andl, andr, andi, impli, imple$ *and*

E[PC] are defined as follows:

$$PC : (env : List\ Propos)(form : Propos)Prop$$

$$andl : (env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$
$$(pr : Prf(PC\ (env, and\ \phi_1\ \phi_2)))$$
$$(Prf(PC\ (env, \phi_1))))$$

$$andr : (env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$
$$(pr : Prf(PC\ (env, and\ \phi_1\ \phi_2)))$$
$$(Prf(PC\ (env, \phi_2))))$$

$$andi : (env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$
$$(pr_1 : Prf(PC\ (env, \phi_1)))(pr_2 : Prf(PC\ (env, \phi_2)))$$
$$(Prf(PC\ (env, and\ \phi_1\ \phi_2))))$$

$$impli : (env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$
$$(pr : Prf(PC\ (cons\ Propos\ \phi_1\ env, \phi_2)))$$
$$Prf(PC\ (env, implies\ \phi_1\ \phi_2)))$$

$$imple : (env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$
$$(pr_1 : Prf(PC\ (env, implies\ \phi_1\ \phi_2)))(pr_2 : Prf(PC\ (env, \phi_1)))$$
$$(Prf(PC\ (env, \phi_2))))$$

$$E[PC] : (C : (env : List\ Propos)(form : Propos)Prop)$$

$$(andlc : (env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$
$$(pr : Prf(PC\ (env, and\ \phi_1\ \phi_2)))$$
$$(pr_1 : Prf(C\ (env, and\ \phi_1\ \phi_2)))$$
$$(Prf(C\ (env, \phi_1))))$$

$$(andrc : (env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$
$$(pr : Prf(PC\ (env, and\ \phi_1\ \phi_2)))$$
$$(pr' : Prf(C\ (env, and\ \phi_1\ \phi_2)))$$
$$(Prf(C\ (env, \phi_2))))$$

$$(andi : (env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$
$$(pr_1 : Prf(PC\ (env, \phi_1)))(pr_2 : Prf(PC\ (env, \phi_2)))$$
$$(pr'_1 : Prf(C\ (env, \phi_1)))(pr'_2 : Prf(C\ (env, \phi_2)))$$
$$(Prf(C\ (env, and\ \phi_1\ \phi_2))))$$

$$(impli : (env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$
$$(pr : Prf(PC\ (cons\ Propos\ \phi_1\ env, \phi_2)))$$
$$(pr' : Prf(C\ (cons\ Propos\ \phi_1\ env, \phi_2)))$$
$$Prf(PC\ (env, implies\ \phi_1\ \phi_2)))$$

$$(imple : (env : List\ Propos)(\phi_1 : Propos)(\phi_2 : Propos)$$
$$(pr_1 : Prf(PC\ (env, implies\ \phi_1\ \phi_2)))(pr_2 : Prf(PC\ (env, \phi_1)))$$
$$(pr'_1 : Prf(C\ (env, implies\ \phi_1\ \phi_2)))(pr'_2 : Prf(C\ (env, \phi_1)))$$
$$(Prf(C\ (env, \phi_2))))$$

$$(env : List\ Propos)(form : Propos)Prf(C\ (env, form))$$

E Basic definitions and predefined functions in UTT

In this section, first we will encode several logical operators in the universe of propositions like for example conjunction, disjunction and existential quantification and then we will redefine the inductive types and inductive relations which appear as examples in previous sections using a more readable notation which we will use in some chapters of the thesis. For these cases we will explicit the primitive recursive operators and the induction principles associated to these inductive types which we will normally assume predefined. We will also define some functions associated to the inductive types in a notation similar to classical functional programming languages. Finally we will give an example of mutually recursive inductive data types making explicit also their induction principles and their computational rules following a similar schemata as the one presented for inductive types in the previous sections.

Definition E.1 *The encoding of the logical operators* **false**, **true**, $=, \vee, \wedge, \exists$ *is as follows:*

$$false =_{def} \forall P : \mathbf{Prop}.P$$

$$true =_{def} \forall P : \mathbf{Prop}.P \supset P$$

$$t =_{\tau} r =_{def} \forall P : [\tau].P\ t \supset P\ r$$

$$\phi \supset \phi' =_{def} \forall p : \phi.\phi'$$

$$\neg\, \phi =_{def} \phi\ \supset\ false$$

$$\phi \wedge \phi' =_{def} \forall P : \mathbf{Prop}.(\phi \supset \phi' \supset P) \supset P$$

$$\phi \vee \phi' =_{def} \forall P : \mathbf{Prop}.(\phi \supset P) \supset (\phi' \supset P) \supset P$$

$$\exists x : \tau.\phi =_{def} \forall P : \mathbf{Prop}.((\forall x : \tau.\phi) \supset P) \supset P$$

Notation: *The equality* $=_{\tau}$ *will be referred to as leibniz equality.*

Proposition E.2 *The following rules are admissible in ECC and UTT:*

$$\frac{}{\{\} \vdash tpterm\ :\ \mathbf{true}}\ (T) \qquad \frac{}{\Gamma \vdash fpterm\ :\ \mathbf{false} \supset \phi}\ (F)$$

$$\frac{\Gamma \vdash\ p\ :\ \phi_1 \wedge \phi_2}{\Gamma \vdash\ andlpterm\ :\ \phi_1}\ (\wedge El) \qquad \frac{\Gamma \vdash\ p\ :\ \phi_1 \wedge \phi_2}{\Gamma \vdash\ andrpterm\ :\ \phi_2}\ (\wedge Er)$$

$$\frac{\Gamma \vdash\ p_1 : \phi_1 \quad \Gamma \vdash p_2 : \phi_2}{\Gamma \vdash\ andpterm\ :\ \phi_1 \wedge \phi_2}\ (\wedge I)$$

$$\frac{\Gamma \vdash p : \phi_1}{\Gamma \vdash orlpterm : \phi_1 \vee \phi_2} \quad (\vee Il) \qquad \frac{\Gamma \vdash p : \phi_2}{\Gamma \vdash orrpterm : \phi_1 \vee \phi_2} \quad (\vee Ir)$$

$$\frac{\Gamma \vdash or : \phi_1 \vee \phi_2 \quad \Gamma \vdash or_1 : \phi_1 \supset \psi \quad \Gamma \vdash or_2 : \phi_2 \supset \psi}{\Gamma \vdash orelpterm : \psi} \quad (\vee E)$$

$$\frac{\Gamma \vdash t : \tau \quad \Gamma \vdash p : \phi\{t/x\}}{\Gamma \vdash exipterm : \exists x : \tau.\phi} \quad (\exists I)$$

$$\frac{\Gamma \vdash p : \exists x : \tau.\phi \quad \Gamma \cup \{p_1 : \phi\} \vdash p_2 : \psi}{\Gamma \vdash exelpterm : \psi} \quad (\exists E)$$

$$\frac{\Gamma \cup \{p_1 : \phi\} \Rightarrow_X p_2 : \mathbf{false}}{\Gamma \Rightarrow_X npterm : \neg\phi} \quad (\neg I)$$

for some terms tpterm, fpterm , andlpterm, andrpterm, andpterm, orlpterm, orrpterm, orelpterm, exipterm, exelpterm, npterm

Proof E.3 *See [**?**] for the proofs and the form of the terms.*

E.1 Functions on Bool type

Definition E.4 *The inductive type Bool* $: Type_0$ *is defined by the following set of constructors:*

$$true : Bool$$

$$false : Bool$$

The induction principle Ind(Bool) which we will use to reason about propositions of type Bool $\rightarrow$ *Prop is the following:*

$$\Pi P : Bool \rightarrow Prop.(P\ true) \supset (P\ false) \supset (\forall b : Bool.P\ b)$$

and the primitive recursion principle Primrec Bool with arity

$$Primrec\ Bool : T \rightarrow T \rightarrow Bool \rightarrow T$$

for any type $T : Type_0$ *has the following computational rules:*

$$Primrec\ Bool\ bct\ bcf\ true \rightarrow bct$$

$$Primrec\ Bool\ bct\ bcf\ false \rightarrow bcf$$

Definition E.5 *For any type* $T : Type_0$*, a function* Eq *with arity* $Eq : T \rightarrow T \rightarrow Bool$ *is reflexive, symmetric and transitive if the following propositions hold:*

$$\forall t : T.(Eq\ t\ t) =_{Bool} true$$

$$\forall t, t' : T.((Eq\ t\ t') =_{Bool} true) \supset ((Eq\ t'\ t) =_{Bool} true)$$

$$\forall t, t', t'' : T.((Eq\ t\ t') =_{Bool} true) \wedge ((Eq\ t'\ t'') =_{Bool} true) \supset ((Eq\ t\ t'') =_{Bool} true)$$

Notation: *We will denote by* $Equiv(Eq)$ *the conjunction of the previous three axioms.*

Definition E.6 *The function* $and\_Bool : Bool \rightarrow Bool \rightarrow Bool$ *is defined as follows:*

$$and\_Bool\ b\ b' = Primrec\ Bool\ b'\ false\ b$$

Definition E.7 *The function* $or\_Bool : Bool \rightarrow Bool \rightarrow Bool$ *is defined as follows:*

$$or\_Bool\ b\ b' = Primrec\ Bool\ true\ b'\ b$$

Definition E.8 *The function* $not\_Bool : Bool \rightarrow Bool$ *is defined as follows:*

$$not\_Bool\ b = Primrec\ Bool\ false\ true\ b$$

E.2 Functions on type Nat

Definition E.9 *The inductive type* $Nat : Type_0$ *is defined by the following set of constructors:*

$$zero : Nat$$

$$succ : Nat \rightarrow Nat$$

The induction principle $Ind(Nat)$ *which we will use to reason about propositions of type* $Nat \rightarrow Prop$ *is the following:*

$$\Pi P : Nat \rightarrow Prop.(P\ zero) \supset (\forall n : Nat.(P\ n) \supset (P\ (succ\ n))) \supset (\forall n : Nat.P\ n)$$

and the primitive recursion principle $Primrec\ Nat$ *with arity*

$$Primrec\ Nat : T \rightarrow (Nat \rightarrow T \rightarrow T) \rightarrow Nat \rightarrow T$$

for any type $T : Type_0$ *has the following computational rules:*

$$Primrec\ Nat\ bcn\ gcn\ zero \rightarrow bcn$$

$$Primrec\ Nat\ bcn\ gcn\ (succ\ n) \rightarrow gcn\ n\ (Primrec\ Nat\ bcn\ gcn\ n)$$

Definition E.10 *The function* $decr : Nat \rightarrow Nat$ *is defined as follows:*

$$decr\; n \;=\; Primrec\; Nat\; zero\; dgc\; n$$

$$where$$

$$dgc\; n \;=\; n$$

Definition E.11 *The function* $Eqbool\_Nat : Nat \rightarrow Nat \rightarrow Bool$ *is defined as follows:*

$$Eqbool\_Nat\; n\; n' \;=\; Primrec\; Nat\; zc\; sc\; n\; n'$$

$$zc \;=\; \lambda n'' : Nat.Primrec\; Nat\; true\; ssc\; n''$$

$$ssc\; n\; b \;=\; false$$

$$sc\; n\; eqn \;=\; \lambda n'' : Nat.$$

$$Primrec\; Nat\; false\; (scsc\; eqn)\;\; n''$$

$$scsc\; eqn\; n'\; b \;=\; eqn\; n'$$

Proposition E.12 *Eqbool_Nat is symmetric, reflexive and transitive.*

Definition E.13 *The function* $Ltbool\_Nat : Nat \rightarrow Nat \rightarrow Bool$ *is defined as follows:*

$$Ltbool\_Nat\; n\; n' \;=\; Primrec\; Nat\; zc\; sc\; n\; n'$$

$$zc \;=\; \lambda n'' : Nat.Primrec\; Nat\; false\; true\; n''$$

$$sc\; n\; ltn \;=\; \lambda n'' : Nat.$$

$$Primrec\; Nat\; false\; (scsc\; ltn)\;\; n''$$

$$scsc\; ltn\; n'\; b \;=\; ltn\; n'$$

Proposition E.14 *Ltbool_Nat is reflexive and transitive.*

Definition E.15 *The inductive relation* $LtProp\_Nat : Nat \rightarrow Nat \rightarrow Prop$ *is defined by the following constructors:*

$$bc\_LtP : \Pi n : Nat.bc\_LtP\; zero\; n$$

$$gc\_LtP : \Pi n, n' : Nat.\Pi pr : gc\_LtP\; n\; n'.gc\_Ltp\; (succ\; n)\; (succ\; n')$$

Definition E.16 *For any type* $T : Type_0$*, a function* Rel *with arity* $Rel : T \rightarrow T \rightarrow Prop$ *is reflexive, symmetric and transitive if the following propositions hold:*

$$\forall t : T.(Rel\ t\ t)$$

$$\forall t, t' : T.(Rel\ t\ t') \supset (Rel\ t'\ t)$$

$$\forall t, t', t'' : T.(Rel\ t\ t') \land (Rel\ t'\ t'') \supset (Rel\ t\ t'')$$

Proposition E.17 *LtProp_Nat is reflexive and transitive.*

Definition E.18 *The inductive relation* $LeqProp\_Nat : Nat \rightarrow Nat \rightarrow Prop$ *is defined by the following constructors:*

$$bc\_LeP : \Pi n : Nat.LeqProp\_Nat\ zero\ (succ\ n)$$

$$gc\_LeP : \Pi n, n' : Nat.\Pi pr : LeqProp\_Nat\ n\ n'.LeqProp\_Nat\ (succ\ n)\ (succ\ n')$$

Proposition E.19 *LeqProp_Nat is reflexive and transitive.*

E.3 Functions on type Pair

Definition E.20 *The inductive type* $Pair : Type_0 \rightarrow Type_0 \rightarrow Type_0$ *is defined by the following constructor:*

$$mkpair : \Pi A : Type_0.\Pi B : Type_0.A \rightarrow B \rightarrow (Pair\ A\ B)$$

The induction principle $Ind(Pair\ A\ B)$ *for any type* $A, B : Type_0$ *which we will use to reason about propositions of type* $(Pair\ A\ B) \rightarrow Prop$ *is the following:*

$$\Pi P : (Pair\ A\ B) \rightarrow Prop.\forall a : A.\forall b : B.(P\ (mkpair\ A\ B\ a\ b)) \supset (\forall p : Pair\ A\ B.P\ p)$$

and the primitive recursion principle $Primrec\ (Pair\ A\ B)$ *with arity*

$$Primrec\ (Pair\ A\ B)\ : (A \rightarrow B \rightarrow T) \rightarrow (Pair\ A\ B) \rightarrow T$$

for any type $T : Type_0$ *has the following computational rules:*

$$Primrec\ (Pair\ A\ B)\ bcp\ (mkpair\ A\ B\ a\ b) \rightarrow (bcp\ a\ b)$$

Notation: *In some cases we will denote the type* $Pair\ A\ B$ *by the infix operator* $A \times B$ *for any types* $A, B : Type_0$*, and normailly we will denote the expression* $mkpair\ A\ B\ a\ b$ *just by* (a, b) *or we will omit the types* A *and* B *of the expression.*

Definition E.21 *Assume that* $A, B : Type_0$*. The function* $fst : (Pair\ A\ B) \rightarrow A$ *is defined as follows:*

$$fst\ p = Primrec\ (Pair\ A\ B)\ getfst\ p$$

$$where$$

$$getfst\ a\ b = a$$

Definition E.22 *Assume that* $A, B : Type_0$*. The function* $snd : (Pair\ A\ B) \rightarrow B$ *is defined as follows:*

$$snd\ p = \quad Primrec\ (Pair\ A\ B)\ getsnd\ p$$

$$where$$

$$getsnd\ a\ b\ =\ b$$

E.4 Functions on type List

Definition E.23 *The inductive type* $List : Type_0 \rightarrow Type_0$ *is defined by the following set of constructors:*

$$nil : \Pi A : Type_0.List\ A$$

$$cons : \Pi A : Type_0.A\ \rightarrow\ (List\ A)\ \rightarrow\ (List\ A)$$

The induction principle $Ind(List\ A)$ *for any type* $A : Type_0$ *which we will use to reason about propositions of type* $(List\ A) \rightarrow Prop$ *is the following:*

$$\Pi P : (List\ A)\ \rightarrow\ Prop.(P\ (nil\ A))\ \supset\ (\forall a : A.\forall l : List\ A.(P\ l) \supset (P\ (cons\ A\ a\ l)))\ \supset$$

$$(\forall l : List\ A.P\ l)$$

and the primitive recursion principle $Primrec\ (List\ A)$ *with arity*

$$Primrec\ (List\ A)\ : T \rightarrow\ (A \rightarrow (List\ A) \rightarrow T \rightarrow T)\ \rightarrow\ (List\ A) \rightarrow\ T$$

for any type $T : Type_0$ *has the following computational rules:*

$$Primrec\ (List\ A)\ bcl\ gcl\ (nil\ A)\ \ \rightarrow\ \ bcl$$

$$Primrec\ (List\ A)\ bcl\ gcl\ (cons\ A\ a\ l)\ \ \rightarrow\ \ (gcl\ A\ a\ l\ (Primrec\ (List\ A)\ bcl\ gcl\ l))$$

Definition E.24 *The function* $hd : \Pi T : Type_0.T\ \rightarrow\ (List\ T)\ \rightarrow T$ *is defined as follows:*

$$hd\ T\ a\ l\ = \quad Primrec\ T\ hdbc\ hdgc\ l$$

$$where$$

$$hdbc\ =\ a$$

$$hdgc\ b\ l\ c\ =\ b$$

Definition E.25 *The function* $last : \Pi T : Type_0.T \rightarrow (List\ T) \rightarrow T$ *is defined as follows:*

$$\begin{aligned} last\ T\ a\ l\ = \quad & Primrec\ T\ lbc\ lgc\ l \\ & where \\ & lbc\ =\ a \\ & lgc\ b\ l\ c\ =\ Primrec\ Bool\ b\ c\ (Eqbool\_List\ l\ (nil\ T)) \end{aligned}$$

Definition E.26 *The function* $emptylist : \Pi T : Type_0.\ (List\ T) \rightarrow Bool$ *is defined as follows:*

$$\begin{aligned} emptylist\ T\ l\ = \quad & Primrec\ T\ elbc\ elgc\ l \\ & where \\ & elbc\ =\ true \\ & elgc\ a\ l\ b\ =\ false \end{aligned}$$

Definition E.27 *The function* $tail : \Pi T : Type_0.\ (List\ T) \rightarrow (List\ T)$ *is defined as follows:*

$$\begin{aligned} tail\ T\ l\ = \quad & Primrec\ T\ tlbc\ tlgc\ l \\ & where \\ & tlbc\ =\ nil\ T \\ & tlgc\ a\ l\ l'\ =\ l \end{aligned}$$

Definition E.28 *The function* $addlast : \Pi T : Type_0.T \rightarrow (List\ T) \rightarrow T$ *is defined as follows:*

$$\begin{aligned} addlast\ T\ a\ l\ = \quad & Primrec\ T\ albc\ algc\ l \\ & where \\ & albc\ =\ cons\ T\ a\ (nil\ T) \\ & algc\ a\ l\ l'\ =\ cons\ T\ a\ l' \end{aligned}$$

Definition E.29 *The function* $concat : \Pi T : Type_0.(List\ T) \rightarrow (List\ T) \rightarrow$

(List T) is defined as follows:

$$concat\ T\ l\ l' = Primrec\ (List\ T)\ l'\ congc\ l$$

$$where$$

$$congc\ a\ l\ l' = cons\ T\ a\ l'$$

Definition E.30 *The function* $remove : \Pi T : Type_0.(\Pi eqbt : T \rightarrow T \rightarrow Bool) \rightarrow T \rightarrow (List\ T) \rightarrow (List\ T)$ *is defined as follows:*

$$remove\ T\ eqbt\ el\ l = Primrec\ (List\ T)\ (nil\ T)\ (addifneq\ el)\ l$$

$$where$$

$$addifneq\ a\ a'\ l\ l' = Primrec\ bool\ l'\ (cons\ a\ l')\ (eqbt\ a\ a')$$

Definition E.31 *The function* $is\_in\_bool : \Pi T : Type_0.(\Pi eqbt : T \rightarrow T \rightarrow Bool) \rightarrow T \rightarrow (List\ T) \rightarrow bool$ *is defined as follows:*

$$is\_in\_bool\ T\ eqbt\ el\ l = Primrec\ (List\ T)\ false\ (trueifeq\ el)\ l$$

$$where$$

$$trueifeq\ a\ a'\ l\ b = Primrec\ bool\ true\ b\ (eqbt\ a\ a')$$

Definition E.32 *The function* $reverse : \Pi T : Type_0.(List\ T) \rightarrow (List\ T)$ *is defined as follows:*

$$reverse\ T\ l = Primrec\ (List\ T)\ revbc\ revgc\ l$$

$$where$$

$$revbc = nil\ T$$

$$revgc\ a\ l\ l' = add\_last\ a\ l'$$

Definition E.33 *The function* $map : \Pi T, T' : Type_0.(T \rightarrow T') \rightarrow (List\ T)\ (List\ T')$ *is defined as follows:*

$$map\ T\ T'\ f\ l = Primrec\ (List\ T)\ mapbc\ mapgc\ l$$

$$where$$

$$mapbc = nil\ T'$$

$$mapgc\ a\ l\ l' = cons\ T'\ (f\ a)\ l'$$

Definition E.34 *The function join* : $\Pi T_0 : Type_0.\Pi T_1 : Type_0.(List\ T_0) \rightarrow (List\ T_1) \rightarrow (List\ (Pair\ T_0\ T_1))$ *is defined as follows:*

$$join\ T_0\ T_1\ l_1\ l_2 \ = \ Primrec\ (List\ T_0)\ jnilc\ jconsc\ l_1\ l_2$$

$$jnilc \ = \ \lambda l_2 : List\ T_1.(nil\ (Pair\ T_1\ T_2))$$

$$jconsc\ a\ l\ joinn \ =$$

$$\lambda l_2 \ : \ List\ T_2.Primrec\ (List\ T_2)\ (nil(Pair\ T_1\ T_2))\ (jcconsc\ a\ joinn)\ l_2$$

$$jcconsc\ a\ joinn\ b\ l\ pl \ =$$

$$cons\ (Pair\ T_1\ T_2)\ (mkpair\ T_1\ T_2\ a\ b)\ (joinn\ l)$$

Proposition E.35 *The following propositions hold:*

$$join\ (nil\ T_1)\ (nil\ T_2) \ = \ (nil\ (Pair\ T_1\ T_2)$$

$$\forall ht : T_1.\forall hv : T_2.\forall htl : List\ T_1.\forall hvl : List\ T_2.$$

$$(join\ (cons\ T_1\ ht\ htl)\ (cons\ T_2\ hv\ hvl)) \ =$$

$$(cons\ (Pair\ T_1\ T_2)\ (mkpair\ T_1\ T_2\ ht\ hv)\ (join\ htrml\ hvl))$$

Definition E.36 *The function Eqbool_list* : $\Pi T : Type.(List\ T) \rightarrow (List\ T) \rightarrow Bool$ *is defined as follows:*

$$Eqbool\_list\ T\ l\ l' \ = \ Primrec\ (List\ T)\ nilc\ consc\ l\ l'$$

$$nilc \ = \ \lambda el'' : (List\ T).Primrec\ (List\ T)\ true\ conscnc\ vn''$$

$$conscnc\ el\ l\ b \ = \ false$$

$$consc\ el\ l\ eqln \ = \ \lambda l'' : List\ T.$$

$$Primrec\ (List\ T)\ (\lambda el : T.false)\ (consccc\ el\ eqln)\ \ l''$$

$$consccc\ el\ eqvnn\ el'\ l'\ b \ = \ (and\ (eqvnn\ l')\ (Eqbool\_Vs\ el\ el'))$$

Definition E.37 *The inductive relation Not_in_list* : $\Pi T : Type.(T \rightarrow T \rightarrow$

$Bool) \rightarrow T \rightarrow (List\ T) \rightarrow Prop$ *is defined by the following constructors:*

$$\{basec\_Nin : \Pi T : Type.\Pi eqbt : T \rightarrow T \rightarrow Bool.\Pi el : T.Not\_in\_list\ T\ eqb\ (nil\ T)$$

$$consc\_Nin : \Pi T : Type.\Pi eqbt : T \rightarrow T \rightarrow Bool.\Pi el, el' : T.\Pi l : (List\ T).$$

$$\Pi noteq : ((eqbt\ el\ el') = false).Not\_in\_list\ T\ eqb\ el\ (cons\ T\ el'\ l)\}$$

Definition E.38 *The inductive relation*

$$Norep\_list : \Pi T : Type.\Pi eqbt : T \rightarrow T \rightarrow Bool.(List\ T) \rightarrow Prop$$

is defined by the following constructors:

$$\{norep\_bc : \Pi T : Type.\Pi eqbt : T \rightarrow T \rightarrow Bool.Norep\_list\ T\ eqbt\ (nil\ T)$$

$$norep\_gc : \Pi T : Type.\Pi eqbt : T \rightarrow T \rightarrow Bool.\Pi el : T.\Pi l : List\ T.$$

$$\Pi notin : Not\_in\_list\ T\ eqbt\ el\ l.Norep\_list\ T\ (cons\ T\ el\ l)\}$$

Definition E.39 *The inductive relation* $Is\_in\_list : \Pi T : Type.T \rightarrow (List\ T) \rightarrow Prop$ *is defined by the following constructors:*

$$\{basec\_Inlist : \Pi T : Type.\Pi el : T.\Pi l : (List\ T).Is\_in\_list\ el\ (cons\ T\ el\ l)$$

$$consc\_Inlist : \Pi T : Type.\Pi el, el' : T.\Pi l : (List\ T).\Pi pr : Is\_in\_list\ el\ l.$$

$$Is\_in\_list\ el\ (cons\ T\ el'\ l)$$

$$\}$$

Definition E.40 *The inductive relation*

$$Not\_emptyl : \Pi T : Type_0.\Pi l : List\ T.Prop$$

is defined by the following constructor:

$$bc\_Ne : \Pi T : Type_0.\Pi el : T.\Pi l : List\ T.Not\_emptyl\ T\ el\ (cons\ el\ l)$$

Definition E.41 *The inductive relation*

$$Same\_length : \Pi T_1, T_2 : Type_0.\Pi l : List\ T_1.\Pi l' : List\ T_2.Prop$$

is defined by the following set of constructors:

$$nil\_sl : \Pi T_1, T_2 : Type_0.Same\_length\ T_1\ T_2\ (nil\ T_1)\ (nil\ T_2)$$

$$cons\_sl : \Pi T_1, T_2 : Type_0.\Pi t : T_1.\Pi t' : T_2.\Pi tl : List\ T_1.\Pi tl' : List\ T_2.$$

$$\Pi slpr : Same\_length\ tl\ tl'.Same\_length\ T_1\ T_2\ (cons\ T_1\ t\ tl)\ (cons\ T_2\ t'\ tl')$$

Notation: If it can be inferred from the context, we will usually omit the type arguments of the functions on lists.

F Adequate encoding of $\Pi_{HOL}(\Gamma, \Sigma)$

F.1 Adequate encondings of the sentences

In this subsection we are going to present the adequate encoding of a type system for higher-order logic which is equivalent to the one presented in chapter 5. The main difference is that we split the set of free variables in two: the initial set of free variables of the derivation and the set of bound variables of a variables which become free in the derivation process. We will denote this new set of free variables as a pair of the form (X, X') where the first is the initial set of free variables and the second the set of bound variables which have become free, and if the second component is empty we will normally denote the set $(X, [])$ just by X.

This split is necessary to determine the difference between the last DeBruijn index assigned to the bound variables in the scope of every ocurrence of a variable in a higher-order term and the last index assigned in the original set of free variables. This index (which is referred as bound level and it is an information which every variable in a higher-order term has) is necessary to update the indexes of the variables of the higher-order term which replaces a variable in the substitution operation.

Definition F.1 *The set of typing rules of* Π_{HOL} *is defined as follows:*

$$\frac{}{(X, X') \blacktriangleright x_\tau : \tau}\; x \notin X'_\tau, x \in X_\tau \qquad (Ass1)$$

$$\frac{}{(X, X') \blacktriangleright x_\tau : \tau}\; x \in X'_\tau \qquad (Ass2)$$

$$\frac{(X, X') \blacktriangleright t_1 : s_1 \quad \ldots \quad (X, X') \blacktriangleright t_n : s_n}{(X, X') \blacktriangleright f(t_1, \ldots, t_n) : s}\; f : s_1 \times \ldots \times s_n \to s \in \Sigma \qquad (Appl)$$

$$\frac{(X, X') \cup \{x_1 : \tau_1, \ldots, x_n : \tau_n\}) \blacktriangleright \phi : \mathbf{Prop}}{(X, X') \blacktriangleright \lambda(x_1 : \tau_1, \ldots, x_n : \tau_n).\phi : [\tau_1, \ldots, \tau_n]} \qquad (\lambda Abs)$$

$$\frac{(X, X') \blacktriangleright t_1 : \tau_1 \quad \ldots \quad (X, X') \blacktriangleright t_n : \tau_n \quad (X, X') \blacktriangleright t : [\tau_1, \ldots, \tau_n]}{(X, X') \blacktriangleright t(t_1, \ldots, t_n) : \mathbf{Prop}} \qquad (\lambda APPL)$$

$$\frac{(X, X' \cup x : \tau) \blacktriangleright \phi : \mathbf{Prop}}{(X, X') \blacktriangleright \forall x : \tau.\phi : \mathbf{Prop}} \qquad (Forall)$$

$$\frac{(X, X') \blacktriangleright \phi : \mathbf{Prop} \quad (X, X') \blacktriangleright \phi' : \mathbf{Prop}}{(X, X') \blacktriangleright \phi \supset \phi' : \mathbf{Prop}} \qquad (Implies)$$

Definition F.2 *The substitution operation on terms* $\_\{\_/\_\} : T_{\Sigma,s}(X) \to T_{\Sigma,r}(X) \to X_r \to T_{\Sigma,s}(X)$ *for any signature* $\Sigma \in |AlgSig|$ *and for any sort* $s \in Sorts(\Sigma)$

is inductively defined as follows:

$$y_{r'}\ \{t\ /\ x_r\} \quad = t \quad ,if x_r\ =\ y_{r'}$$
$$\quad = y_{r'} \quad ,otherwise$$

$$f\ (t_1, \ldots, t_n)\ \{t\ /\ x_r\}\ =\ \ f\ (t_1\ \{\ t\ /\ x_r\ \}, \ldots, t_n\{\ t\ /\ x_r\ \})$$

where

$t \in T_{\Sigma,r}(X),\ \ f\ :\ s_1\ \times\ \ldots\ \times\ s_n\ \rightarrow\ s\ \in\ \Sigma,$

$t_1\ \in\ T_{\Sigma,s_1}(X), \ldots, t_n\ \in\ T_{\Sigma,s_n}(X)$

Definition F.3 *The substitution operation on terms* $\_\{\_/\_\} : T_\Sigma(X)\ \rightarrow\ T_{\Sigma,r}(X)\ \rightarrow\ X_r\ \ \rightarrow\ T_\Sigma(X)$ *for any signature* $\Sigma\ \in\ |AlgSig|$ *is inductively defined as follows:*

$$y_{r'}\ \{t\ /\ x_r\} \quad = t \quad ,if x_r\ =\ y_{r'}$$
$$\quad = y_{r'} \quad ,otherwise$$

$$f\ (t_1, \ldots, t_n)\ \{\ t\ /\ x_r\ \}\ =\ \ f\ (t_1\ \{\ t\ /\ x_r\ \}, \ldots, t_n\{\ t\ /\ x_r\ \})$$

where

$t \in T_{\Sigma,r}(X),\ \ f\ :\ s_1\ \times\ \ldots\ \times\ s_n\ \rightarrow\ s\ \in\ \Sigma,$

$t_1\ \in\ T_{\Sigma,s_1}(X), \ldots, t_n\ \in\ T_{\Sigma,s_n}(X)$

Definition F.4 *The substitution operation on terms of a free variable of this term by another term* $\_\{\_/\_\}:\ Term_{HOL}(\Sigma)\ \rightarrow\ Term_{HOL}(\Sigma)\ \rightarrow\ X_{HOL}\ \rightarrow\ Term_{HOL}(\Sigma)$ *for any signature* $\Sigma\ \in\ |AlgSig|$ *is inductively defined as follows:*

$$y_{r'}\ \{t\ /\ x_r\} \quad = t \quad ,if x_r\ =\ y_{r'}$$
$$\quad = y_{r'} \quad ,otherwise$$

$$f\ (t_1, \ldots, t_n)\ \{t\ /\ x_r\}\ =\ \ f\ (t_1\ \{t\ /\ x_r\}, \ldots, t_n\{t\ /\ x_r\})$$

where

$t \in T_{\Sigma,r}(X),\ \ f\ :\ s_1\ \times\ \ldots\ \times\ s_n\ \rightarrow\ s\ \in\ \Sigma,$

$t_1\ \in\ T_{\Sigma,s_1}(X), \ldots, t_n\ \in\ T_{\Sigma,s_n}(X)$

$$\lambda(x_1 : \tau_1, \ldots, x_n : \tau_n).\phi \; \{t \;/\; x_\tau\} \;=$$

$$\lambda(x_1 : \tau_1, \ldots, x_n : \tau_n).\phi \qquad\qquad , if\ \exists i \in [1..n].x_{i,\tau_i} = x_\tau$$

$$\lambda(x'_1 : \tau_1, \ldots, x'_n : \tau_n).(((\ldots(\phi \; \{x'_{1,\tau_1} \;/\; x_{1,\tau_1}\}) \; \ldots)\{x'_{n,\tau_n} \;/\; x_{n,\tau_n}\}) \; \{t \;/\; x_\tau\})$$

$$, if\ \forall i \in [1..n].x_{i,\tau_i} \neq x_\tau$$

$where$

$\forall\; i \in [1..n].\; x_{i,\tau_i} \notin FV(t) \;\Rightarrow\; x'_i \;=\; x_i \;\wedge$

$x_{i,\tau_i} \in FV(t) \;\Rightarrow\; x'_{i,\tau} \notin FV(t) \;\wedge\; x'_{i,\tau} \notin FV(\phi) \;\wedge x'_{i,\tau} \notin BV(\phi)$

$t \in Sen_{HOL}(\Sigma, X_{HOL}, \tau), \phi \in Sen_{HOL}(\Sigma, X_{HOL}, \mathbf{Prop})$

$\tau_1 \;\in\; Types_{HOL}(\Sigma), \ldots, \tau_n \;\in\; Types_{HOL}(\Sigma)$

$$t \; (t_1, \ldots, t_n) \; \{t \;/\; x_\tau\} \;=\; t \; \{t \;/\; x_\tau\} \; (t_1 \; \{t \;/\; x_\tau\}, \ldots, t_n \; \{t \;/\; x_\tau\})$$

$where$

$t \;\in\; Sen_{HOL}(\Sigma, X_{HOL}, [\tau_1, \ldots, \tau_n]), t_1 \;\in\; Sen_{HOL}(\Sigma, X_{HOL}, \tau_1),$

$t_n \;\in\; Sen_{HOL}(\Sigma, X_{HOL}, \tau_n)$

$$\forall x_1 : \tau_1.\phi \; \{t \;/\; x_\tau\} \;=\; \forall x'_1 : \tau_1.((\phi\{x'_{1,\tau}/x_{1,\tau}\}) \; \{t \;/\; x_\tau\}) \qquad , if\ x_{1,\tau_1} \neq x_\tau$$

$$=\; \forall x_1 : \tau_1.\phi \qquad\qquad , if\ x_{1,\tau_1} = x_\tau$$

$where$

$x_{1,\tau} \notin FV(t) \;\Rightarrow\; x'_{1,\tau} \;=\; x_{1,\tau} \;\wedge$

$x_{1,\tau} \in FV(t) \;\Rightarrow\; x'_{1,\tau} \notin FV(t) \;\wedge x'_{1,\tau} \notin FV(\phi) \;\wedge x'_{1,\tau} \notin BV(\phi),$

$t \in Sen_{HOL}(\Sigma, X_{HOL}, \tau), \phi \in Sen_{HOL}(\Sigma, X_{HOL}, \mathbf{Prop})$

$$\phi \supset \phi' \; \{t \;/\; x_\tau\} \;=\; \phi \; \{t \;/\; x_\tau\} \;\supset\; \phi' \; \{t \;/\; x_\tau\}$$

$where$

$t \in Sen_{HOL}(\Sigma, X_{HOL}, \tau),$

$\phi, \phi' \in Sen_{HOL}(\Sigma, X_{HOL}, \mathbf{Prop})$

F.1.1 Encoding of sorted variables and first-order terms

First, we define the encoding of sorts of signatures,sorted variables and sorted variables with indexes. We make the presentation self-contained and therefore we repeat some definitions of chapter 3.

Definition F.5 *For any $\Sigma \in |AlgSig|$, the inductive relation Sorts is inductively defined by the following set of constructors:*

$$\{s\_Srts : Sorts \mid s \in Sorts(\Sigma)\}$$

Definition F.6 *For any $\Sigma \in |AlgSig|$, the function $Eqbool\_Srts : Sorts \to Sorts \to Bool$ is defined as follows:*

$$Eqbool\_Srts\ s\ s' \ = \ Primrec\ Sorts\ (s_1c\ s')\ \ldots\ (s_nc\ s')\ s$$

$$\begin{aligned} s_1c\ s' &= Primrec\ Sorts\ true\ \ldots\ false\ s' \\ &\vdots \\ s_nc\ s' &= Primrec\ Sorts\ true\ \ldots\ false\ s' \end{aligned}$$

Definition F.7 *The type Var_symbol is inductively defined by the following set of constructors:*

$$a, \ldots, z : Var\_symbol$$

$$A, \ldots, Z : Var\_symbol$$

$$\_, ', \$: Var\_symbol$$

Definition F.8 *The function $Eqbool\_Vs : Var\_symbol \to Var\_symbol \to Bool$ is defined as follows:*

$$Eqbool\_Vs\ vs\ vs' \ = \ Primrec\ (ac\ vs')\ \ldots\ (Zc\ vs')\ \ldots\ (\$c\ vs')\ vs$$

$$\begin{aligned} ac\ vs' &= Primrec\ true\ \ldots\ false\ \ldots\ false\ vs' \\ &\vdots \\ Zc\ vs' &= Primrec\ false\ \ldots\ true\ \ldots\ false\ vs' \\ &\vdots \\ \$c\ vs' &= Primrec\ false\ \ldots\ false\ \ldots\ true\ vs' \end{aligned}$$

Definition F.9 *For any type $T : Type_0$, the inductive type Nelist T is defined by the following constructors:*

$$first\_Nel\ :\ T\ \to\ Nelist\ T$$

$$cons\_Nel\ :\ T\ \to\ (Nelist\ T)\ \to\ (Nelist\ T)$$

Definition F.10 *The type* Var_name *is defined as follows:*

$$Var\_name \;=\; Ne\_list\; Var\_symbol$$

Definition F.11 *The function* $Eqbool\_Vn : Var\_name \;\rightarrow\; Var\_name \;\rightarrow\; Bool$ *is defined as follows:*

$$Eqbool\_Vn\; vn\; vn' \;=\; Primrec\; Var\_name\; firstc\; consc\; vn\; vn'$$

$$firstc\; vs \;=\; \lambda vn'' : Var\_name.Primrec\; Var\_name\; (\lambda vs : Var\_symbol.true)\; conscfc\; vn''$$

$$conscfc\; vs\; vn\; b \;=\; false$$

$$consc\; vs\; vn\; eqvnn \;=\; \lambda vn'' : Var\_name.$$

$$Primrec\; Var\_name\; (\lambda vs : Var\_symbol.false)\; (consccc\; vs\; eqvnn)\;\; vn''$$

$$consccc\; vs\; eqvnn\; vs'\; vn'\; b \;=\; (and\; (eqvnn\; vn')\; (Eqbool\_Vs\; vs\; vs'))$$

Definition F.12 *The type* Var_index *is inductively defined by the following set of constructors:*

$$first\_Vi : Var\_index$$

$$next\_Vi : Var\_index \rightarrow Var\_index$$

Definition F.13 *The function* $Eqbool\_Vi : Var\_index \;\rightarrow\; Var\_index \;\rightarrow\; Bool$ *is defined as follows:*

$$Eqbool\_Vi\; vi\; vi' \;=\; Primrec\; Var\_index\; firstc\; nextc\; vi\; vi'$$

$$firstc \;=\; \lambda vn'' : Var\_index.Primrec\; Var\_index\; true\; nextcfc\; vn''$$

$$nextcfc\; vi\; b \;=\; false$$

$$nextc\; vi\; eqvi \;=\; \lambda vi'' : Var\_index.$$

$$Primrec\; Var\_index\; false\; (nextcnc\; vi\; eqvi)\;\; vi''$$

$$nextcnc\; vi\; eqvi\; vi'\; b \;=\; eqvi\; vi$$

Definition F.14 *The function* $Ltbool\_Vi : Var\_index \;\rightarrow\; Var\_index \;\rightarrow$

Bool is defined as follows:

$$Ltbool\_Vi\ vi\ vi' \ = \ Primrec\ Var\_index\ firstc\ nextc\ vi\ vi'$$

$$firstc \ = \ \lambda vn'' : Var\_index.Primrec\ Var\_index\ false\ nextcfc\ vn''$$

$$nextcfc\ vi\ b \ = \ true$$

$$nextc\ vi\ ltvi \ = \ \lambda vi'' : Var\_index.$$

$$Primrec\ Var\_index\ false\ (nextcnc\ vi\ ltvi)\ \ vi''$$

$$nextcnc\ vi\ ltvi\ vi'\ b \ = \ ltvi\ vi$$

Definition F.15 *The function* $add\_Vi : Var\_index \rightarrow Var\_index \rightarrow Var\_index$ *is defined as follows:*

$$add\_Vi\ vi\ vi' \ = \ Primrec\ Var\_index\ vi\ nextc\ vi'$$

$$where$$

$$nextc\ vi\ vif \ = \ next\_Vi\ vif$$

Definition F.16 *The function* $decr\_Vi : Var\_index \rightarrow Var\_index \rightarrow Var\_index$ *is defined as follows:*

$$decr\_Vi\ vi \ = \ Primrec\ Var\_index\ first\_Vi\ nextc\ vi'$$

$$where$$

$$nextc\ vi\ vif \ = \ vi$$

Definition F.17 *The function* $substract\_Vi : Var\_index \rightarrow Var\_index \rightarrow Var\_index$ *is defined as follows:*

$$substract\_Vi\ vi\ vi' \ = \ Primrec\ Var\_index\ vi\ nextc\ vi'$$

$$where$$

$$nextc\ vi\ vif \ = \ decr\_Vi\ vif$$

Definition F.18 *For any* $\Sigma \in |AlgSig|$*, the type* Var *is defined as:*

$$Var \ = Pair\ Var\_name\ Sorts$$

Definition F.19 *The function* $Eqbool\_Var$ *with arity*

$$Eqbool\_Var : Var \rightarrow Var \rightarrow Bool$$

is defined as follows:

$$Eqbool\_Var\ v\ v' \ = \ Primrec\ Var\ (mkpairc\ v')\ v$$

$$where$$

$$mkpairc\ v'\ vn\ s \ = \ Primrec\ Var\ (mkpaircc\ vn\ s)\ v'$$

$$mkpaircc\ vn\ s\ vn'\ s' \ = \ (and\ (Eqbool\_Vn\ vn\ vn')\ (Eqbool\_Srts\ s\ s'))$$

Definition F.20 *For any* $\Sigma \in |AlgSig|$*, the type* $Invar$ *is defined as:*

$$Invar \ = \ Pair\ Var\ Pair\ Var\_index\ Var\_index$$

Definition F.21 *The function* $Eqbool\_Ivar$ *with arity*

$$Eqbool\_Ivar : Invar \to Invar \to Bool$$

is defined as follows:

$$Eqbool\_Ivar\ iv\ iv' \ = \ Primrec\ Var\ (mkpairc\ iv')\ iv$$

$$where$$

$$mkpairc\ iv'\ v\ vip \ = \ Primrec\ Var\ (mkpaircc\ v\ vip)\ iv'$$

$$mkpaircc\ v\ vip\ v'\ vip' \ = \ (and\ (Eqbool\_Var\ v\ v')\ (Eqbool\_Vi\ (fst\ vi)\ (fst\ vi')))$$

Definition F.22 *The function* $getindex\_Iv : Invar \ \to \ Var\_index$ *is defined as follows:*

$$getindex\_Iv\ iv \ = \ (fst\ (snd\ iv))$$

Definition F.23 *The function* $getblevel\_Iv : Invar \ \to \ Var\_index$ *is defined as follows:*

$$getblevel\_Iv\ iv \ = \ (snd\ (snd\ iv))$$

Definition F.24 *The function* $assindex\_Iv : Invar \ \to \ Var\_index \ \to \ Invar$ *is defined as follows:*

$$assindex\_Iv\ iv\ vi = \ (fst\ iv, (vi, (snd\ (snd\ iv))))$$

Definition F.25 *The function* $assblevel\_Iv : Invar \ \to \ Var\_index \ \to \ Invar$ *is defined as follows:*

$$assblevel\_Iv\ iv\ vi = \ (fst\ iv, ((fst\ (snd\ iv)), vi))$$

Definition F.26 *The function* $addindex\_Iv : Invar \rightarrow Var\_index \rightarrow Invar$ *is defined as follows:*

$$addindex\_Iv\ iv\ vi = \ (fst\ iv, add\_Vi\ vi\ (fst\ (snd\ iv)), (snd\ (snd\ iv)))$$

Definition F.27 *The function* $addblevel\_Iv : Invar \rightarrow Var\_index \rightarrow Invar$ *is defined as follows:*

$$addblevel\_Iv\ iv\ vi = \ (fst\ iv, ((fst\ (snd\ iv)), add\_Vi\ (snd\ (snd\ iv))\ vi))$$

Next, we define terms of sort s for any sort s of a given signature Σ and the set of all terms.

Definition F.28 *For any* $\Sigma \in |AlgSig|$ *the mutually recursive inductive types* $\{\ Term\_s \mid s \in Sorts(\Sigma)\}$ *is defined by the following set of constructors for any sort* $s \in Sorts(\Sigma)$*:*

$$\{var\_s\_Trms : Invar \ \rightarrow \ Term\_s\} \cup$$

$$\{f\_Trms : Term\_s_1 \rightarrow \ldots \rightarrow Term\_s_n \rightarrow Term\_s \mid$$

$$f : s_1 \times \ldots \times s_n \rightarrow s \in \Sigma \text{ and } f \text{ is not overloaded in } \Sigma\} \cup$$

$$\{f\_s_1\_\ldots\_s_n\_s\_Trms : Term\_s_1 \rightarrow \ldots \rightarrow Term\_s_n \rightarrow Term\_s \mid$$

$$f : s_1 \times \ldots \times s_n \rightarrow s \in \Sigma \text{ and } f \text{ is overloaded in } \Sigma\}$$

Definition F.29 *For any* $\Sigma \in |AlgSig|$*, the inductive type* $Term$ *is defined by the following set of constructors:*

$$\begin{array}{l} trm\_s_1\_Trm : Term\_s_1 \ \rightarrow \ Term \\ \vdots \\ trm\_s_n\_Trm : Term\_s_n \ \rightarrow \ Term \end{array}$$

F.1.2 Encoding of higher-order syntax

In the following we define higher-order types, higher-order variables, higher order variables with indexes, set of higher-order variables, higher-order terms and well-formed higher-order terms. Since in higher-order types can appear list of higher-order types we define both as mutually recursive types. The same happens with higher-order terms and list of higher-order terms and well-formed higher-order terms and list of well-formed higher-order terms.

Definition F.30 *The mutually recursive inductive types* $Holtype$ *and* $Holtype\_list$ *for a given signature* Σ *are defined by the following set of construc-*

tors:

$$\{\ s\_Holt : Sorts \ \rightarrow\ Holtype \mid s \in Sorts(\Sigma)\ \} \cup$$

$$\{\ prop\_Holt : Holtype,$$

$$holrel\_Holt : Holtype\_list \rightarrow Holtype\ ,$$

$$nil\_Holt : Holtype\_list$$

$$cons\_Holt : Holtype \rightarrow Holtype\_list \rightarrow Holtype\_list\}$$

Definition F.31 *The mutually recursive functions $Eqbool\_Hty : Holtype \rightarrow Holtype \rightarrow Bool$ and $Eqbool\_Htyl : Holtype\_list \rightarrow Holtype\_list \rightarrow Bool$ for a given signature Σ is defined as follows:*

$$\begin{aligned} Eqbool\_Hty\ hty\ hty' \ &=\ Primrec\ Holtype\ s_1c\_Holt\ \ldots\ s_nc\_Holt \\ &\quad propc\ holrelc\ nilc\ consc\ hty\ hty' \\ Eqbool\_Htyl\ hty\ hty' \ &=\ Primrec\ Holtype\_list\ s_1c\_Holt\ \ldots\ s_nc\_Holt \\ &\quad propc\ holrelc\ nilc\ consc\ htyl\ htyl' \end{aligned}$$

where

$$s_1c\_Holt \ =\ \lambda hty' : Holtype.Primrec\ Holtype\ (\lambda s : Sorts.true)\ \ldots$$

$$(\lambda s : Sorts.false)\ false\ relcs1\ false\ conscs1\ hty'$$

$$relcs1\ htl\ beqhtls1 \ =\ false$$

$$conscs1\ ht\ htl\ beqhts1\ beqhtls1 \ =\ false$$

$$\vdots$$

$$s_nc\_Holt \ =\ \lambda hty' : Holtype.Primrec\ Holtype\ (\lambda s : Sorts.false)\ \ldots$$

$$(\lambda s : Sorts.true)\ false\ relcsn\ false\ conscsn\ hty'$$

$$relcsn\ htl\ beqhtlsn \ =\ false$$

$$conscsn\ ht\ htl\ beqhtlsn\ beqhtsm \ =\ false$$

$propc \ = \ \lambda hty' : Holtype.Primrec \ Holtype \ (\lambda s : Sorts.false)$

$\ldots \ (\lambda s : Sorts.false) \ true \ relcp \ \ false \ conscp \ hty'$

$relcp \ htyl \ beqhtl \ = \ false$

$conscp \ ht \ beqhtp \ htl \ beqhtlp \ = \ false$

$holrelc \ htl \ htleqf \ \ = \lambda hty' : Holtype. \ Primrec \ Holtype \ (\lambda s : Sorts.false) \ \ldots$

$(\lambda s : Sorts.false) \ false \ relcr \ false \ conscr \ hty'$

$relcr \ htyl \ beqhtl \ = \ beqhtl$

$conscr \ ht \ beqhtr \ htl \ beqhtlr \ = \ false$

$nilc \ = \ \lambda htyl'' : List \ Holtype.Primrec \ (List \ Holtype) \ (\lambda s : Sorts.false)$

$\ldots \ (\lambda s : Sorts.false) \ false \ relcnc \ true \ conscnc \ htyl''$

$relcnc \ htyl \ beqhtl \ = \ false$

$conscnc \ ht \ b \ htl \ b' \ = \ false$

$consc \ ht \ hteqf \ htl \ htleqf \ \ = \ \lambda htyl'' : List \ Holtype.$

$Primrec \ (List \ Holtype) \ (\lambda s : Sorts.false) \ \ldots$

$(\lambda s : Sorts.false) \ false \ relccc \ false \ (conscc \ hteqf \ htleqf) \ \ htyl''$

$relccc \ htyl \ beqhtl \ = \ false$

$consccc \ hteqf \ htleqf \ ht' \ htyl' \ bht' \ bhtyl' \ \ = \ (and \ (htleqf \ htyl') \ (hteqf \ ht'))$

Definition F.32 *The functions Eqbsort_Hty with arity*

$$Eqbsort\_Hty : Holtype \rightarrow Bool$$

and the function Eqbsort_Htyl with arity

$$Eqbsort\_Htyl : (Holtype\_list) \rightarrow Bool$$

are defined by mutual recursion as follows:

$$Eqbsort\_Hty\ hty \ = \ Primrec\ Holtype\ s_1c\ ldots\ s_nc\ propc\ holrelc\ nilc\ consc\ hty$$

$$Eqbsort\_Htyl\ htyl \ = \ Primrec\ Holtype\_list\ s_1c\ ldots\ s_nc\ propc\ holrelc\ nilc\ consc\ htyl$$

$$where$$

$$s_1c\ s_1 \ = \ true$$
$$\vdots$$
$$s_nc\ s_n \ = \ true$$

$$propc \ = \ false$$

$$holrelc\ htyl\ b \ = \ false$$

$$nilc \ = \ false$$

$$consc\ hty\ htyl\ b\ b' \ = \ false$$

Definition F.33 *The type Holvar for a given signature* Σ *is defined as:*

$$Holvar = pair\ Var\_name\ Holtype$$

Definition F.34 *The type Holinvar for a given signature* Σ *is defined as:*

$$Holinvar = pair\ Holvar\ (pair\ Var\_index\ Var\_index)$$

Definition F.35 *The function Eqbool_Hvar with arity*

$$Eqbool\_Hvar : Holvar \to Holvar \to Bool$$

is defined as follows:

$$Eqbool\_Hvar\ hv\ hv' \ = \ Primrec\ Holvar\ (mkpairc\ hv')\ hv$$

$$where$$

$$mkpairc\ hv'\ vn\ ht \ = \ Primrec\ Holvar\ (mkpaircc\ vn\ ht)\ hv'$$

$$mkpaircc\ vn\ ht\ vn'\ ht' \ = \ (and\ (Eqbool\_Vn\ vn\ vn')\ (Eqbool\_Hty\ ht\ ht'))$$

Definition F.36 *The function Eqbool_Hivar with arity*

$$Eqbool\_Hivar : Holinvar \to Holinvar \to Bool$$

is defined as follows:

$$Eqbool\_Hivar\ hiv\ hiv' \ = \ Primrec\ Holinvar\ (mkpairc\ hiv')\ hv$$

$$where$$

$$mkpairc\ hiv'\ hv\ vip \ = \ Primrec\ Holinvar\ (mkpaircc\ hv\ vip)\ hiv'$$

$$mkpaircc\ hv\ vip\ hv'\ vip' \ = \ (and\ (Eqbool\_Hvar\ hv\ hv')\ (Eqbool\_Vi\ (fst\ vip)\ (fst\ vip')))$$

Definition F.37 *The function* $getindex\_Hiv : Holinvar \rightarrow Var\_index$ *is defined as follows:*

$$getindex\_Hiv\ hiv \ = \ (fst\ (snd\ hiv))$$

Definition F.38 *The function* $getblevel\_Hiv : Holinvar \rightarrow Var\_index$ *is defined as follows:*

$$getblevel\_Hiv\ hiv \ = \ (snd\ (snd\ hiv))$$

Definition F.39 *The function* $assindex\_Hiv : Holinvar \rightarrow Var\_index \rightarrow Holinvar$ *is defined as follows:*

$$assindex\_Hiv\ hiv\ vi = \ (fst\ hiv, (vi, (snd\ (snd\ hiv))))$$

Definition F.40 *The function* $assblevel\_Iv : Holinvar \rightarrow Var\_index \rightarrow Holinvar$ *is defined as follows:*

$$assblevel\_Hiv\ hiv\ vi = \ (fst\ hiv, ((fst\ (snd\ hiv)), vi))$$

Definition F.41 *The function* $addindex\_Hiv : Holinvar \rightarrow Var\_index \rightarrow Holinvar$ *is defined as follows:*

$$addindex\_Hiv\ hiv\ vi = \ (fst\ hiv, (add\_Vi\ vi\ (fst\ (snd\ hiv)), (snd\ (snd\ hiv))))$$

Definition F.42 *The function* $addblevel\_Hiv : Holinvar \rightarrow Var\_index \rightarrow Holinvar$ *is defined as follows:*

$$addblevel\_Hiv\ hiv\ vi = \ (fst\ hiv, ((fst\ (snd\ hiv)), add\_Vi\ (snd\ (snd\ hiv))\ vi))$$

Definition F.43 *The function* $addindex\_Hivl : (List\ Holinvar) \rightarrow Var\_index \rightarrow (List\ Holinvar)$ *is defined as follows:*

$$addindex\_Hivl\ hivl\ vi = \ map\ (addf\ vi)\ hivl$$

$$where$$

$$addf\ vi\ hiv \ = \ addindex\_Hiv\ hiv\ vi$$

Definition F.44 *The function* $addblevel\_Hivl : Holinvar \rightarrow Var\_index \rightarrow Holinvar$ *is defined as follows:*

$$addblevel\_Hivl\ hivl\ vi\ =\ map\ (addbf\ vi)\ hivl$$

$$where$$

$$addbf\ vi\ hiv\ =\ addblevel\_Hiv\ hiv\ vi$$

Definition F.45 *The type* $Holvar\_set$ *for a given signature* Σ *is defined as:*

$$Holvar\_set = pair\ (pair\ Var\_index\ (List\ Holinvar))(pair\ Var\_index\ (List\ Holinvar))$$

Definition F.46 *The function*

$$empty\_Hvst : Holvar\_set$$

is defined as follows:

$$empty\_Hvst\ =\ ((first\_Vi, nil\ Holinvar), (first\_Vi, nil\ Holinvar))$$

Definition F.47 *The function*

$$addfvar\_Hvst : Holvar \rightarrow Holvar\_set \rightarrow Holvar\_set$$

is defined as follows:

$$addfvar\_Hvst\ hv\ vs\ =$$

$$(((next\_Vi\ (fst\ (fst\ vs))), (cons\ Holinvar\ (mkhivar\ hv\ hvs)\ (snd\ (fst\ vs)))),$$

$$((next\_Vi\ (fst\ (snd\ vs))), (snd\ (snd\ vs))))$$

$$where$$

$$mkhivar\ hv\ hvs = (hv, ((fst\ (fst\ hvs)),\ first\_Vi))$$

Definition F.48 *The function*

$$addbvar\_Hvst : Holvar \rightarrow Holvar\_set \rightarrow Holvar\_set$$

is defined as follows:

$$addbvar\_Hvst\ hv\ vs\ =$$

$$(((fst\ (fst\ vs)), (snd\ (fst\ vs))),$$

$$((next\_Vi\ (fst\ (snd\ vs))), (cons\ Holinvar\ (mkhivar\ hv\ hvs)\ (snd\ (snd\ vs)))))$$

$$where$$

$$mkhivar\ hv\ hvs = (hv, ((fst\ (fst\ hvs)), first\_Vi))$$

Definition F.49 *The function*

$$getbindex\_Hvst : Holvar\_set \rightarrow Var\_index$$

is defined as follows:

$$getbindex\_Hvst\ hvs\ =\ fst\ (snd\ hvs)$$

Definition F.50 *The function*

$$getfindex\_Hvst : Holvar\_set \rightarrow Var\_index$$

is defined as follows:

$$getbindex\_Hvst\ hvs\ =\ fst\ (fst\ hvs)$$

Definition F.51 *The function*

$$getblevel\_Hvst : Holvar\_set \rightarrow Var\_index$$

is defined as follows:

$$getblevel\_Hvst\ hvs\ =\ substract\_Vi\ (fst\ (snd\ hvs))\ (fst\ (fst\ hvs))$$

Definition F.52 *The function*

$$getvar\_Hvst : Holvar \rightarrow Holvar\_set \rightarrow Holinvar$$

is defined as follows:

$$getvar\_Hvst\ hv\ hvs\ =$$

$$Prim\_rec\ bool\ (assblevel\_Hiv\ (getblevel\_Hvst\ hvs)\ (getfvar\_Hvst\ hv\ hvs))$$

$$(assblevel\_Hiv\ (getblevel\_Hvst\ hvs)\ (getbvar\_Hvst\ hv\ hvs))$$

$$(Eqbool\_Vi\ (fst\ (snd\ (getbvar\_Hvst\ hv\ hvs)))\ (getbindex\_Hvst\ hvs))$$

$$where$$

$$getbvar\_Hvst\ hv\ hvs\ =\ Prim\_rec\ (List\ Holinvar)\ (bvar\_notfound\ hv\ hvs)$$

$$(get\_if\_eq\ hv)\ (snd\ (snd\ hvs))$$

$$bvar\_notfound\ hv\ hvs\ =\ (hv, (getbindex\_Hvst\ hvs, first\_Vi))$$

$$getfvar\_Hvst\ hv\ hvs\ =\ Prim\_rec\ (List\ Holinvar)\ (fvar\_notfound\ hv\ hvs)$$

$$(get\_if\_eq\ hv)\ (snd\ (fst\ hvs))$$

$$fvar\_notfound\ hv\ hvs\ =\ (hv, (getbindex\_Hvst\ hvs, first\_Vi))$$

$$get\_if\_eq\ hv\ hv'\ hvl\ hvf\ =\ Prim\_rec\ Bool\ hv'\ hvf\ (Eqbool\_Hvar\ hv\ (fst\ hv'))$$

Definition F.53 *The function*

$$getvarl\_Hvst : (List\ Holvar) \rightarrow Holvar\_set \rightarrow (List\ Holinvar)$$

is defined as follows:

$$getvarl\_Hvst\ hvl\ hvs\ =\ map\ (get\_varp\ hvs)\ hvl$$

$$where$$

$$get\_varp\ hvs\ hv\ =\ getvar\_Hvst\ hv\ hvs$$

Definition F.54 *The inductive relation*

$$Is\_in\_Hivl : \Pi v : Holvar.\Pi vs : List\ Holinvar.Prop$$

is defined by the following set of constructors:

$$base\_Inhivl : \Pi hv : Holvar.\Pi hiv : Holinvar.\Pi hivl : List\ Holinvar.$$

$$\Pi eqpr : (Eqbool\_Hvar\ hv\ (fst\ hiv)) =_{bool} true.$$

$$Is\_in\_Hivl\ hv\ (cons\ hiv\ hivl)$$

$$genc\_Inhivl : \Pi hv : Holvar.\Pi hiv : Holinvar.\Pi hivl : list\ Holinvar.$$

$$\Pi pr : Is\_in\_Hivl\ hv\ hivl.$$

$$Is\_in\_Hivl\ hv\ (cons\ Holinvar\ hiv\ hivl)$$

Definition F.55 *The inductive relation*

$$Notisin\_Hivl : \Pi v : Holvar.\Pi vs : List\ Holinvar.Prop$$

is defined by the following set of constructors:

$$base\_Ninhivl : \Pi hv : Holvar.Notisin\_Hivl\ hv\ (nil\ Holinvar)$$

$$genc\_Ninhivl : \Pi hv : Holvar.\Pi hiv : Holinvar.\Pi hivl : list\ Holinvar.$$

$$\Pi eqpr : (Eqbool\_Hvar\ hv\ (fst\ hiv)) =_{bool} false.$$

$$\Pi pr : Notisin\_Hivl\ hv\ hivl.$$

$$Notisin\_Hivl\ hv\ (cons\ Holinvar\ hiv\ hivl)$$

Definition F.56 *The inductive relation*

$$Isin\_boundv\_Hvs : \Pi hv : Holvar.\Pi vs : Holvar\_set.Prop$$

is defined by the following set of constructors:

$$ctr\_Inbhvs : \Pi hv : Holvar.\Pi hvs : Holvar\_set.\Pi isinpr : Is\_in\_hivl\ hv\ (snd\ (snd\ hvs)).$$

$$Isin\_boundv\_Hvs\ hv\ hvs$$

Definition F.57 *The inductive relation*

$$Notisin\_boundv\_Hvs : \Pi hv : Holvar.\Pi vs : Holvar\_set.Prop$$

is defined by the following set of constructors:

$$ctr\_Ninbhvs : \Pi hv : Holvar.\Pi hvs : Holvar\_set.\Pi isinpr : Notisin\_hivl\ hv\ (snd\ (snd\ hvs)).$$

$$Notissin\_boundv\_Hvs\ hv\ hvs$$

Definition F.58 *The inductive relation*

$$Isin\_freev\_Hvs : \Pi hv : Holvar.\Pi vs : Holvar\_set.Prop$$

is defined by the following set of constructors:

$$ctr\_Inbhvs : \Pi hv : Holvar.\Pi hvs : Holvar\_set.\Pi isinpr : Is\_in\_hivl\ hv\ (snd\ (fst\ hvs)).$$

$$Isin\_freev\_Hvs\ hv\ hvs$$

Definition F.59 *The function*

$$addfvarl\_Hvst : (List\ Holvar) \to (Holvar\_set) \to (Holvar\_set)$$

is defined as follows:

$$addfvarl\_Hvst\ hvl\ hvs =$$

$$Prim\_rec\ \ (List\ Holvar)\ \ hvs\ \ addfvar\_aux\ \ hvl$$

$$where$$

$$addfvar\_aux\ hv\ hvl\ hvst\ =\ addfvar\_Hvst\ hv\ hvst$$

Definition F.60 *The function*

$$addbvarl\_Hvst : (List\ Holvar) \to (Holvar\_set) \to (Holvar\_set)$$

is defined as follows:

$$addbvarl\_Hvst\ hvl\ hvs =$$

$$Prim\_rec\ \ (List\ Holvar)\ \ hvs\ \ addvar\_aux\ \ hvl$$

$$where$$

$$addbvar\_aux\ hv\ hvl\ hvst\ =\ addbvar\_Hvst\ hv\ hvst$$

Definition F.61 *The function*

$$getholtypel\_Hvl : (List\ Holvar) \to Holtype\_list$$

is defined as follows:

$$getholtypel\_Hvl\ hvs\ =\ map\ (List\ Holvar)\ snd\ hvl$$

Definition F.62 *For any signature $\Sigma \in |AlgSig|$, the mutually recursive types Holterm and Holterm_list is defined by the following set of constructors:*

$$holvar\_Htrm : Holinvar \to Holterm$$

$$term\_Htrm : Term \to Holterm$$

$$appl\_Htrm : Holterm \to (List\ Holterm) \to Holterm$$

$$abstr\_Htrm : (List\ Holinvar) \to Holterm \to Holterm$$

$$forall\_Htrm : Holinvar \to Holterm \to Holterm$$

$$implies\_Htrm : Holterm \to Holterm \to Holterm$$

$$nil\_Htrm : Holterm\_list$$

$$cons\_Htrm : Holterm \to Holterm\_list \to Holterm\,list$$

F.1.3 The substitution operation

In the following, we present the substitution operation on higher-order terms which given a variable index, a higher-order term ht, a higher-order term ht' and a free higher-order variable with indexes *hiv*, returns the higher-order term which is obtained by replacing all the appearences of the variable *hiv* in *ht* by ht'. Once a higher-order term is replaced by a variable, the variable indexes of the bound variables of the higher-order term must be updated and the bound level of every variable of the higher-order term must also be updated. The first parameter of the substitution operation (the first variable index which is not assigned to the set of free variables of ht and ht') is used to determine whether a variable is free or bound.

Definition F.63 *The functions*

$$subst\_Htrm : Var\_index \to Holterm \to Holterm \to Holinvar \to Holterm$$

and

$$subst\_Htrml : Var\_index \to (List\ Holterm) \to Holterm \to Holvar \to (List\ Holterm)$$

are defined by mutual recursion as follows:

$$subst\_Htrm\ vi\ htrm\ htrm'\ hiv\ =$$

$$Prim\_rec\ Holterm\ (holvarc\ vi\ htrm'\ hiv)\ (termc\ vi\ htrm'\ hv)\ (applc\ htrm'\ hiv)$$

$$(abstrc\ htrm'\ hiv)\ (forallc\ htrm'\ hiv)\ (impliesc\ htrm'\ hiv)$$

$$(nilc\ htrm'\ hiv)\ (consc\ htrm'\ hiv)\ htrm$$

$subst\_Htrml\ vi\ htrml\ htrm'\ hiv\ =$

$Prim\_rec\ Holterm\_list\ (holvarc\ vi\ htrm'\ hiv)\ (termc\ vi\ htrm'\ hv)\ (applc\ htrm'\ hiv)$

$(abstrc\ htrm'\ hiv)\ (forallc\ htrm'\ hiv)\ (impliesc\ htrm'\ hiv)$

$(nilc\ htrm'\ hiv)\ (consc\ htrm'\ hiv)\ htrml$

$where$

$holvarc\ vi\ htrm'\ hiv\ hiv'\ =$

$Primrec\ Bool\ (update\_index\_Htrm\ vi\ (getblevel\_Hiv\ hiv')\ htrm')$

$(holvar\_Htrm\ hiv')\ (Eqbool\_Hivar\ hiv\ hiv')$

$termc\ vi\ htrm\ hiv\ trm\ =$

$term\_Htrm\ (subst\_Trm\ vi\ trm\ (coerce\_htrm\_Trm\ htrm)\ (coerce\_hiv\_Ivr\ hiv)$

$applc\ htrm'\ hiv\ htrm\ htrmf\ htrml\ =$

$appl\_Htrm\ htrmf\ (subst\_Htrml\ htrml\ htrm'\ hiv)$

$abstrc\ htrm'\ hiv\ hvl\ htrm\ htrmf\ =\ (abstr\_Htrm\ hvl\ htrmf)$

$forallc\ htrm'\ hiv\ hiv'\ htrm\ htrmf\ =\ (forall\_Htrm\ hiv\ htrmf)$

$impliesc\ htrm'\ hv\ htrm\ htrmf\ htrm'\ htrmf'\ =\ implies\_Htrm\ htrmf\ htrmf'$

$nilc\ ht\ hiv\ = nil\_Htrml$

$consc\ htrm'\ hiv\ htrm\ htrml\ htrmf\ htrmlf\ =\ cons\_Htrml\ htrmf\ htrmlf$

Definition F.64 *For any signature* $\Sigma \in |AlgSig|$, *the function*

$$coerce\_hiv\_Ivar : Holinvar \rightarrow Invar$$

is defined as follows:

$$coerce\_hiv\_Ivar\ hiv\ =\ mkpair\ Invar\ (mkpair\ Var\ (fst\ (fst\ hiv))$$

$$(Primrec\ Holtype\ s_1c\ \ldots\ s_nc\ propc\ holrelc\ nilc\ consc\ (snd\ (fst\ hiv)))\ (snd\ hiv)$$

$$anysort\ =\ s_1\_Srts$$

$$s_1c\ s_1\ =\ s_1$$

$$\vdots$$

$$s_nc\ s_n\ =\ s_n$$

$$propc\ =\ anysort$$

$$holrelc\ htyl\ s\ =\ anysort$$

$$nilc\ =\ anysort$$

$$consc\ hty\ s\ htyl\ s'\ =\ anysort$$

Definition F.65 *For any signature* $\Sigma \in |AlgSig|$*, the function*

$$coerce\_htrm\_Trm : Holterm\ \rightarrow\ Term$$

is defined as follows:

$$coerce\_htrm\_Trm\ htrm\ =\ Primrec\ Holterm\ holvarc\ termc$$

$$applc\ abstrc\ forallc\ impliesc\ nilc\ consc\ htrm$$

$$where$$

$$anysort\ =\ s_1\_Srts$$

$$anyterm\ =\ var\_s_1\_Trms_1\ (((first\_Nel\ a\_Vs), s_1\_Srts), first\_Vi)\ ()$$

$$holvarc\ hv\ =\ Primrec\ bool\ (((fst\ (fst\ hv)), (snd\ (fst\ (coerce\_hiv\_Ivar\ hv))),$$

$$(snd\ hv))$$

$$(((fst\ (fst\ hv)), anysort), (snd\ hv))$$

$$(Eqbsort\_Hty\ (snd\ (fst\ hv))$$

$termc\ trm\ =\ trm$

$applc\ ht\ htl\ trm =\ anyterm$

$abstrc\ hivl\ htrm\ trm =\ anyterm$

$forallc\ hiv\ htrm\ trm =\ anyterm$

$impliesc\ htrm_1\ htrm_2\ trm_1\ trm_2 =\ anyterm$

$nilc\ =\ nilTerm$

$consc\ ht\ htl\ tr\ tlr =\ cons\ tr\ tlr$

Definition F.66 *The function* $update\_index\_Htrm : Var\_index\ \rightarrow\ Var\_index\ \rightarrow\ Holterm\ \rightarrow\ Holterm$ *is defined as follows:*

$update\_index\_Htrm\ vi\ bl\ htrm\ =\ Primrec\ Holterm$

$(holvarc\ vi\ bl)\ (termc\ vi\ bl)\ applc\ (abstrc\ bl)\ (forallc\ bl)\ impliesc\ nilc\ consc\ htrm$

$where$

$holvarc\ vi\ bl\ hiv\ =\ Primrec\ bool\ (addblevel\_Hiv\ bl\ hiv)$

$(addblevel\_Hiv\ bl\ (addindex\_Hiv\ bl\ hiv))\ (Ltbool\_Vi\ (getindex\_Hiv\ hiv)\ vi)$

$termc\ vi\ bl\ trm\ =\ update\_index\_Trm\ vi\ bl\ trm$

$applc\ ht\ htl\ htf\ htlf\ =\ appl\_Htrm\ htf\ htlf$

$abstrc\ bl\ hivl\ ht\ htf\ =$

$abstr\_Htrm\ (addblevel\_Hivl\ bl\ (addindex\_Hivl\ bl\ hivl))\ htf$

$forallc\ bl\ hiv\ ht\ htf\ =$

$forall\_Htrm\ (addblevel\_Hiv\ bl\ (addindex\_Hiv\ bl\ hiv))\ htf$

$impliesc\ ht\ ht'\ htf\ htf'\ =\ implies\_Htrm\ htf\ htf'$

$nilc\ =\ nil\_Htrml$

$consc\ ht\ htl\ htf\ htlf\ =\ cons\_Htrml\ htf\ htlf$

Definition F.67 *The function* $update\_index\_Trm : Var\_index\ \rightarrow\ Var\_index$

$\rightarrow$ *Term* $\rightarrow$ *Term is defined as follows:*

$$update\_index\_Trm\ vi\ bl\ trm\ =\ Primrec\,Term\ (trms1c\ vi\ bl)\ \ldots\ (trmsnc\ vi\ bl)$$

$$trms1c\ vi\ bl\ trms1\ =\ trm\_s_1\_Trm\ (update\_index\_Trm\_s_1\ vi\ bl\ trms1)$$

$$\vdots$$

$$trmsnc\ vi\ bl\ trmsn\ =\ trm\_s_n\_Trm\ (update\_index\_Trm\_s_n\ vi\ bl\ trmsn)$$

Definition F.68 *For any signature* $\Sigma \in |AlgSig|$ *and for any sort* $s \in Sorts(\Sigma)$*, the function* $update\_index\_Trm\_s : Var\_index\ \rightarrow\ Var\_index\ \rightarrow\ Term\_s$ $\rightarrow Term\_s$ *is defined as follows:*

$$update\_index\_Trm\_s\ vi\ bl\ trms\ =\ Primrec\,Term\_s\ (varc\ vi\ bl)\ func\_1\ \ldots\ func\_n$$

$$funovc\_1\ \ldots\ funovc\_m$$

where

$$varc\ vi\ bl\ iv\ = Primrec\ bool\ (addblevel\_Iv\ bl\ iv)$$

$$(addblevel\_Iv\ bl\ (addindex\_Iv\ bl\ iv))$$

$$(Ltbool\_Vi\ vi\ (getindex\_Iv\ iv))$$

$$func\_1\,trm\_11\ \ldots\ trm\_1n_1\,trmf\_11\ \ldots\ trmf\_1n_1\ =$$

$$f_1\_Trms\ trmf_{11}\ \ldots\ trmf\_1n_1$$

$$\vdots$$

$$func\_n\,trm\_n1\ \ldots\ trm\_nn_n\,trmf\_n1\ \ldots\ trmf\_nn_n\ =$$

$$f_n\_Trms\ trmf\_n1\ \ldots\ trmf\_nn_n$$

$$funovc\_1\,trm\_11\ \ldots\ trm\_1m_1\,trmf\_11\ \ldots\ trmf\_1m_1\ =$$

$$g_1\_r_{11}\_\ldots\_s_{1m_1}\_s_{m_1}\_Trms\ trmf\_11\ \ldots\ trmf\_1m_1$$

$$\vdots$$

$$funovc\_m\,trm\_m1\ \ldots\ trm\_mm_m\,trmf\_m1\ \ldots\ trmf\_mm_m\ =$$

$$g_m\_r_{m1}\_\ldots\_r_{mm_m}\_r_{m_m}\_Trms\ trmf\_m1\ \ldots\ trmf\_mm_m$$

$$where\ f_1 : s_{11} \times \ldots \times s_{1n_1} \rightarrow s_{n_1}, \ldots, f_n : s_{n1} \times \ldots \times s_{nn_n} \rightarrow s_{n_n},$$

$$g_1 : r_{11} \times \ldots \times r_{1m_1} \rightarrow r_{m_1}, g_n : r_{m1} \times \ldots \times r_{mm_m} \rightarrow r_{m_m},$$

Definition F.69 *The function*

$$subst\_Trm : Var\_index \to Term \to Term \to Invar \to Term$$

is defined as follows:

$$subst\_Trm\ vi\ trm\ trm'\ hv\ =$$

$$Primrec\ Term\ (ts\_1c\ vi\ trm'\ v)\ \ldots\ (ts\_nc\ vi\ trm'\ v)\ trm$$

$$\begin{aligned} ts\_1c\ vi\ trm'\ v\ trms\ &=\ (trm\_s_1\_Trm\ (subst\_Trms\_s_1\ vi\ trms\ trm'\ v')) \\ &\vdots \\ ts\_nc\ vi\ trm'\ v\ trms\ &=\ (trm\_s_n\_Trm\ (subst\_Trms\_s_n\ vi\ trms\ trm'\ v')) \end{aligned}$$

Definition F.70 *For any signature $\Sigma \in |AlgSig|$ and for any sort $s \in Sorts(\Sigma)$,*

$$subst\_Trm\_s : Var\_index \to Term\_s \to Term \to Invar \to Term\_s$$

is defined by mutual recursion as follows:

$$subst\_Trm\_s\ vi\ trms\ trm\ v\ =\ Primrec\ Term\_s\ (varc\ vi\ trm\ v)\ (func\_1\ trm\ v)\ \ldots$$

$$(func\_n\ trm\ v)\ (funovc\_1\ trm\ v)\ \ldots\ (funovc\_m\ trm\ v)\ trms$$

$where$

$varc\ vi\ trm\ v\ v' \ = \ Primrec\ Bool$

$(update\_index\_Trm\_s\ vi\ (getblevel\_Iv\ v')\ (coerce\_trm\_Trms\ trm))$

$(var\_s\_Trms\ v')\ (Eqbool\_Ivar\ v\ v')$

$func\_1\ trm\ v\ trm\_11\ \ldots\ trm\_1n_1\ trmf\_11\ \ldots\ trmf\_1n_1 \ =$

$f_1\_Trms\ trmf_{11}\ \ldots\ trmf\_1n_1$

$\vdots$

$func\_n\ trm\ v\ trm\_n1\ \ldots\ trm\_nn_n\ trmf\_n1\ \ldots\ trmf\_nn_n \ =$

$f_n\_Trms\ trmf\_n1\ \ldots\ trmf\_nn_n$

$funovc\_1\ trm\ v\ trm\_11\ \ldots\ trm\_1m_1\ trmf\_11\ \ldots\ trmf\_1m_1 \ =$

$g_1\_r_{11}\_\ldots\_s_{1m_1}\_s_{m_1}\_Trms\ trmf\_11\ \ldots\ trmf\_1m_1$

$\vdots$

$funovc\_m\ trm\ v\ trm\_m1\ \ldots\ trm\_mm_m\ trmf\_m1\ \ldots\ trmf\_mm_m \ =$

$g_m\_r_{m1}\_\ldots\_r_{mm_m}\_r_{m_m}\_Trms\ trmf\_m1\ \ldots\ trmf\_mm_m$

$where\ f_1 : s_{11} \times \ldots \times s_{1n_1} \to s_{n_1}, \ldots, f_n : s_{n1} \times \ldots \times s_{nn_n} \to s_{n_n},$

$g_1 : r_{11} \times \ldots \times r_{1m_1} \to r_{m_1}, g_n : r_{m1} \times \ldots \times r_{mm_m} \to r_{m_m},$

Definition F.71 *For any signature $\Sigma \in |AlgSig|$ and for any sort $s \in Sorts(\Sigma)$, the functions*

$$coerce\_trm\_Trm\_s : Term \to Term\_s$$

(one for each sort s) is defined as follows:

$$coerce\_trm\_Trm\_s\ t \ = \ Primrec\ Term\ trm\_s_1c \ldots trm\_sc \ldots trm\_s_nc\ t$$

$where$

$$anysvar \ = \ var\_s\_Trms\ (mkpair\ Var\ Var\_index$$

$$(mkpair\ Var\_name\ Sorts\ (first\_Nel\ a\_Vs)\ s\_Srts)\ first\_Vi)$$

$$trm\_s_1c\ ts_1 \ = \ anysvar$$
$$\vdots$$
$$trm\_sc\ ts \ = \ trm\_s\_Trm\ ts$$

$$trm\_s_nc\ ts_n \ = \ anysvar$$

Definition F.72 *The function*

$$substhvl\_Htrm : Holterm \to (Holterm\_list) \to (List\ Holvar) \to Holterm$$

is defined as follows:

$$substhvl\_Htrm\ htrm\ htl\ hvl \ =$$

$$Primrec\ (List\ (Pair\ Holterm\ Holvar))\ bcl\ gcl\ (join\ Holterm\ Holvar\ htl\ hvl)$$

$where$

$$bcl \ = \ htrm$$

$$gcl\ hthv\ hthvl\ htrm \ = \ subst\_Htrm\ htrm\ (fst\ hthv)\ (snd\ hthv)$$

Definition F.73 *The mutually recursive inductive relation*

$$Wfhterm : Holvar\_set \to Holterm \to Holtype \to Prop$$

and

$$Wfhtermlist : Holvar\_set \to (Holterm\_list) \to (Holtype\_list) \to Prop$$

are defined by the following set of constructors:

$\{ass1\_tr : \Pi vs : Holvar\_set.\Pi hv : Holvar.\Pi pr : Notisin\_boundv\_Hvs\ hv\ vs.$

$\Pi prin : Isin\_freev\_Hvs\ hv\ vs.$

$Wfhterm\ \ vs\ \ (holvar\_Htrm\ (getvar\_Hvst\ hv\ vs))\ \ (snd\ hv)\} \cup$

$\{ass2\_tr : \Pi vs : Holvar\_set.\Pi hv : Holvar.\Pi pr : Isin\_boundv\_Hvs\ hv\ vs.$

$Wfhterm\ \ vs\ \ (holvar\_Htrm\ (getvar\_Hvst\ hv\ vs))\ \ (snd\ hv)\} \cup$

$\{appl\_f\_s\_tr : \Pi vs : Holvar\_set.\Pi t_1 : Term\_s_1. \ldots .\Pi t_n : Term\_s_n.$

$\Pi wft_1 : Wfhterm\ vs\ (term\_Htrm\ t_1)\ s_1\_Holt. \ldots .$

$\Pi wft_n : Wfhterm\ vs\ (term\_Htrm\ t_n)\ s_n\_Holt.$

$Wfhterm\ vs\ (term\_Htrm\ f\_Trms\ t_1\ \ldots\ t_n)\ s\_Holt\ |$

$f : s_1 \times \cdots \times s_n \to s\ and\ f\ \ is\ not\ overloaded\ in\ \Sigma\} \cup$

$\{appl\_f\_s_1\_\ldots\_s_n\_s\_tr : \Pi vs : Holvar\_set.\Pi t_1 : Term\_s_1. \ldots .\Pi t_n : Term\_s_n.$

$\Pi wft_1 : Wfhterm\ vs\ (term\_Htrm\ t_1)\ s_1\_Holt. \ldots .$

$\Pi wft_n : Wfhterm\ vs\ (term\_Htrm\ t_n)\ s_n\_Holt.$

$Wfhterm\ vs\ (term\_Htrm\ f\_s_1\_\ldots\_s_n\_s\_Trms\ t_1 \ldots t_n)\ s\_Holt\ |$

$f : s_1 \times \cdots \times s_n \to s\ and\ f\ \ is\ overloaded\ in \Sigma\} \cup$

$\{\lambda abs\_tr : \Pi vs : Holvar\_set.\Pi hvl : List\ Holvar.$

$\Pi norep : Norep\_list\ Holvar\ Eqbool\_Hvar\ hvl.$

$\Pi nem : Not\_emptyl\ Holvar\ hvl.\Pi\phi : Holterm.$

$\Pi wfprop : Wfhterm\ (addbvarl\_Hvst\ hvl\ vs)\ \phi\ prop\_Holt.$

$Wfhterm\ vs\ (abstr\_Htrm\ (getvarl\_Hvst\ hvl$

$(addbvarl\_Hvst\ hvl\ vs))\ \phi)\ (getholtypel\_Hvarl\ hvl),$

$\lambda appl\_tr : \Pi vs : Holvar\_set.\Pi t : Holterm.\Pi tl : Holterm\_list.\Pi holtl : Holtype\_list$

$\Pi nem : Not\_emptyl\ Holterm\ tl.\Pi prt : Wfhterm\ vs\ t\ (holrel\_Holt\ holtl).$

$\Pi prtl : Wfhtermlist\ vs\ tl\ holtl.Wfhterm\ vs\ (appl\_Htrm\ t\ tl)\ prop\_Holt,$

$forall\_tr : \Pi vs : Holvar\_set.\Pi hv : Holvar.\Pi\phi : Holterm.$

$\Pi prp : Wfhterm\ (addbvar\_Hvst\ hv\ vs)\ \phi\ prop\_Holt.Wfhterm\ vs$

$(forall\_Htrm\ (getvar\_Hvst\ hv\ (addbvar\_Hvst\ hv\ vs))\ \phi)\ prop\_Holt,$

$implies\_tr : \Pi vs : Holvar\_set.\Pi\phi, \phi' : Holterm.$

$\Pi pr : Wfhterm\ vs\ \phi\ prop\_Holt.\Pi pr' : Wfhterm\ vs\ \phi'\ prop\_Holt.$

$Wfhterm\ vs\ (implies\_Htrm\ \phi\ \phi')\ prop\_Holt\}$

$\{nil\_Whtl : \Pi vs : Holvar\_set.$

$Wfhtermlist\ vs\ (nil\_Htrm)\ (nil\_Holt),$

$cons\_Whtl : \Pi vs : Holvar\_set.\Pi htr : Holterm.\Pi htrl : Holterm\_list.$

$\Pi hty : Holtype.\Pi htyl : Holtype\_list.$

$\Pi prt : Wfhterm\ vs\ htr\ hty.\Pi prtl : Wfhtermlist\ vs\ htrl\ htyl.$

$Wfhtermlist\ vs\ (consHtrm\ htr\ htrl)\ (cons\_Holt\ hty\ htyl)\ \}$

F.1.4 Adequacy of syntax and the proof system

In the following, we present the encoding and decoding functions of types, list of types, variables, names, list of variables, variable set, list of indexed variables, set of higher-order variables, higher-order terms, the proof of adequacy of syntax and finally the encoding of the proof system and its proof of adequacy.

Definition F.74 *For any signature* $\Sigma = (S, Op) \in |AlgSig|$*, the encoding function of types* ϵ_τ *with arity*

$$\epsilon_\tau : Types_{HOL}(\Sigma) \rightarrow Holtype$$

is inductively defined as follows:

$$\epsilon_\tau\ s\ =\ s\_Holt$$

where s ranges over $Sorts(\Sigma)$

$$\epsilon_\tau\ tl\ =\ holrel\_Holt\ (\epsilon_{\tau l}\ tl)$$

Definition F.75 *For any signature* $\Sigma = (S, Op) \in |AlgSig|$*, the decoding function of types* ϵ_τ^{-1} *with arity*

$$\epsilon_\tau^{-1} : Holtype \rightarrow Types_{HOL}(\Sigma)$$

$$\epsilon_\tau^{-1}\ (s\_Holt\ s)\ =\ s$$

$$\epsilon_\tau^{-1}\ (holrel\_Holt\ [\tau_1, \ldots, \tau_n])\ =\ (\epsilon_{\tau l}^{-1}\ [\tau_1, \ldots, \tau_n])$$

Definition F.76 *For any signature* $\Sigma = (S, Op) \in |AlgSig|$*, the encoding function of list of types* $\epsilon_{\tau l}$ *with arity*

$$\epsilon_{\tau l} : (List\ Types_{HOL}(\Sigma)) \rightarrow Holtype\_list$$

is inductively defined as follows:

$$\epsilon_{\tau l}\ []\ =\ nil\_Holt$$

$$\epsilon_{\tau l}\ (cons\ Types_{HOL}(\Sigma)\ hty\ htyl)\ =\ cons\_Holt\ (\epsilon_\tau\ hty)\ (\epsilon_{\tau l}\ htyl)$$

Definition F.77 *For any signature $\Sigma = (S, Op) \in |AlgSig|$, the decoding function of list of types $\epsilon_{\tau l}^{-1}$ with arity*

$$\epsilon_{\tau l}^{-1} : Holtype\_list \to Types_{HOL}(\Sigma)$$

$$\epsilon_{\tau l}^{-1}\ (nil\_Holt)\ =\ \mathbf{Prop}$$

$$\epsilon_{\tau l}^{-1}\ (cons\_Holt\ hty\ htyl)\ =\ cons\ Types_{HOL}(\Sigma)\ (\epsilon_\tau^{-1}\ hty)\ (\epsilon_{\tau l}^{-1}\ htyl)$$

Definition F.78 *For any signature $\Sigma = (S, Op) \in |AlgSig|$ and for any type $\tau \in Types_{HOL}(\Sigma)$, the encoding function of variable names ϵ_{vn} with arity*

$$\epsilon_{vn} : X_\tau \to Var\_name$$

is inductively defined as follows:

$$\epsilon_{vn}\ \text{“}c''\ =\ first\_Nel\ c\_Vs$$

$$\epsilon_{vn}\ c.str\ =\ cons\_Nel\ Var\_symbols\ c\ (\epsilon_{vn}\ str)$$

Notation*: We assume that the denumerable set of variables X_τ is denoted by alphanumeric non-empty strings plus the symbols $\$, \_, '$.*

Definition F.79 *For any signature $\Sigma = (S, Op) \in |AlgSig|$ and for any type $\tau \in Types_{HOL}(\Sigma)$, the decoding function of variable names ϵ_{vn} with arity*

$$\epsilon_{vn}^{-1} : Var\_name \to X_\tau$$

is inductively defined as follows:

$$\epsilon_{vn}^{-1}\ (first\_Nel\ c\_Vs)\ =\ \text{“}c''$$

$$\epsilon_{vn}^{-1}\ cons\_Nel\ Var\_symbols\ c\ (\epsilon_{vn}\ str)\ =\ c.str$$

Definition F.80 *For any signature $\Sigma = (S, Op) \in |AlgSig|$ and for any type τ, the encoding function of list of variables ϵ_{hvl} with arity*

$$\epsilon_{hvl} : [(X, Types_{HOL}(\Sigma))] \to (List\ Holvar)$$

is inductively defined as follows

$$\epsilon_{hvl}\ []\ =\ nil\ Holvar$$

$$\epsilon_{hvl}\ (cons\ (x_\tau, \tau)\ hvl)\ =\ cons\ Holvar\ (\epsilon_{vn}\ x, \epsilon_\tau\ \tau)\ (\epsilon_{hvl}\ hvl)$$

Definition F.81 *For any signature* $\Sigma = (S, Op) \in |AlgSig|$ *and for any type* τ*, the decoding function of list of variables* ϵ_{hvl} *with arity*

$$\epsilon_{hvl}^{-1} : [(X, Types_{HOL}(\Sigma))] \rightarrow (List\ Holvar)$$

is inductively defined as follows:

$$\epsilon_{hvl}^{-1}\ (nil\ Holvar)\ =\ []$$

$$\epsilon_{hvl}^{-1}\ (cons\ Holvar\ (x, \tau)\ hvl)\ =\ cons\ \ (\epsilon_{vn}^{-1}\ x, \epsilon_{\tau}^{-1}\ \tau)\ (\epsilon_{hvl}^{-1}\ hvl)$$

Definition F.82 *For any signature* $\Sigma = (S, Op) \in |AlgSig|$*, the encoding function of variable set* ϵ_{vs} *with arity*

$$\epsilon_{vs} : ([X], [X]) \rightarrow (Holvar\_set)$$

is inductively defined as follows:

$$\epsilon_{vs}\ (hvl, hvl')\ =\ \epsilon_{vsb}\ hvl'\ (\epsilon_{vsf}\ hvl\ empty\_Hvst)$$

$$where$$

$$\epsilon_{vsb}\ []\ hvs\ =\ hvs$$

$$\epsilon_{vsb}\ (cons\ (x, \tau)\ hvl)\ hvs\ =$$

$$\epsilon_{vsb}\ hvl\ (addbvar\ (\epsilon_{vn}\ x, \epsilon_{\tau} \tau)\ hvs)$$

$$\epsilon_{vsf}\ []\ hvs\ =\ hvs$$

$$\epsilon_{vsf}\ (cons\ (x, \tau)\ hvl)\ hvs\ =$$

$$\epsilon_{vsb}\ hvl\ (addfvar\ (\epsilon_{vn}\ x, \epsilon_{\tau} \tau)\ hvs)$$

Definition F.83 *For any signature* $\Sigma = (S, Op) \in |AlgSig|$ *and for any type* τ*, the decoding function of variable set* ϵ_{vs}^{-1} *with arity*

$$\epsilon_{vs}^{-1} : (Holvar\_set) \rightarrow [X]$$

is defined as follows:

$$\epsilon_{vs}^{-1}\ ((vi, hivl), (vi', hivl'))\ =\ (\epsilon_{hivl}^{-1}\ hivl, \epsilon_{hivl}^{-1}\ hivl')$$

Definition F.84 *For any signature* $\Sigma = (S, Op) \in |AlgSig|$ *and for any type* τ*, the decoding function of list of indexed variables* ϵ_{hivl}^{-1} *with arity*

$$\epsilon_{hivl}^{-1} : (List\ \ Holinvar) \rightarrow [X]$$

is inductively defined as follows:

$$\epsilon_{hivl}^{-1}\ (nil\ Holinvar)\ =\ []$$

$$\epsilon_{hivl}^{-1}\ (cons\ Holinvar\ ((x,\tau), vi)\ hivl)\ =\ cons\ \ (\epsilon_{vn}^{-1}\ x, \epsilon_{\tau}^{-1}\ \tau)\ (\epsilon_{hivl}^{-1}\ hivl)$$

Definition F.85 *For any signature* $\Sigma = (S, Op) \in |AlgSig|$ *and for any sort* s*, the encoding function of terms* ϵ_t *with arity*

$$\epsilon_t : Holvar\_set \to Term_{\Sigma,s}(X) \to Term\_s$$

is inductively defined as follows:

$$\epsilon_t\ vs\ x_s\ =\ var\_s\_Trms\ ((\epsilon_{vn}\ x, s\_Srts), snd\ (getvar\_Hvst\ (\epsilon_{vn}\ x, s\_Holt)\ vs))$$

$$\epsilon_t\ vs\ f(t_1, \dots, t_n)\ =$$

$$f\_Trms\ (\epsilon_t\ vs\ t_1)\ \dots\ (\epsilon_t\ vs\ t_n),\ \ if\ f\ is\ not\ overloaded\ in\ \Sigma$$

$$f\_s_1\_\dots\_s_n\_s\_Trms\ (\epsilon_t\ vs\ t_1)\ \dots\ (\epsilon_t\ vs\ t_n),\ \ if\ f\ is\ overloaded\ in\ \Sigma$$

Definition F.86 *For any signature* $\Sigma = (S, Op) \in |AlgSig|$ *and for any sort* s*, the decoding function of terms* ϵ_t^{-1} *with arity*

$$\epsilon_t^{-1} : Holvar\_set \to Term\_s \to Term_{\Sigma,s}(X)$$

is inductively defined as follows:

$$\epsilon_t^{-1}\ vs\ (var\_s\_Trms\ iv)\ =\ (\epsilon_{vn}^{-1}\ (fst\ (fst\ iv)))_s$$

$$\epsilon_t^{-1}\ vs\ (f\_trms\ t_1\ \dots\ t_n)\ =\ f(\epsilon_t^{-1}\ vs\ t_1, \dots, \epsilon_t^{-1}\ vs\ t_n)$$

$$\epsilon_t^{-1}\ vs\ (f\_s_1\_\dots\_s_n\_s\_trms\ t_1\ \dots\ t_n)\ =\ f(\epsilon_t^{-1}\ vs\ t_1, \dots, \epsilon_t^{-1}\ vs\ t_n)$$

Definition F.87 *For any signature* $\Sigma = (S, Op) \in |AlgSig|$*, the encoding function of higher-order terms* ϵ_{ht} *with arity*

$$\epsilon_{ht} : Holvar\_set \to Term_{HOL}(\Sigma) \to Holterm$$

is inductively defined as follows:

$$\epsilon_{ht}\ vs\ x_\tau\ =\ holvar\_Htrm\ (getvar\_Hvst\ (\epsilon_{vn}\ x, \epsilon_\tau\ \tau)\ vs)$$

$$\epsilon_{ht}\ vs\ f(t_1, \ldots, t_n)\ =\ term\_Htrm\ (\epsilon_t\ vs\ f(t_1, \ldots, t_n))$$

$$\epsilon_{ht}\ vs\ \lambda(x_1 : \tau_1, \ldots, x_n : \tau_n).\phi\ =$$

$$abstr\_Htrm\ (getvarl\_Hvst\ (\epsilon_{hvl}\ [(x_1, \tau_1), \ldots, (x_n, \tau_n)])$$

$$(addbvarl\_Hvst\ (\epsilon_{hvl}\ [(x_1, \tau_1), \ldots, (x_n, \tau_n)])\ vs)$$

$$(\epsilon_{ht}\ (addbvarl\_Hvst\ (\epsilon_{hvl}\ [(x_1, \tau_1), \ldots, (x_n, \tau_n)])\ vs)\ \phi)$$

$$\epsilon_{ht}\ vs\ t\ (t_1, \ldots, t_n)\ =\ appl\_Htrm\ (\epsilon_{ht}\ vs\ t)\ (\epsilon_{htl}\ vs\ [t_1, \ldots, t_n])$$

$$\epsilon_{ht}\ vs\ \forall x : \tau.\phi\ =\ forall\_Htrm\ (getvar\_Hvst\ (\epsilon_{vn}\ x_n, \epsilon_\tau\ \tau_n)$$

$$(addbvar\_Hvst\ (\epsilon_{vn}\ x_n, \epsilon_\tau\ \tau_n)\ vs))$$

$$(\epsilon_{ht}\ (addbvar\_Hvst\ (\epsilon_{vn}\ x_n, \epsilon_\tau\ \tau_n)\ vs)\ \phi)$$

$$\epsilon_{ht}\ vs\ \phi \supset \phi'\ =\ implies\_Htrm\ (\epsilon_{ht}\ vs\ \phi)\ (\epsilon_{ht}\ vs\ \phi')$$

Definition F.88 *For any signature $\Sigma = (S, Op) \in |AlgSig|$, the decoding function of higher-order terms ϵ_{ht}^{-1} with arity*

$$\epsilon_{ht}^{-1} : Holvar\_set \rightarrow Holterm \rightarrow Term_{HOL}(\Sigma)$$

is inductively defined as follows:

$$\epsilon_{ht}^{-1}\ vs\ (holvar\_Htrm\ hiv)\ =\ (\epsilon_{vn}^{-1}\ (fst\ (fst\ hiv)))_{\epsilon_\tau^{-1}\ (snd\ (fst\ hiv))}$$

$$\epsilon_{ht}^{-1}\ vs\ (term\_Htrm\ t)\ =\ \epsilon_t^{-1}\ vs\ t$$

$$\epsilon_{ht}^{-1}\ vs\ (appl\_Htrm\ ht\ htl) =\ (\epsilon_{ht}^{-1}\ vs\ ht)\ (\epsilon_{htl}^{-1}\ vs\ htl)$$

$$\epsilon_{ht}^{-1}\ vs\ (abstr\_Htrm\ hinvl\ htrm) =$$

$$\lambda(\epsilon_{hinvl}^{-1}\ hinvl).(\epsilon_{ht}^{-1}\ (addbvarl\_Hvst\ hvl\ vs)\ htrm)$$

$$\epsilon_{ht}^{-1}\ vs\ (forall\_Htrm\ hiv\ htrm) =\ \forall \epsilon_{vn}^{-1}\ (fst\ (fst\ hiv)) : \epsilon_\tau^{-1}\ (snd\ (fst\ hv)).$$

$$(\epsilon_{ht}^{-1}\ (addbvar\_Hvst\ hv\ vs)\ htrm)$$

$$\epsilon_{ht}^{-1}\ vs\ (implies\_Htrm\ htrm\ htrm')\ =\ (\epsilon_{ht}^{-1}\ vs\ htrm)\ \supset\ (\epsilon_{ht}^{-1}\ vs\ htrm')$$

Definition F.89 *For any signature $\Sigma = (S, Op) \in |AlgSig|$, the encoding function of list of higher-order terms ϵ_{htl} with arity*

$$\epsilon_{htl} : [Term_{HOL}(\Sigma)] \ \rightarrow \ (Holterm\_list)$$

is inductively defined as follows:

$$\epsilon_{htl} \ [] \ = \ nil\_Htrm$$

$$\epsilon_{htl} \ (cons \ ht \ htl) \ = \ cons\_Htrm \ (\epsilon_{ht} \ ht) \ (\epsilon_{htl} \ htl)$$

Definition F.90 *For any signature $\Sigma = (S, Op) \in |AlgSig|$, the decoding function of list of higher-order terms ϵ_{htl}^{-1} with arity*

$$\epsilon_{htl}^{-1} : (Holterm\_list) \ \rightarrow \ [Term_{HOL}(\Sigma)]$$

is inductively defined as follows:

$$\epsilon_{htl}^{-1} \ (nil\_Htrm) \ = \ []$$

$$\epsilon_{htl}^{-1} \ (cons\_Htrm \ ht \ htl) \ = \ cons \ (\epsilon_{ht}^{-1} \ ht) \ (\epsilon_{htl}^{-1} \ htl)$$

Definition F.91 *The encoding function of derivations of well typed terms ϵ_{td} which given a signature $\Sigma \in |AlgSig|$ and a closed derivation in $\Delta_{\Pi_{HOL}}(X \blacktriangleright \phi : \tau)$ returns a proof of the proposition*

$$Wfhterm \ (\epsilon_{vs} \, X) \ \ (\epsilon_{ht} \ (\epsilon_{vs} \, X) \ \phi) \ (\epsilon_{\tau} \ \tau)$$

is inductively defined by closed derivations as follows:

$$\begin{aligned}
&\epsilon_{td} \ Ass1((X, X') \blacktriangleright x : \tau) = \\
&\quad ass_tr(\epsilon_{vs} \ (X, X')) \ encx \\
&\qquad (ctr_Ninbhvs \ encx \ (\epsilon_{vs} \ (X, X')) \\
&\qquad (genc_Ninhivl \ encx \ (getvar_Hvst \ enchv'_n \\
&\qquad\quad (\epsilon_{vs} \ [hv'_1, \ldots, hv'_n]))
\end{aligned}$$

$(fst\ (\epsilon_{vs}\ [hv'_1, \ldots, hv'_{n-1}]))$

$\lambda P : bool \rightarrow Prop.\lambda pr : P\ false.pr$

$(\ldots(genc\_Ninhivl\ encx\ (getvar\_Hvst\ \ enchv'_n\ \epsilon_{vs}\ [])$

$(fst\ (\epsilon_{vs}\ []))$

$(base\_Ninhivl\ encx))\ldots)))$

$(ctr\_Infhvs\ encx\ (\epsilon_{vs}\ (X, X'))$

$(genc\_Inhivl\ encx\ (getvar\_Hvst\ enchv_n$

$\quad (\epsilon_{vs}\ [hv_1, \ldots, hv_{i-1}, (x, \tau), hv_i, \ldots, hv_n]))$

$(fst\ (\epsilon_{vs}\ [hv_1, \ldots, hv_{i-1}, (x, \tau), hv_i, \ldots, hv_{n-1}]))$

$(\ldots(genc\_Inhivl\ encx\ (getvar\_Hvst\ enchv_i$

$\quad (\epsilon_{vs}\ [hv_1, \ldots, hv_{i-1}, (x, \tau), hv_i]))$

$(fst\ (\epsilon_{vs}\ [hv_1, \ldots, hv_{i-1}, (x, \tau)]))$

$(base\_Hivs\ encx\ (getvar\_Hvst\ encx$

$\quad (\epsilon_{vs}\ [hv_1, \ldots, hv_{i-1}, (x, \tau)]))$

$(fst\ (\epsilon_{vs}\ [hv_1, \ldots, hv_{i-1}, (x, \tau)]))$

$(\lambda P : bool \rightarrow Prop.\lambda pr : P\ true.pr)))\ldots)))$

$where\ X = [hv_1, \ldots, hv_{i-1}, (x, \tau), hv_i, \ldots, hv_n]$

$\quad encx\ =\ mkpair\ Holvar\ (\epsilon_{vn}\ x)\ (\epsilon_\tau\ \tau)$

$\quad enchv_n\ =\ mkpair\ Holvar\ (fst\ hv_n)\ (snd\ hv_n)$

$\quad enchv_i\ =\ mkpair\ Holvar\ (fst\ hv_i)\ (snd\ hv_i)$

$X' = [hv'_1, \ldots, hv'_n]$

$$enchv'_n \; = \; mkpair\ Holvar\ (fst\ hv'_n)\ (snd\ hv'_n)$$
$$\vdots$$
$$enchv'_1 \; = \; mkpair\ Holvar\ (fst\ hv'_1)\ (snd\ hv'_1)$$

$\epsilon_{td}\ Ass2((X, X') \blacktriangleright x : \tau) =$
$ass2\_tr(\epsilon_{vs}\ (X, X'))\ encx$

$(ctr\_Inbhvs\ encx\ (\epsilon_{vs}\ (X, X'))$

$(genc\_Inhivl\ encx\ (getvar\_Hvst\ enchv'_n$

$(\epsilon_{vs}\ [hv'_1, \ldots, hv'_{i-1}, (x, \tau), hv'_i, \ldots, hv'_n]))$

$(fst\ (\epsilon_{vs}\ [hv'_1, \ldots, hv'_{i-1}, (x, \tau), hv'_i, \ldots, hv'_{n-1}]))$

$(\ldots (genc\_Inhivl\ encx\ (getvar\_Hvst\ enchv'_i$

$(\epsilon_{vs}\ [hv'_1, \ldots, hv'_{i-1}, (x, \tau), hv'_i]))$

$(fst\ (\epsilon_{vs}\ [hv'_1, \ldots, hv'_{i-1}, (x, \tau)]))$

$(base\_Hivs\ encx\ (getvar\_Hvst\ encx$

$(\epsilon_{vs}\ [hv'_1, \ldots, hv'_{i-1}, (x, \tau)]))$

$(fst\ (\epsilon_{vs}\ [hv'_1, \ldots, hv'_{i-1}, (x, \tau)]))$

$(\lambda P : bool \rightarrow Prop.\lambda pr : P\ true.pr))) \ldots)))$

$where\ X' = [hv'_1, \ldots, hv'_{i-1}, (x, \tau), hv'_i, \ldots, hv'_n]$

$$encx \; = \; mkpair\ Holvar\ (\epsilon_{vn}\ x)\ (\epsilon_\tau\ \tau)$$

$$enchv'_n \; = \; mkpair\ Holvar\ (fst\ hv'_n)\ (snd\ hv'_n)$$

$$enchv'_i \; = \; mkpair\ Holvar\ (fst\ hv'_i)\ (snd\ hv'_i)$$

$$\epsilon_{td}\ Appl((X, X') \blacktriangleright f(t_1, \ldots, t_n) : s, [\delta_1, \ldots, \delta_n]) =$$

$$appl\_f\_s\_tr\ (\epsilon_{vs}\ X)\ (\epsilon_t\ (\epsilon_{vs}\ X)\ t_1) \ldots (\epsilon_t\ (\epsilon_{vs}\ X)\ t_n)$$

$$(\epsilon_{td}\ \delta_1) \ldots (\epsilon_{td}\ \delta_n)$$

$$where\ f : s_1 \times \ldots \times s_n \rightarrow s\ \ is\ not\ overloaded\ in \Sigma,$$

$$\delta_1 \in \Delta_{\Pi_{HOL}}((X, X') \blacktriangleright t_1 : s_1), \ldots, \delta_n \in \Delta_{\Pi_{HOL}}((X, X') \blacktriangleright t_n : s_n)$$

$$\epsilon_{td}\ Appl((X, X') \blacktriangleright f(t_1, \ldots, t_n) : s, [\delta_1, \ldots, \delta_n]) =$$

$$appl\_f\_s_1\_\ldots\_s_n\_s\_tr\ (\epsilon_{vs}\ X)\ (\epsilon_t\ (\epsilon_{vs}\ X)\ t_1) \ldots (\epsilon_t\ (\epsilon_{vs}\ X)\ t_n)$$

$$(\epsilon_{td}\ \delta_1) \ldots (\epsilon_{td}\ \delta_n)$$

$$where\ \ f : s_1 \times \ldots \times s_n \rightarrow s\ \ is\ overloaded\ in \Sigma, \delta_1 \in \Delta_{\Pi_{HOL}}((X, X') \blacktriangleright t_1 : s_1),$$

$$\ldots, \delta_n \in \Delta_{\Pi_{HOL}}((X, X') \blacktriangleright t_n : s_n)$$

$$\epsilon_{td}\ \lambda Abs((X, X') \blacktriangleright \lambda abs(x_1 : \tau_1, \ldots, x_n : \tau_n).\Phi : [\tau_1, \ldots, \tau_n], [\delta]) =$$

$$\lambda abs\_tr\ (\epsilon_{vs}\ X)\ (\epsilon_{hvl}\ [(x_1, \tau_1), \ldots, (x_n, \tau_n)])$$

$$(bc\_Ne\ Holvar\ encx\ (\epsilon_{hvl}\ [(x_1, \tau_1), \ldots, (x_{n-1}, \tau_{n-1})]))$$

$$(norep\_gc\ Holvar\ Eqbool\_Hvar\ encx1notinx1$$

$$(\ldots(norep\_gc\ Holvar\ Eqbool\_Hvar\ encxn\ notinxn$$

$$(norep\_bc\ Holvar\ Eqbool\_Hvar)) \ldots))$$

$$(\epsilon_{ht}\ (addbvarl\_Hvst\ Holvar\ (\epsilon_{hvl}\ [(x_1, \tau_1), \ldots, (x_n, \tau_n)]))(\epsilon_{vs}\ X)\ \Phi)$$

$$(\epsilon_{td}\ \delta)$$

$$where$$

$$\delta \in \Delta_{\Pi_{HOL}}((X, X') \cup \{x_1 : \tau_1, \ldots, x_n : \tau_n\} \blacktriangleright \phi)$$

$$encx1\ =\ mkpair\ Holvar\ (\epsilon_{vn}\ x_1)\ (\epsilon_\tau\ \tau_1)$$

$encx2 \ = \ mkpair\ Holvar\ (\epsilon_{vn}\ x_2)\ (\epsilon_\tau\ \tau_2)$

$\vdots$

$encxn \ = \ mkpair\ Holvar\ (\epsilon_{vn}\ x_n)\ (\epsilon_\tau\ \tau_n)$

$notinx1 \ = \ consc\_Nin\ Holvar\ Eqbool\_Hvar\ encx1\ encx2\ (\epsilon_{hvl}\ [x_3, \ldots, x_n])$

$(\lambda P : bool \to Prop.\lambda p : P\ false.p)\ (\ldots (basec\_Nin$

$Holvar\ Eqbool\_Hvar\ encxn\ (nil\ Holvar)$

$\vdots$

$notinxn \ = \ basec\_Nin\ Holvar\ Eqbool\_Hvar\ encxn\ (nil\ Holvar)$

$\epsilon_{td}\ \lambda Appl((X, X') \blacktriangleright t(t_1, \ldots, t_n) : \mathbf{Prop}, [\delta, \delta_1, \ldots, \delta_n]) =$

$\lambda appl\_tr\ (\epsilon_{vs}\ X)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ t)$

$(\epsilon_{htl}\ (\epsilon_{vs}\ X)\ [t_1, \ldots, t_n])\ (\epsilon_{\tau l}\ [\tau_1, \ldots, \tau_n])$

$(bc\_Ne\ Holterm\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ t_1)\ (\epsilon_{hvl}\ X)\ [t_2, \ldots, t_n])$

$(\epsilon_{td}\ \delta)\ (\epsilon_{tdl}\ [\delta_1, \ldots, \delta_n])$

$where\ \delta_1 \in \Delta_{\Pi_{HOL}}((X, X') \blacktriangleright t_1 : \tau_1), \ldots, \delta_n \in \Delta_{\Pi_{HOL}}((X, X') \blacktriangleright t_n \tau_n),$

$\delta \in \Delta_{\Pi_{HOL}}((X, X') \blacktriangleright t : [\tau_1, \ldots, \tau_n])$

$\epsilon_{td}\ Forall(X \blacktriangleright \forall x : \tau.\phi : \mathbf{Prop}, [\delta]) =$

$forall\_tr\ (\epsilon_{vs}\ X)\ encx\ (\epsilon_{ht}\ (addbvar\_Hvst\ encx\ (\epsilon_{vs}\ X))\ \phi)\ (\epsilon_{td}\ \delta)$

$where\ \delta \in \Delta_{\Pi_{HOL}}((X, X' \cup x : \tau) \blacktriangleright \phi : \mathbf{Prop}).$

$encx \ = \ mkpair\ Holvar\ (\epsilon_{vn}\ x)\ (\epsilon_\tau\ \tau)$

$\epsilon_{td}\ Implies((X, X') \blacktriangleright \phi \supset \phi' : \mathbf{Prop}, [\delta_1, \delta_2]) =$

$implies\_tr\ (\epsilon_{vs}\ X)\ \ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi')\ (\epsilon_{td}\ \delta_1)\ (\epsilon_{td}\ \delta_2)$

$where\ \delta_1 \in \Delta_{Pi_{HOL}}((X, X') \blacktriangleright \phi : \mathbf{Prop}), \delta_2 \in \Delta_{Pi_{HOL}}((X, X') \blacktriangleright \phi' : \mathbf{Prop}).$

Definition F.92 *The encoding function of list of derivations of well typed terms* ϵ_{tdl} *which given a signature* $\Sigma \in |AlgSig|$ *and a list of closed derivations of well* *typed terms* ($[\Delta_{\Pi_{HOL}}]$) *returns a proof of the proposition*

$$Wfhtermlist\ (\epsilon_{vs}\ X)\ \ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ htl)\ (\epsilon_{\tau l}\ htyl)$$

where $htl\ =\ [ht_1, \ldots, ht_n]$ *is the list of higher-order terms,* $htyl\ =\ [hty_1, \ldots, hty_n]$ *is the list of their associated types and* X *the set of free variables is inductively defined as follows:*

$$\epsilon_{tdl}\ []\ =\ nil\_Whtl\ (\epsilon_{vs}\ X)$$

$$\epsilon_{tdl}\ (cons\ \Delta_{\Pi_{HOL}}\ \delta\ \delta l)\ =$$

$$cons\_Whtl\ (\epsilon_{vs}\ X)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ ht_1)\ (\epsilon_{\tau}\ hty_1)$$

$$(\epsilon_{htl}\ (\epsilon_{vs}\ X)\ [ht_2, \ldots, ht_n])\ (\epsilon_{\tau l}\ [hty_2, \ldots, hty_n])$$

$$(\epsilon_{td}\ \delta)\ (\epsilon_{tdl}\ \delta l)$$

$$where\ \delta \in \Delta_{\Pi_{HOL}}(X \blacktriangleright ht_1 : \tau_1), \delta l \in [\Delta_{\Pi_{HOL}}].$$

Theorem F.93 *There exists a bijection between the closed derivations of a judgement* ($((X, []) \blacktriangleright \phi : \tau)$) *and the normal forms of the proofs of the proposition*

$$Wfhterm\ (\epsilon_{vs}\ (X, []))\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)\ (\epsilon_{\tau}\ \tau)$$

Proof:
To prove the bijection we define a decoding function with type

$$\epsilon_{td}^{-1} : Wfhterm\ (\epsilon_{vs}\ (X, []))\ (\epsilon_{ht}\ (\epsilon_{vs}\ (X, []))\ \phi)\ (\epsilon_{\tau}\ \tau) \to \Delta_{\Pi_{HOL}}((X, [])\ \blacktriangleright\ \phi : \tau)$$

inductively defined as follows:

$$\epsilon_{td}^{-1}\ (ass1\_tr\ vs\ hv\ pr\ prin)\ =$$

$$ASS1((\epsilon_{vs}^{-1}\ vs) \blacktriangleright (\epsilon_{vn}^{-1}\ (fst\ hv) : (\epsilon_{\tau}^{-1}\ (snd\ hv)))$$

$$\epsilon_{td}^{-1}\ (ass2\_tr\ vs\ hv\ pr)\ =$$

$$ASS2((\epsilon_{vs}^{-1}\ vs) \blacktriangleright (\epsilon_{vn}^{-1}\ (fst\ hv) : (\epsilon_{\tau}^{-1}\ (snd\ hv)))$$

$$\epsilon_{td}^{-1}\ (appl\_f\_s\_tr\ vs\ t_1\ \ldots\ t_n\ wft_1\ \ldots\ wft_n)\ =$$

$$APPL((\epsilon_{vs}^{-1}\ vs) \blacktriangleright f((\epsilon_{ht}^{-1}\ vs\ t_1), \ldots, (\epsilon_{ht}^{-1}\ vs\ t_n)) : s,$$

$$[(\epsilon_{td}^{-1}\ wft_1), \ldots, (\epsilon_{td}^{-1}\ wft_n)])$$

$$where\ f : s_1 \times \ldots \times s_n \to s \in \Sigma$$

$$\epsilon_{td}^{-1}\ (appl\_f\_s_1\_\ldots\_s_n\_s\_tr\ vs\ t_1\ \ldots\ t_n\ wft_1\ \ldots\ wft_n) =$$

$$APPL((\epsilon_{vs}^{-1}\ vs) \blacktriangleright f((\epsilon_{ht}^{-1}\ vs\ t_1), \ldots, (\epsilon_{ht}^{-1}\ vs\ t_n) : s,$$

$$[(\epsilon_{td}^{-1}\ wft_1), \ldots, (\epsilon_{td}^{-1}\ wft_n)])$$

$$where\ f : s_1 \times \ldots \times s_n \to s \in \Sigma$$

$$\epsilon_{td}^{-1}\ (\lambda abs\_tr\ vs\ hvl\ nelpr\ ht\ proppr) =$$

$$\lambda ABS((\epsilon_{vs}^{-1}\ vs) \blacktriangleright \lambda(\epsilon_{hvl}^{-1}\ hvl).(\epsilon_{ht}^{-1}\ (addbvarl\_Hvst\ hvl\ vs)\ ht) :$$

$$(type\_list\_from\_hvlhvl)), [(\epsilon_{td}^{-1}\ proppr)]))$$

$$\epsilon_{td}^{-1}(\lambda appl\_tr\ vs\ t\ tl\ htl\ nepr\ wftpr\ wftlpr) =$$

$$\lambda APPL\ (\epsilon_{vs}^{-1}\ vs)\ \blacktriangleright\ (\epsilon_{ht}^{-1}\ vs\ ht)\ (\epsilon_{htl}^{-1}\ vs\ htl) : \mathbf{Prop},$$

$$cons(\epsilon_{td}^{-1}(wftpr))\ (\epsilon_{tdl}^{-1}\ wftlpr)$$

$$\epsilon_{td}^{-1}\ (forall\_tr\ vs\ hv\ ht\ prp) =$$

$$forall(\epsilon_{vs}^{-1}\ vs \blacktriangleright (\epsilon_{ht}^{-1}\ (addbvar\_Hvst\ hv\ vs)$$

$$\forall\ \epsilon_{vn}^{-1}\ (fst\ hv) : \epsilon_{\tau}^{-1}\ (snd\ hv).(\epsilon_{ht}^{-1}\ (addbvar\_Hvst\ hv\ vs)\ ht)\ :\ \mathbf{Prop}, [\epsilon_{td}^{-1}\ prp])$$

$$\epsilon_{td}^{-1}\ (implies\_tr\ vs\ ht\ ht'\ pr\ pr') =$$

$$implies\_tr(\epsilon_{vs}^{-1}\ vs\ \blacktriangleright (\epsilon_{ht}^{-1}\ vs\ ht)\ \supset\ (\epsilon_{ht}^{-1}\ vs\ ht)\ :\ \mathbf{Prop},$$

$$[(\epsilon_{td}^{-1}\ pr), (\epsilon_{td}^{-1}\ pr')])$$

where the function $type\_list\_from\_hvl$ which for any $\Sigma \in |AlgSig|$ has arity

$$type\_list\_from\_hvl : List\ Holvar \to List\ Types_{HOL}(\Sigma)$$

is inductively defined as follows:

$$type\_list\_from\_hvl\ (nil\ Holvar)\ =\ (nil\ Types_{HOL}(\Sigma))$$

$$type\_list\_from\_hvl\ (cons\ Holvar\ hv\ hvl)\ =$$

$$(cons\ Types_{HOL}(\Sigma)\ (\epsilon_{\tau}^{-1}\ (snd\ hv))\ (type\_list\_from\_hvl\ hvl))$$

and the decoding function ϵ_{dtl}^{-1} which for any signature $\Sigma \in |AlgSig|$, for any $vs : Holvar\_set$, $htl : Holterm\_list$, $htyl : Holtype\_list$ has type

$$\epsilon_{dtl}^{-1} : Wfhtermlist\ (\epsilon_{vs}\ X)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)\ (\epsilon_{\tau l}\ htyl) \rightarrow [\Delta_{\Pi_{HOL}}]$$

is inductively defined as follows:

$$\epsilon_{dtl}^{-1}\ (nil\_Whtl\ vs)\ = nil\ \Delta_{\Pi_{HOL}}$$

$$\epsilon_{dtl}^{-1}\ \Delta_{\Pi_{HOL}}\ (cons\_Whtl\ vs\ htr\ htrl\ hty\ htyl\ prt\ prtl)\ =$$

$$cons\ \Delta_{\Pi_{HOL}}\ (\epsilon_{dt}^{-1}\ prt)\ (\epsilon_{dtl}^{-1}\ prtl)$$

The rest of the proof is as explained in Chapter 3.

F.2 Adequate encoding of β-equality

In a similar way as the previous proof system, we present the encoding and decoding functions and the proof of adequacy of the following proof system which defines β-equality

Definition F.94 *The set of rules of* Π_{HOL} *which defines* β*-equality is the following:*

$$\frac{}{x_\tau =_{\beta,X} x_\tau}\ x \in X_\tau \qquad (Vareq)$$

$$\frac{X \blacktriangleright t_1 : s_1\ \ldots\ X \blacktriangleright t_n : s_n \quad t_1 =_{\beta,X} t'_1 \ldots t_n =_{\beta,X} t'_n}{f(t_1,\ldots,t_n) =_{\beta,X} f(t'_1,\ldots,t'_n)}\ f : s_1 \times \ldots \times s_n \rightarrow s_n \in \Sigma \qquad (Termeq)$$

$$\frac{X \blacktriangleright t\ (t_1,\ldots,t_n) : \mathbf{Prop} \quad t =_{\beta,X} t' \quad t_1 =_{\beta,X} t'_1 \ldots t_n =_{\beta,X} t'_n}{t(t_1,\ldots,t_n) =_{\beta,X} t'(t'_1,\ldots,t'_n)}\ (Appleq)$$

$$\frac{X \blacktriangleright t_1 : \tau_1\ \ldots\ X \blacktriangleright t_n : \tau_n \quad X \blacktriangleright \lambda(x_1 : \tau_1,\ldots,x_n : \tau_n).\phi : [\tau_1,\ldots,\tau_n] \quad \phi\ \{t_1/x_1\}\ldots\{t_n/x_n\} =_{\beta,X} \phi'}{\lambda(x_1 : \tau_1,\ldots,x_n : \tau_n).\phi\ (t_1,\ldots,t_n) =_{\beta,X} \phi'}\ (Llambdaeq)$$

$$\frac{\begin{array}{c} X \blacktriangleright \lambda(x_1 : \tau_1, \ldots, x_n : \tau_n).\phi : [\tau_1, \ldots, \tau_n] \\ \phi \ =_{\beta, X \cup \{x_1:\tau_1,\ldots,x_n:\tau_n\}} \ \phi' \ \{x_1/x_1'\} \ldots \{x_n/x_n'\} \end{array}}{\lambda(x_1 : \tau_1, \ldots, x_n : \tau_n).\phi \ =_{\beta,X} \ \lambda(x_1' : \tau_1, \ldots, x_n' : \tau_n).\phi'} \ (Lambdaeq)$$

$$\frac{\begin{array}{c} X \cup \{x : \tau\} \blacktriangleright \phi : \mathbf{Prop} \\ \phi \ =_{\beta, X \cup \{x:\tau\}} \ \phi'\{x/x'\} \end{array}}{\forall x : \tau.\phi \ =_{\beta,X} \ \forall x' : \tau.\phi'} \ (Foralleq)$$

$$\frac{\begin{array}{c} X \blacktriangleright \phi' : \mathbf{Prop} \\ X \blacktriangleright \phi : \mathbf{Prop} \\ \phi' \ =_{\beta,X} \ \phi \end{array}}{\phi \ =_{\beta,X} \ \phi'} \ (Sym)$$

Definition F.95 *The inductive relation*

$$Same\_length\_and\_type : .\Pi l : List\ Holvar.\Pi l' : List\ Holvar.Prop$$

is defined by the following set of constructors:

$nil\_slt : Same\_length\ (nil\ Holvar)\ (nil\ Holvar)$

$cons\_slt : \Pi t : Holtype.\Pi vn, vn' : Var\_name.\Pi tl, tl' : List\ Holvar.$

$\Pi slpr : Same\_length\_and\_type\ tl\ tl'.$

$Same\_length\ (cons\ Holvar\ (vn, t)\ tl)\ (cons\ Holvar\ (vn, t)\ tl')$

Definition F.96 *The inductive relation*

$$Beta\_eq : \Pi ht : Holterm.\Pi hvs : Holvar\_set.\Pi ht : Holterm.Prop$$

is defined by the following set of constructors:

$\{vareq : \Pi hv : Holvar.\Pi vs : Holvar\_set.\Pi prin : Isin\_freev\_Hvs\ hv\ (snd\ vs).$

$Beta\_eq\ hv\ vs\ hv\}\ \} \cup$

$\{termeq\_f : \Pi vs : Holvar\_set.\Pi t_1, \ldots, t_n, t_1', \ldots, t_n' : Holterm.$

$\Pi wfhtrm_1 : Wfhterm\ vs\ t_1\ \tau_1. \ldots .\Pi wfhtrm_n : Wfhterm\ vs\ t_n\ \tau_n.$

$\Pi beqpr_1 : Beta\_eq\ t_1\ vs\ t_1'. \ldots .\Pi beqpr_n : Beta\_eq\ t_n\ vs\ t_n'.$

$Beta\_eq\ f(t_1, \ldots, t_n)\ vs\ f(t_1', \ldots, t_n')\ |$

$f : s_1 \times \ldots \times s_n \to s \in \Sigma$ *and* f *is not overloaded in* $\Sigma\} \cup$

$\{termeq\_f\_s_1\_\ldots\_s_n\_s : \Pi vs : Holvar\_set.\Pi t_1, \ldots, t_n, t'_1, \ldots, t'_n : Holterm.$

$\Pi wfhtrm_1 : Wfhterm\ vs\ t_1\ \tau_1. \ldots .\Pi wfhtrm_n : Wfhterm\ vs\ t_n\ \tau_n.$

$\Pi beqpr_1 : Beta\_eq\ t_1\ vs\ t'_1. \ldots .\Pi beqpr_n : Beta\_eq\ t_n\ vs\ t'_n.$

$Beta\_eq\ f(t_1, \ldots, t_n)\ vs\ f(t'_1, \ldots, t'_n) \mid$

$f : s_1 \times \ldots \times s_n \to s \in \Sigma\ and\ f\ is\ overloaded\ in\ \Sigma\} \cup$

$\{appleq : \Pi vs : Holvar\_set.\Pi t, t' : Holterm.\Pi tl, tl' : Holterm\_list.$

$\Pi slpr : Same\_length\ tl\ ; tl'.$

$\Pi wfhtrm : Wfhterm\ vs\ (appl\_Htrm\ t\ tl)\ prop\_Holt.$

$\Pi beqpr : Beta\_eq\ t\ vs\ t'.\Pi beqpr_1 : Beta\_eql\ tl\ vs\ tl'.$

$Beta\_eq\ (appl\_Htrm\ t\ tl)\ vs\ (appl\_Htrm\ t'\ tl')$

$llambdaeq : \Pi vs : Holvar\_set.\Pi ht, ht' : Holterm.$

$\Pi htl : Holterm\_list.\Pi hvl : List\ Holvar.$

$\Pi slpr : Same\_length\ \ hvl\ htl$

$\Pi wfhtrmlpr : Wfhtermlist\ vs\ htl\ (get\_holtypel\ hvl).$

$\Pi wft : Wfhterm\ vs\ (getvarl\_Hvst\ hvl\ (addfvarl\_Hvst\ hvl\ vs))\ ht)$

$(getholtypel\_Hvl\ hvl).$

$\Pi beqpr : Beta\_eq\ ht\ vs\ (substhvl\_Htrm$

$(getfindex\_Hvst\ (addfvarl\_Hvst\ hvl\ vs))ht\ htl\ hvl).$

$Beta\_eq\ (appl\_Htrm\ (abstr\_Htrm\ hvl\ ht)\ htl)\ vs\ ht'$

$lambdaeq : \Pi vs : Holvar\_set.\Pi hvl, hvl' : List\ Holvar.$

$\Pi slpr : Same\_length\_and\_type\ hvl\ hvl'.\Pi ht, ht' : Holterm.$

$\Pi wft : Wfhterm\ vs\ (getvarl\_Hvst\ hvl\ (addfvarl\_Hvst\ hvl\ vs))\ ht)\ (getholtypel\_Hvl\ hvl).$

$\Pi beqpr : Beta\_eq\ ht\ (addfvarl\_Hvst\ hvl\ vs)\ (substhvl\_Htrm$

$(getfindex\_Hvst\ (addfvarl\_Hvst\ hvl'\ vs))\ ht'\ hvl\ hvl').$

$Beta\_eq\ (abstr\_Htrm\ (getvarl\_Hvst\ hvl\ (addfvarl\_Hvst\ hvl\ vs))\ ht)\ vs$

$(abstr\_Htrm\ (getvarl\_Hvst\ hvl'\ (addfvarl\_Hvst\ hvl'\ vs))\ ht')$

$foralleq : \Pi vs : Holvar\_set.\Pi hv, hv' : Holvar.\Pi ht, ht' : Holterm.$

$\Pi wft : Wfhterm\ (addfvar\_Hvst\ hv\ vs)\ ht\ prop\_Holt.$

$\Pi beqpr : Beta\_eq\ ht\ (addfvar\_Hvst\ hv\ vs)\ (subst\_Htrm$

$(getfindex\_Hvst\ (addfvar\_Hvst\ hv'\ vs))\ ht'\ hv\ hv').$

$Beta\_eq\ (forall\_Htrm\ hv\ ht)\ vs\ (forall\_Htrm\ hv'\ ht')$

$sym : \Pi vs : Holvar\_set.\Pi ht, ht' : Holterm.$

$\Pi wft : Wfhterm\ vs\ ht\ prop\_Holt.\Pi wft : Wfhterm\ vs\ ht'\ prop\_Holt.$

$\Pi beqpr : Beta\_eq\ ht\ vs\ ht'.Beta\_eq\ ht'\ vs\ ht\}$

Definition F.97 *The inductive relation*

$Beta\_eql : \Pi htl : Holterm\_list.\Pi hvs : Holvar\_set.\Pi htl : Holterm\_list.Prop$

is defined by the following set of constructors:

$nil\_Beql : \Pi vs : Holvar\_set.Beta\_eql\ (nil\_Htrm)\ vs\ (nil\_Htrm)$

$cons\_Beql : \Pi t, t' : Holterm.\Pi tl, tl' : Holterm\_list.\Pi vs : Holvar\_set.$

$\Pi beqpr : Beta\_eq\ t\ vs\ t'.\Pi beqprl : Beta\_eql\ tl\ vs\ tl'.$

$Beta\_eql\ (cons\_Htrm\ t\ tl)\ vs\ (cons\_Htrm\ t'\ tl')$

Definition F.98 *The encoding function*

$$\epsilon_\beta : \Delta_{\Pi_{HOL}}(\phi \;=_{\beta,X}\; \phi') \to Beta\_eq\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)\ (\epsilon_{vs}\ X)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi')$$

for any sequence of variables X *and for any* $\phi, \phi' \in Sen_{HOL}(X, \Sigma, \mathbf{Prop})$ *is inductively defined as follows:*

$$\epsilon_\beta(Vareq(x_\tau =_{\beta,X} x_\tau)) \;=\; vareq\ encx\ (\epsilon_{vs}\ X)$$

$$(ctr\_Infhvs\ encx\ (\epsilon_{vs}\ (X, X'))$$

$$(genc\_Inhivl\ encx\ (getvar\_Hvst\ enchv_n$$

$$(\epsilon_{vs}\ [hv_1, \ldots, hv_{i-1}, (x,\tau), hv_i, \ldots, hv_n]))$$

$$(fst\ (\epsilon_{vs}\ [hv_1, \ldots, hv_{i-1}, (x,\tau), hv_i, \ldots, hv_{n-1}]))$$

$$(\ldots(genc\_Inhivl\ encx\ (getvar\_Hvst\ enchv_i$$

$$(\epsilon_{vs}\ [hv_1, \ldots, hv_{i-1}, (x,\tau), hv_i]))$$

$$(fst\ (\epsilon_{vs}\ [hv_1, \ldots, hv_{i-1}, (x,\tau)]))$$

$$(base\_Hivs\ encx\ (getvar\_Hvst\ encx$$

$$(\epsilon_{vs}\ [hv_1, \ldots, hv_{i-1}, (x,\tau)]))$$

$$(fst\ (\epsilon_{vs}\ [hv_1, \ldots, hv_{i-1}, (x,\tau)]))$$

$$(\lambda P : bool \to Prop.\lambda pr : P\ true.pr)))\ldots)))$$

where $X = [hv_1, \ldots, hv_{i-1}, (x,\tau), hv_i, \ldots, hv_n]$

$$encx \;=\; mkpair\ Holvar\ (\epsilon_{vn}\ x)\ (\epsilon_\tau\ \tau)$$

$$enchv_n \;=\; mkpair\ Holvar\ (fst\ hv_n)\ (snd\ hv_n)$$

$$enchv_i \;=\; mkpair\ Holvar\ (fst\ hv_i)\ (snd\ hv_i)$$

$$\epsilon_\beta \ (Termeq(f(t_1,\ldots,t_n) =_{\beta,X} f(t'_1,\ldots,t'_n),[\delta_1,\ldots,\delta_n,\delta'_1,\ldots,\delta'_n])) \ =$$

$$termeq\_f \ (\epsilon_{vs} \ X) \ (\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t_1) \ \ldots \ (\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t_n)$$

$$(\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t'_1) \ \ldots \ (\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t'_n)$$

$$(\epsilon_{td} \ \delta_1) \ \ldots \ (\epsilon_{td} \ \delta_n) \ (\epsilon_\beta \ \delta'_1) \ldots (\epsilon_\beta \ \delta'_n),$$

$$if \ \ f : s_1 \times \ldots \times s_n \to s \ \in \ \Sigma \ \ and \ \ f \ \ is \ \ not \ \ overloaded \ \ in \ \ \Sigma$$

$$termeq\_f\_s_1\_\ldots\_s_n\_s \ (\epsilon_{vs} \ X) \ (\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t_1) \ \ldots \ (\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t_n)$$

$$(\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t'_1) \ \ldots \ (\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t'_n)$$

$$(\epsilon_{td} \ \delta_1) \ \ldots \ (\epsilon_{td} \ \delta_n) \ (\epsilon_\beta \ \delta'_1) \ldots (\epsilon_\beta \ \delta'_n),$$

$$if \ \ f : s_1 \times \ldots \times s_n \to s \ \in \ \Sigma \ \ and \ \ f \ \ is \ \ not \ \ overloaded \ \ in \ \ \Sigma$$

$where$

$$\delta_1 \in \Delta_{\Pi_{HOL}}(X \blacktriangleright t_1 : s_1),\ldots,\delta_n \in \Delta_{\Pi_{HOL}}(X \blacktriangleright t_n : s_n),$$

$$\delta'_1 \in \Delta_{\Pi_{HOL}}(t_1 =_{\beta,X} t'_1),\ldots,\delta'_n \in \Delta_{\Pi_{HOL}}(t_n =_{\beta,X} t'_n)$$

$$\epsilon_\beta(Appleq(t(t_1,\ldots,t_n) \ =_{\beta,X} \ t'(t'_1,\ldots,t'_n),[\delta',\delta'',\delta_1,\ldots,\delta_n])) \ =$$

$$appleq \ (\epsilon_{vs} \ X) \ (\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t) \ (\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t') \ (\epsilon_{htl} \ (\epsilon_{vs} \ X) \ [t_1,\ldots,t_n])$$

$$(\epsilon_{htl} \ (\epsilon_{vs} \ X) \ [t'_1,\ldots,t'_n])$$

$$(cons\_sl \ (\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t_1) \ (\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t'_1) \ (\epsilon_{htl} \ (\epsilon_{vs} \ X) \ [t_2,\ldots,t_n])$$

$$(\epsilon_{htl} \ (\epsilon_{vs} \ X) \ [t'_2,\ldots,t'_n]) \ (\ldots(cons\_sl \ (\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t_n) \ (\epsilon_{ht} \ (\epsilon_{vs} \ X) \ t'_n)$$

$$(nil\_Htrm) \ (nil\_Htrm) \ (nil\_sl))\ldots))$$

$$(\epsilon_{dt} \ \delta') \ (\epsilon_\beta \ \delta'') \ (\epsilon_{\beta l} \ [\delta_1,\ldots,\delta_n])$$

$where$ $\delta' \in \Delta_{\Pi_{HOL}}(X \blacktriangleright t \ (t_1,\ldots,t_n) : \mathbf{Prop}),$

$$\delta'' \in \Delta_{\Pi_{HOL}}(t \ =_{\beta,X} \ t'),$$

$$\delta_1 \in \Delta_{\Pi_{HOL}}(t_1 \ =_{\beta,X} \ t'_1),\ldots,\delta_n \in \Delta_{\Pi_{HOL}}(t_n =_{\beta,X} t'_n)$$

$$\epsilon_\beta\ (Llambdaeq(\lambda(x_1:\tau_1,\ldots,x_n:\tau_n).\phi\ (t_1,\ldots,t_n)\ =_{\beta,X}\ \phi',[\delta',\delta'',\delta_1,\ldots,\delta_n])) =$$

$$llambdaeq\ (\epsilon_{vs}\ X)$$

$$(\epsilon_{ht}\ (addfvarl\_Hvst\ (\epsilon_{hvl}\ [(x_1,\tau_1),\ldots,(x_n,\tau_n)])\ (\epsilon_{vs}\ X))\ \phi)$$

$$(\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi')\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ [t_1,\ldots,t_n])$$

$$(\epsilon_{hvl}\ [(x_1,\tau_1),\ldots,(x_n,\tau_n)])$$

$$(cons\_sl\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ t_1)\ (mkpair\ Holvar\ (\epsilon_{vn}\ x_1)$$

$$(\epsilon_\tau\ \tau_1))\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ [t_2,\ldots,t_n])$$

$$(\epsilon_{hvl}\ [(x_2,\tau_2),\ldots,(x_n,\tau_n)])\ (\ldots(cons\_sl\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)$$

$$(mkpair\ Holvar\ (\epsilon_{vn}\ x_n,\ )(\epsilon_\tau\ \tau_n))(nil\_Htrm)$$

$$(nil\ Holvar)\ (nil\_sl))\ldots))$$

$$(cons\ (\epsilon_\beta\ \delta')\ (cons\ (\epsilon_{td}\ \delta'')\ (\epsilon_{tdl}\ [\delta_1,\ldots,\delta_n])))$$

$$where\ \delta'\in\Delta_{\Pi_{HOL}}(\phi'\ =_{\beta,X}\ \phi\ \{t_1/x_1\}\ldots\{t_n/x_n\}),$$

$$\delta''\in\Delta_{\Pi_{HOL}}(X\blacktriangleright\lambda(x_1:\tau_1,\ldots,x_n:\tau_n).\phi:[\tau_1,\ldots,\tau_n]),$$

$$\delta_1\in\Delta_{\Pi_{HOL}}(X\blacktriangleright t_1:\tau_1),\ldots,\delta_n\in\Delta_{\Pi_{HOL}}(X\blacktriangleright t_n:\tau_n)$$

$$\epsilon_\beta(Lambdaeq(\lambda(x_1:\tau_1,\ldots,x_n:\tau_n).\phi\ =_{\beta,X}\ \lambda(x_1':\tau_1,\ldots,x_n':\tau_n).\phi'),[\delta_1,\delta_2])$$

$$lambdaeq\ (\epsilon_{vs}\ X)$$

$$(\epsilon_{hvl}\ [(x_1,\tau_1),\ldots,(x_n,\tau_n)])\ (\epsilon_{hvl}\ [(x_1',\tau_1),\ldots,(x_n',\tau_n)])$$

$(cons\_slt\ \ (\epsilon_\tau\ \tau_1)\ (\epsilon_{vn}\ x_1)\ \ (\epsilon_{vn}\ x'_1)$

$(\epsilon_{hvl}\ [(x_2, \tau_2), \ldots, (x_n, \tau_n)])\ (\epsilon_{hvl}\ [(x'_2, \tau'_2), \ldots, (x'_n, \tau'_n)])$

$(\ldots(cons\_slt\ (\epsilon_\tau\ \tau_n)\ (\epsilon_{vn}\ x_n)\ (\epsilon_{vn}\ x'_n)$

$(nil\ Holvar)\ (nil\ Holvar)\ (nil\_sl\ Holvar\ Holvar))\ldots)))$

$(\epsilon_{ht}\ (addfvarl\_Hvst\ (\epsilon_{hvl}\ [(x_1, \tau_1), \ldots, (x_n, \tau_n)])\ (\epsilon_{vs}\ X))\ \phi)$

$(\epsilon_{ht}\ (addfvarl\_Hvst\ (\epsilon_{hvl}\ [(x'_1, \tau_1), \ldots, (x'_n, \tau_n)])\ (\epsilon_{vs}\ X))\ \phi')$

$(\epsilon_\beta\ \delta_1)\ (\epsilon_{td}\ \delta_2)$

$where\ \delta_1 \in \Delta_{\Pi_{HOL}}(\phi\ =_{\beta, X\cup\{x_1:\tau_1,\ldots,x_n:\tau_n\}}\ \phi'\ \{x_1/x'_1\}\ldots\{x_n/x'_n\}),$

$\delta_2 \in \Delta_{\Pi_{HOL}}(X \blacktriangleright \lambda(x_1 : \tau_1, \ldots, x_n : \tau_n).\phi : [\tau_1, \ldots, \tau_n])$

$\epsilon_\beta(Foralleq(\forall x : \tau.\phi\ =_{\beta,X}\ \forall x' : \tau.\phi'), [\delta_1, \delta_2])\ =$

$foralleq\ (\epsilon_{vs}\ X)\ \ encx\ encx'$

$(\epsilon_{ht}\ (addfvar\_Hvst\ encx\ (\epsilon_{vs}\ X))\ \phi)$

$(\epsilon_{ht}\ (addfvar\_Hvst\ encx'\ (\epsilon_{vs}\ X))\ \phi')\ (\epsilon_\beta\ \delta_1)\ (\epsilon_{td}\ \delta_2)$

$where\ \ encx\ =\ mkpair\ Holvar\ (\epsilon_{vn}\ x)\ (\epsilon_\tau\ \tau)$

$encx'\ =\ mkpair\ Holvar\ (\epsilon_{vn}\ x')\ (\epsilon_\tau\ \tau)$

$\delta_1 \in \Delta_{\Pi_{HOL}}(\phi\ =_{\beta,X\cup\{x:\tau\}}\ \phi'\{x/x'\}),$

$\delta_2 \in \Delta_{\Pi_{HOL}}(X \cup x : \tau \blacktriangleright \phi : \mathbf{Prop})$

$\epsilon_\beta\ (Sym(\phi\ =_{\beta,X}\ \ \phi', [\delta, \delta', \delta'']))\ =$

$sym\ (\epsilon_{vs}\ X)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi')$

$(\epsilon_{dt}\ \delta)\ (\epsilon_{dt}\ \delta')(\epsilon_\beta\ \delta'')$

Definition F.99 *The encoding function*

$$\epsilon_{\beta l} : [\Delta_{\Pi_{HOL}}] \rightarrow Beta\_eql\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ htl)\ (\epsilon_{vs}\ X)\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ htl')$$

where $htl = [ht_1, \ldots, ht_n], htl' = [ht'_1, \ldots, ht'_n]$ *are of type* $(Holterm\_list)$ *is inductively defined as follows:*

$\epsilon_{\beta l}\ [] = (nil\_Beql\ (\epsilon_{vs}\ X))$

$\epsilon_{\beta l}\ (cons\ beqd\ beqdl) = (cons\_Beql\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ ht_1)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ ht'_1)\ (\epsilon_{vs}\ X)$

$(\epsilon_{htl}\ (\epsilon_{vs}\ X)\ [ht_2, \ldots, ht_n])\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ [ht'_2, \ldots, ht'_n])\ (\epsilon_{\beta}\ beqd)\ (\epsilon_{\beta l}\ beqdl))$
where $beqd \in \Delta_{\Pi_{HOL}}(ht_1\ =_{\beta,X}\ ht'_1)$*, and* $beqdl$ *is a list of derivations*

of $\Delta_{\Pi_{HOL}}$ *of the judgements* $ht_1\ =_{\beta,X}\ ht'_1, \ldots, ht_n\ =_{\beta,X}\ ht'_n$.

Proposition F.100 *For any signature* Σ*, for any type* $\tau : Holtype$*, for any variable set* $vs : Holvar\_set$*, for any higher-order terms* $ht, ht' : Holterm$*, if* $Wfhterm\ vs\ ht\ \tau$ *is inhabited and if* $Beta\_eq\ ht\ vs\ ht'$ *is inhabited then* $Wfhterm\ vs\ ht'\ \tau'$ *is inhabited.*

Proof:
By induction on the derivations of $Beta\_eq\ ht\ vs\ ht'$.

Proposition F.101 *For any signature* $\Sigma \in |AlgSig|$*, for any type* $\tau \in Types_{HOL}(\Sigma)$*, for any terms* $ht, ht' \in Sen_{HOL}(\Sigma, X)$*, if* $X \blacktriangleright ht : \tau$ *and* $ht\ =_{\beta,X}\ ht'$ *then* $X \blacktriangleright ht' : \tau$

Proof:
By induction on the derivations of $ht\ =_{\beta,X}\ ht$.

Theorem F.102 *For any signature* Σ*, for any sequence of variables* X*, for any sentences* ϕ, ϕ' *such that* $X \blacktriangleright \phi :$ **Prop** *and* $X \blacktriangleright \phi' :$ **Prop** *, there exists a bijection between closed derivations of the judgement* $\phi =_{\beta,X} \phi$ *and the inhabitants of the inductive relation* $Beta\_eq\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)\ (\epsilon_{vs}\ X)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)$

Proof:

The decoding function is inductively defined as follows:

$$\epsilon_{\beta}^{-1}\ (vareq\ hv\ vs\ prin)\ =\ Vareq(\epsilon_{hv}^{-1}(hv)\ =_{\beta,\epsilon_{vs}^{-1}(vs)}\ \epsilon_{hv}^{-1}(hv))$$

$$\epsilon_{\beta}^{-1}\ (termeq\_f\ vs\ t_1\ \ldots\ t_n\ t'_1\ \ldots\ t'_n\ wfhtrm_1 \ldots wfhtrm_n$$
$$beqpr_1\ \ldots\ beqpr_n)\ =$$
$$Termeq\_f(f(\epsilon_{ht}^{-1}\ vs\ t_1, \ldots, \epsilon_{ht}^{-1}\ vs\ t_n)\ =_{\beta,\epsilon_{vs}^{-1}\ vs}$$
$$f(\epsilon_{ht}^{-1}\ vs\ t'_1, \ldots, \epsilon_{ht}^{-1}\ vs\ t'_n), [(\epsilon_{td}^{-1}\ wfhtrm_1), \ldots, (\epsilon_{td}^{-1}\ wfhtrm_n),$$
$$\epsilon_{\beta}^{-1}\ beqpr_1, \ldots, \epsilon_{\beta}^{-1}\ beqpr_n]$$

$$\epsilon_{\beta}^{-1}\ (termeq\_f\_s_1\_\ldots\_s_n\_s\ vs\ t_1\ \ldots\ t_n\ t'_1\ \ldots\ t'_n\ \tau_1 \ldots \tau_n\ wfhtrm_1 \ldots wfhtrm_n$$
$$beqpr_1\ \ldots\ beqpr_n)\ =$$
$$Termeq\_f\_s_1\_\ldots\_s_n\_s(f(\epsilon_{ht}^{-1}\ vs\ t_1, \ldots, \epsilon_{ht}^{-1}\ vs\ t_n)\ =_{\beta,\epsilon_{vs}^{-1}(vs)}$$
$$f(\epsilon_{ht}^{-1}\ vs\ t'_1, \ldots, \epsilon_{ht}^{-1}\ vs\ t'_n),$$
$$[\epsilon_{td}^{-1}\ wfhtrm_1, \ldots, \epsilon_{td}^{-1}\ wfhtrm_n, \epsilon_{\beta}^{-1}\ beqpr_1, \ldots, \epsilon_{\beta}^{-1}\ beqpr_n]$$

$$\epsilon_{\beta}^{-1}\ (appleq\ vs\ t\ t'\ tl\ tl'\ slpr\ wfhtrm\ beqpr\ beqprl)\ =$$
$$Appleq(\ (\epsilon_{ht}^{-1}\ vs\ t)\ (\epsilon_{htl}^{-1}\ vs\ tl)\ =_{\beta,\epsilon_{vs}^{-1}\ vs}$$
$$(\epsilon_{ht}^{-1}\ vs\ t')\ (\epsilon_{htl}^{-1}\ vs\ tl')\ , (cons\ (\epsilon_{td}^{-1}\ wfhtrm)\ (cons\ (\epsilon_{\beta}^{-1}\ beqpr)\ (\epsilon_{\beta l}^{-1}\ beqprl))))$$

$$\epsilon_{\beta}^{-1}\ (llambdaeq\ vs\ ht\ ht'\ htl\ hvl\ slpr\ wftrml\ wft\ beqpr)\ =$$
$$Llambdaeq(\ \lambda(\epsilon_{hvl}^{-1}\ hvl).$$
$$(\epsilon_{ht}^{-1}\ (addfvarl\_Hvst\ hvl\ vs)\ ht)\ (\epsilon_{htl}^{-1}\ vs\ htl) =_{\beta,\epsilon_{vs}^{-1}\ vs} (\epsilon_{ht}^{-1}\ vs\ ht'),$$
$$(cons\ (\epsilon_{td}^{-1}\ wft)\ (cons\ (\epsilon_{\beta}^{-1}\ beqpr)\ (\epsilon_{tdl}^{-1}\ wftrml))))$$

ϵ_β^{-1} $(lambdaeq\ vs\ hvl\ hvl'\ slpr\ ht\ ht'\ wft\ beqpr)\ =$

$$Lambdaeq\ (\lambda(\epsilon_{hvl}^{-1}\ hvl).(\epsilon_{ht}^{-1}\ (addfvarl\_Hvst\ hvl\ vs)\ ht)\ =_{\beta,\epsilon_{vs}^{-1}\ vs}$$

$$\lambda(\epsilon_{hvl}^{-1}\ hvl').(\epsilon_{ht}^{-1}\ (addfvarl\_Hvst\ hvl'\ vs)\ ht')\ ,[\epsilon_{td}^{-1}\ wft, \epsilon_\beta^{-1}\ beqpr])$$

ϵ_β^{-1} $(foralleq\ vs\ hv\ hv'\ ht\ ht'\ wft\ beqpr)\ =$

$$Foralleq(\forall(\epsilon_{vn}^{-1}\ (fst\ hv)) : (\epsilon_\tau^{-1}\ (snd\ hv)).(\epsilon_{ht}^{-1}\ (addfvar\_Hvst\ hv\ vs)\ ht)\ =_{\beta,\epsilon_{vs}^{-1}\ vs}$$

$$\forall(\epsilon_{vn}^{-1}\ (fst\ hv')) : (\epsilon_\tau^{-1}\ (snd\ hv')).(\epsilon_{ht}^{-1}\ (addfvar\_Hvst\ hv'\ vs)\ ht'),$$

$$[\epsilon_{td}^{-1}\ wft, \epsilon_\beta^{-1}\ beqpr])$$

ϵ_β^{-1} $(sym\ vs\ ht\ ht'\ wfht\ wfht'\ betaeq)\ =$

$$(Sym(\epsilon_{ht}^{-1}\ vs\ ht\ =_{\beta,\epsilon_{vs}^{-1}\ vs}\ \epsilon_{ht}^{-1}\ vs\ ht'), [\epsilon_{td}^{-1}\ wfht, \epsilon_{td}^{-1}\ wfht', \epsilon_\beta^{-1}\ betaeq])$$

And the decoding function $\epsilon_{\beta l}^{-1}$ with arity

$$\epsilon_{\beta l}^{-1} : Beta\_eql\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ htl)\ (\epsilon_{vs}\ X)\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ htl') \to [\Delta_{\Pi_{HOL}}]$$

where $htl\ =\ [ht_1, \ldots, ht_n], htl'\ =\ [ht'_1, \ldots, ht'_n]$ is of type $(Holterm\_list)$ is inductively defined as follows:

$$\epsilon_{\beta l}^{-1}\ (nil\_Beql\ vs)\ =\ []$$

$$\epsilon_{\beta l}^{-1}\ (cons\_Beql\ ht\ ht'\ htl\ htl'\ vs\ beqpr\ beqlpr)\ =\ cons\ (\epsilon_\beta^{-1}\ beqpr)\ (\epsilon_{\beta l}^{-1}\ beqlpr)$$

The rest of the proof is as explained in Chapter 3.

F.3 Adequate encondings of the natural deduction system

And finally, in this subsection we present the encoding and decoding functions and the proof of adequacy of the proof system for the deduction of terms of higher-order logic.

Definition F.103 *The natural deduction system* $\Pi_{HOL}(\Gamma, \Sigma)$ *is defined by the* *following set of rules for any* $\Gamma \in \mathcal{P}(Sen_{HOL}(\Sigma, X, \mathbf{Prop}))$*:*

$$\frac{\Gamma \blacktriangleright_X \phi : \mathbf{Prop} \quad \Gamma \cup \{\phi\} \Rightarrow_X \phi'}{\Gamma \Rightarrow_X \phi \supset \phi'} \quad (impl\_i)$$

$$\frac{\Gamma \Rightarrow_X \phi \supset \phi' \quad \Gamma \Rightarrow_X \phi}{\Gamma \Rightarrow_X \phi'} \quad (impl\_e)$$

$$\frac{\Gamma \Rightarrow_{X \cup \{x:\tau\}} \phi}{\Gamma \Rightarrow_X \forall x : \tau.\phi} \quad (forall\_i)$$

$$\frac{\Gamma \Rightarrow_X \forall x : \tau.\phi \quad X \cup \{x : \tau\} \blacktriangleright \phi : Prop \quad X \blacktriangleright t : \tau}{\Gamma \Rightarrow_X \phi\{t/x\}} \quad (forall\_e)$$

$$\frac{\Gamma \Rightarrow_X \phi \quad \psi =_\beta \phi}{\Gamma \Rightarrow_X \psi} \quad (CONV)$$

Definition F.104 *The inductive relation*

$$HOL : Holterm\_list \to Holvar\_set \to Holterm \to Prop$$

is defined by the following set of constructors:

$$impl\_i : \Pi env : List\ Holterm.\Pi vs : Holvar\_set.\Pi ht, ht' : Holterm.$$

$$\Pi wfenv : Wfhtermlist\ vs\ env\ (cons\ prop\_Holt\ (\ldots nil)\ldots).$$

$$\Pi prt : Wfhterm\ vs\ ht\ prop\_Holt.$$

$$\Pi prd : HOL\ (cons\_Htrm\ ht\ env)\ vs\ ht'.$$

$$HOL\ env\ vs\ (implies\_Htrm\ ht\ ht')$$

$$impl\_e : \Pi env : Holterm\_list.\Pi vs : Holvar\_set.\Pi ht, ht' : Holterm.$$

$$\Pi wfenv : Wfhtermlist\ vs\ env\ (cons\ prop\_Holt\ (\ldots nil)\ldots).$$

$$\Pi prd : HOL\ env\ vs\ (implies\_Htrm\ ht\ ht').\Pi prd' : HOL\ env\ vs\ ht.$$

$$HOL\ env\ vs\ ht'$$

$forall\_i : \Pi env : Holterm\_list.\Pi vs : Holvar\_set.\Pi hv : Holvar.\Pi ht : Holterm.$

$\Pi wfenv : Wfhtermlist\ vs\ env\ (cons\ prop\_Holt\ (\ldots nil)\ldots).$

$\Pi dpr : HOL\ env\ (addfvar\_Hvst\ hv\ vs)\ ht.$

$HOL\ env\ vs\ (forall\_Htrm\ (getvar\_Hvst\ hv\ (addfvar\_Hvst\ hv\ vs))\ ht)$

$forall\_e : \Pi env : Holterm\_list.\Pi vs : Holvar\_set.\Pi hv : Holvar.\Pi ht, ht' : Holterm.$

$\Pi wfp : Wfhtermlist\ vs\ env\ (cons\ prop\_Holt\ (\ldots nil)\ldots).$

$\Pi wft : Wfhterm\ (addfvar\_Hvst\ hv\ vs)\ ht\ prop\_Holt.$

$\Pi wft' : Wfhterm\ vs\ ht'\ (snd\ hv).$

$\Pi dpr : HOL\ env\ vs\ (forall\ (getvar\_Hvst\ hv\ (addfvar\_Hvst\ hv\ vs))\ ht).$

$HOL\ env\ vs\ (subst\_Htrm\ (getfindex\_Hvst\ (addfvar\_Hvst\ hv\ vs))$

$ht\ ht'\ (getvar\_Hvst\ hv\ (addfvar\_Hvst\ hv\ vs)))$

$conv : \Pi env : Holterm\_list.\Pi vs : Holvar\_set.\Pi ht, ht' : Holterm.$

$\Pi wfp : Wfhtermlist\ vs\ env(cons\ prop\_Holt\ (\ldots nil)\ldots).$

$\Pi prd : HOL\ env\ vs\ ht.$

$\Pi beqpr : Beta\_eq\_Htrm\ ht\ vs\ ht'.HOL\ env\ vs\ ht'\}$

Definition F.105 *The encoding function of derivations of HOL ϵ_{hd} which given a closed derivation in $\Delta_{\Pi_{HOL}}(\Gamma \Rightarrow_X \phi)$ returns a proof of the proposition*

$$HOL\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ \Gamma)\ \ (\epsilon_{vs}\ X)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)$$

is inductively defined by closed derivations as follows:

For all the cases we will assume that $\Gamma = [\phi_1, \ldots, \phi_n]$, $\delta_1 \in \Delta_{\Pi_{HOL}}(X \blacktriangleright \phi_1 : \mathbf{Prop}), \ldots, \delta_n \in \Delta_{\Pi_{HOL}}(X \blacktriangleright \phi_n : \mathbf{Prop})$ *and* $wfenvpr = \epsilon_{tdl}\ [\delta_1, \ldots, \delta_n]$

$\epsilon_{hd}\ (impl\_i\ (\Gamma \Rightarrow_X \phi \supset \phi', [\delta, \delta']))\ =$

$impl\_i\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ \Gamma)\ (\epsilon_{vs}\ X)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi')\ wfenvpr\ (\epsilon_{td}\ \delta')\ (\epsilon_{hd}\ \delta)$

$where\ \delta \in \Delta_{\Pi_{HOL}}(\Gamma \cup \phi \Rightarrow_X \phi'), \delta' \in \Delta_{\Pi_{HOL}}(X \blacktriangleright \phi' : \mathbf{Prop}).$

$\epsilon_{hd}\ (impl\_e\ (\Gamma \Rightarrow_X \phi', [\delta_1, \delta_2]))\ =$

$(\epsilon_{htl}\ (\epsilon_{vs}\ X)\ \Gamma)\ (\epsilon_{vs}\ X)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi')\ wfenvpr\ (\epsilon_{hd}\ \delta_1)\ (\epsilon_{hd}\ \delta_2)$

$where\ \delta_1 \in \Delta_{\Pi_{HOL}}(\Gamma \Rightarrow_X \phi \supset \phi'), \delta_2 \in \Delta_{\Pi_{HOL}}(\Gamma \Rightarrow_X \phi).$

$\epsilon_{hd}\ (forall\_i(\Gamma \Rightarrow_X \forall x : \tau.\phi, [\delta]))\ =$

$forall\_i\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ \Gamma)\ (\epsilon_{vs}\ X)\ encx$

$(\epsilon_{ht}\ (addfvar\_Hvst\ encx\ (\epsilon_{vs}\ X))\ \phi)\ \ wfenvpr\ (\epsilon_{hd}\ \delta)$

$where\ \delta \in \Delta_{\Pi_{HOL}}(\Gamma \Rightarrow_{X \cup \{x:\tau\}} \phi)$

$encx\ =\ mkpair\ Holvar\ (\epsilon_{vn}\ x)\ (\epsilon_\tau\ \tau)$

$\epsilon_{hd}\ forall\_e(\Gamma \Rightarrow_X \phi\{t/x_\tau\}, [\delta_1, \delta_2, \delta_3])\ =$

$forall\_e\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ \Gamma)\ (\epsilon_{vs}\ X)\ encx$

$(\epsilon_{ht}\ (addfvar\_Hvst\ encx\ (\epsilon_{vs}\ X))\ \phi)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ t)\ \ wfenvpr\ (\epsilon_{hd}\ \delta_1)$

$(\epsilon_{td}\ \delta_2)\ (\epsilon_{td}\ \delta_3)$

$where\ \delta_1 \in \Delta_{\Pi_{HOL}}(\Gamma \Rightarrow_X \forall x : \tau.\phi), \delta_2 \in \Delta_{\Pi_{HOL}}(X \cup \{x : \tau\} \blacktriangleright \phi : Prop)$

$\delta_3 \in \Delta_{\Pi_{HOL}}(X \blacktriangleright t : \tau)$

$encx\ =\ mkpair\ Holvar\ (\epsilon_{vn}\ x)\ (\epsilon_\tau\ \tau)$

$$\epsilon_{hd}\ (conv(\Gamma \Rightarrow_X \psi, [\delta_1, \delta_2]))\ =$$

$$conv\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ \Gamma)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \psi)\ wfenvpr$$

$$(\epsilon_{td}\ \delta_1)\ (\epsilon_\beta\ \delta_2)$$

$$where\ \delta_1 \in \Delta_{\Pi_{HOL}}(\Gamma \Rightarrow_X \phi), \delta_2 \in \Delta_{\Pi_{HOL}}(\psi =_\beta \phi)$$

Proposition F.106 *For any signature* $\Sigma \in |AlgSig|$*, for any* $\Gamma \in \mathcal{P}(Sen_{HOL}(\Sigma, X, \mathbf{Prop}))$ *and for any* $\phi \in Term_{HOL}(\Sigma, X)$*, if* $\Gamma \Rightarrow_X \phi$ *then* $X \blacktriangleright \phi : \tau$

Proof:
By induction on the derivations of $\Gamma \Rightarrow_X \phi$.

Proposition F.107 *For any signature* Σ*, for any type* $\tau \in Holtype$*, for any variable set* $vs : Holvar\_set$*, for any higher-order term* $ht : Holterm$ *and for any list of higher-order terms* $htl : Holterm\_list$*, if* $HOL\ htl\ vs\ ht$ *is inhabited then* $Wfhterm\ vs\ ht$ **Prop** *is inhabited.*

Proof:
By induction on the derivations of $HOL\ htl\ vs\ ht$.

Theorem F.108 *For any* $\Gamma \in \mathcal{P}(Sen_{HOL}(\Sigma, X, \mathbf{Prop}))$ *and for any* $\phi \in Sen_{HOL}(\Sigma, X)$*, there exists a bijection between the closed derivations of a judgement* $(\Gamma \Rightarrow_X \phi)$ *and the normal forms of the proofs of the proposition*

$$HOL\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ \Gamma)\ (\epsilon_{vs}\ X)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)$$

Proof:
To prove the bijection we define a decoding function with type

$$\epsilon_{hd}^{-1} : (HOL\ (\epsilon_{htl}\ (\epsilon_{vs}\ X)\ \Gamma)\ (\epsilon_{vs}\ X)\ (\epsilon_{ht}\ (\epsilon_{vs}\ X)\ \phi)) \rightarrow \Delta_{\Pi_{HOL}}(\Gamma\ \Rightarrow_X\ \phi)$$

inductively defined as follows:

$$\epsilon_{hd}^{-1}\ (impl\_i\ htl\ vs\ ht\ ht'\ wfenpr\ prt\ prd)\ =$$

$$impl\_i((\epsilon_{htl}^{-1}\ vs\ htl)\ \Rightarrow_{\epsilon_{vs}^{-1}\ vs}\ (\epsilon_{ht}^{-1}\ vs\ ht) \supset (\epsilon_{ht}^{-1}\ vs\ ht'), [\epsilon_{td}^{-1}\ prt, \epsilon_{hd}^{-1}\ prd])$$

$$\epsilon_{hd}^{-1}\ (impl\_e\ env\ vs\ ht\ ht'\ wfenpr\ prd\ prd')\ =$$

$$impl\_e((\epsilon_{htl}^{-1}\ vs\ env) \Rightarrow_{\epsilon_{vs}^{-1}\ vs}\ (\epsilon_{ht}^{-1}\ vs\ ht'), [\epsilon_{hd}^{-1}\ prd, \epsilon_{hd}^{-1}\ prd']$$

$$\epsilon_{hd}^{-1}\ (forall\_i\ env\ vs\ hv\ ht\ wfenpr\ \ dpr)\ =$$

$$forall\_i\ (\epsilon_{htl}^{-1}\ vs\ env)\ \Rightarrow_{(\epsilon_{vs}^{-1}\ vs)}\ \forall \epsilon_{vn}^{-1}\ (fst\ hv) : \epsilon_{\tau}^{-1}\ (snd\ hv).$$

$$(\epsilon_{ht}^{-1}\ (addfvar\_Hvst\ hv\ vs)\ ht), [\epsilon_{hd}^{-1}\ dpr])$$

$$\epsilon_{hd}^{-1}\ (forall\_e\ env\ vs\ hv\ ht\ ht'\ wfenpr\ wft\ wft'\ dpr)\ =$$

$$forall\_e\ (\epsilon_{htl}^{-1}\ vs\ env)\ \Rightarrow_{\epsilon_{vs}^{-1}\ vs}\ (\epsilon_{ht}^{-1}\ (addfvar\_Hvst\ hv\ vs)\ ht)$$

$$\{(\epsilon_{ht}^{-1}\ vs\ ht')\ /\ (\epsilon_{vn}^{-1}\ (fst\ hv)\}\ [\epsilon_{hd}^{-1}\ dpr, \epsilon_{dt}^{-1}\ wft, \epsilon_{dt}^{-1}\ wft'])$$

$$\epsilon_{hd}^{-1}\ (conv\ env\ vs\ ht\ ht\ wfenpr\ prd\ beqpr)\ =$$

$$conv((\epsilon_{htl}^{-1}\ (\epsilon_{vs}\ vs)\ env) \Rightarrow_{(\epsilon_{vs}^{-1}\ vs)}\ (\epsilon_{ht}^{-1}\ (\epsilon_{vs}\ X)\ ht'), [\epsilon_{dt}^{-1}\ prd, \epsilon_{\beta}^{-1}\ beqpr])$$

The rest of the proof is as explained in Chapter 3.

G Predefined functions of chapter 6

G.1 Functions on signature morphisms

Definition G.1 *The function* $get\_dom\_sm : Signature\_morphism \rightarrow Signature$ *is defined as follows:*

$$get\_dom\_sm\ signm\ = (sort\_sl\ (get\_dom\_spl\ (fst\ (snd\ signm))),$$

$$sort\_opl\ (get\_dom\_oppl\ (snd\ (snd\ signm))))$$

$$get\_dom\_spl\ spl\ = map\ fst\ spl$$

$$get\_dom\_oppl\ oppl\ = map\ fst\ oppl$$

Definition G.2 *The function* $get\_ran\_sm : Signature\_morphism \rightarrow Signature$ *is defined as follows:*

$$get\_ran\_sm\ signm\ = (sort\_sl\ (get\_ran\_spl\ (fst\ (snd\ signm))),$$

$$sort\_opl\ (get\_ran\_oppl\ (snd\ (snd\ signm))))$$

$$get\_ran\_spl\ spl\ = map\ snd\ spl$$

$$get\_ran\_oppl\ oppl\ = map\ snd\ oppl$$

Definition G.3 *The function* $inverse\_sm : Signature\_morphism \rightarrow Signature\_morphism$ *is defined as follows:*

$$inverse\_sm\ sm\ =\ mkpair\ (get\_ran\_sm\ sm)\ (invert\_pairs\ sm)$$

$$where$$

$$invert\_pairs\ sm\ = mkpair(invp\_sl(fst\ (snd\ sm)))(invp\_opl(snd\ (snd\ sm)))$$

$$invp\_sl\ sl\ =\ map\ invp\ sl$$

$$invp\_opl\ sl\ =\ map\ invp\ opl$$

$$invp\ p\ =\ (snd\ p, fst\ p)$$

G.2 Operations on signatures and specification expressions

Definition G.4 *The function* $new\_index : Signature \rightarrow Sym\_index \rightarrow Signature$ *is defined as follows:*

$$new\_index\ sign\ ind\ =\ mkpair\ (map\ (updinds\ ind)\ (fst\ sign)\ )$$

$$(map\ (updindop\ ind)\ (snd\ sign))$$

$$where$$

$$updinds\ s\ ind\ =\ (fst\ s, ind)$$

$$updindop\ op\ ind\ =\ (fst\ op, ind)$$

Definition G.5 *The function* $union\_Sign : Signature \rightarrow Signature \rightarrow Signature$ *is defined as follows:*

$$union\_Sign\ sign\ sign'\ =\ mkpair\ (union\_Srt\ (first\ sign)\ (first\ sign'))$$

$$(union\_Ops\ (snd\ sign)\ (snd\ sign'))$$

Definition G.6 *The function* $union\_Srt : (List\ Ind\_sorts) \rightarrow (List\ Ind\_sorts) \rightarrow (List\ Ind\_sorts)$ *is defined as follows:*

$$union\_Srts\ l\ l' \ = \ Primrec\ (List\ Ind\_sorts)\ l'\ genc\_uSrts\ l$$

$where$

$genc\_uSrts\ s\ sl\ slf = add\_if\_not\_in\_sl\ s\ slf$

$add\_if\_not\_in\_sl\ s\ sl \ = \ Primrec\ Bool\ (cons\ s\ sl)\ sl\ (not\_in\_sl\ s\ sl)$

$not\_in\_sl\ s\ sl \ = \ Primrec(List\ Ind\_sorts)\ true\ (genc\_ninsl\ s)\ sl$

$genc\_ninsl\ s\ s'\ sl\ b \ =$

$$Primrec\ bool(not\_bool\ Eqbool\_Isrts\ s\ s')\ b\ b$$

Definition G.7 *The function* $union\_Ops : (List\ Ind\_ops) \rightarrow (List\ Ind\_ops) \rightarrow (List\ Ind\_ops)$ *is defined as follows:*

$$union\_Ops\ l\ l' \ = \ Primrec\ (List\ Ind\_ops)\ l'\ genc\_uOps\ l$$

$where$

$genc\_uOps\ op\ opl\ oplf = add\_if\_not\_in\_opl\ op\ oplf$

$add\_if\_not\_in\_opl\ op\ opl \ = \ Primrec\ Bool\ (cons\ op\ opl)\ opl\ (not\_in\_opl\ op\ opl)$

$not\_in\_opl\ op\ opl \ = \ Primrec\ (list\ Ind\_ops)\ true\ (genc\_ninopl\ op)\ opl$

$genc\_ninopl\ op\ op'\ opl\ b = \ Primrec\ bool\ (not\_bool\ (Eqbool\_Iops\ op\ op'))\ b\ b$

Definition G.8 *The function* $intersect\_Sign : Signature \rightarrow Signature \rightarrow$

Signature is defined as follows:

$$inrtersect\_Sign\ sign\ sign' \ = \ mkpair\ (fst\ (inter\_Srt\ (first\ sign)\ (first\ sign')))$$

$$(fst\ (inter\_Ops\ (snd\ sign)\ (snd\ sign')))$$

$$where$$

$$inter\_Srt\ sl\ sl' \ = \ Primrec\ (List\ Ind\_sorts)\ (nil, sl)\ addifinsecsl\ sl'$$

$$addifinsecsl\ s\ sl\ psl \ = \ Primrec\ bool\ (cons\ s\ (fst\ psl), snd\ psl)$$

$$psl\ (is\_in\_bool\ Eqbool\_Isrts\ (snd\ psl))$$

$$inter\_Ops\ opl\ opl' \ = \ Primrec\ (List\ Ind\_ops)\ (nil, opl)\ addifinsecopl\ sl'$$

$$addifinsecopl\ op\ opl\ popl \ = \ Primrec\ bool\ (cons\ s\ (fst\ popl), snd\ popl)$$

$$popl\ (is\_in\_bool\ Eqbool\_Iops\ (snd\ psl))$$

Definition G.9 *The function* $diff\_Sign : Signature \to Signature \to Signature$ *is defined as follows:*

$$diff\_Sign\ sign\ sign' \ = \ mkpair\ (diff\_Srt\ (first\ sign)\ (first\ sign'))$$

$$(diff\_Ops\ (snd\ sign)\ (snd\ sign'))$$

$$where$$

$$diff\_Srt\ sl\ sl' \ = Primrec\ (List\ Ind\_sorts)\ sl\ gencsl\_diff\ sl'$$

$$gencsl\_diff\ s\ sl\ sl' \ = \ remove\ Eqbool\_Isrts\ s\ sl'$$

$$diff\_Ops\ opl\ opl' \ = Primrec\ (List\ Ind\_ops)\ opl\ gencopl\_diff\ opl'$$

$$gencopl\_diff\ op\ opl\ opl' \ = \ remove\ Eqbool\_Iops\ op\ opl'$$

Definition G.10 *The function* $nameclash\_sign : Signature \ \to \ Signature \ \to \ Signature \ \to \ Signature$ *is defined as follows:*

$$nameclash\_sign\ signsp\ sign\ signsp' \ =$$

$$diff\_sign\ (intersect\_sign\ signsp\ signsp')\ sign$$

Definition G.11 *The function* $Signature\_ind\_sp : Specification \ \to \ Sym\_index \ \to$

Signature is defined as follows:

$$Signature\_ind\_sp\ sp\ ind = Primrec\ Specification\ (basec\_sign\ ind)\ (sumc\_sign\ ind)\ (expc\_sign\ ind)$$

$$(renc\_sign\ ind)\ (reachc\_sign\ ind)\ (behc\_sign\ ind)\ (quotc\_sign\ ind)\ (abstrc\_sign\ ind)\ sp$$

$$where$$

$$basec\_sign\ ind\ sign\ htl\ =\ (new\_index\ sign\ ind, ind)$$

$$sumc\_sign\ ind\ sp\ sign\ sp'\ signsp\ signsp'\ =$$

$$mkpair\ (union\_sign\ (new\_index\ (nameclash\_sign\ (fst\ signsp)$$

$$sign\ (fst\ signsp'))$$

$$(next\_Si\ (maxind\_Si\ (snd\ signsp)\ (snd\ signsp'))))$$

$$(union\_sign\ (diff\_sign\ (diff\_sign\ (fst\ signsp)\ sign)$$

$$(nameclash\_sign\ (fst\ signsp)\ sign\ (fst\ signsp')))\ (fst\ signsp'))$$

$$(next\_Si\ (maxind\_Si\ (snd\ signsp)\ (snd\ signsp')))$$

$$renc\_sign\ ind\ sp\ signm\ signsp\ =\ (get\_ran\_sm\ signm, ind)$$

$$expc\_sign\ ind\ sp\ sign\ signsp\ =\ (sign, ind)$$

$$reachc\_sign\ ind\ sp\ reachsgn\ signsp\ =\ signsp$$

$$behc\_sign\ ind\ sp\ obssl\ inssl\ signsp\ =\ signsp$$

$$absc\_sign\ ind\ sp\ obssl\ inssl\ signsp\ =\ signsp$$

$$quoc\_sign\ ind\ sp\ obssl\ inssl\ signsp\ =\ signsp$$

Definition G.12 *The function* $Signature\_sp : Specification\ \to\to\ Signature$ *is defined as follows:*

$$Signature\_sp\ sp\ =\ fst\ (Signature\_ind\_sp\ sp\ first\_Vi)$$

G.3 Some inductive relations

Definition G.13 *The inductive relation* $Same\_signature : \Pi sign, sign' : Signature.Prop$ *is defined by the following set of constructors:*

$$basec\_Sams : \Pi sign : Same\_signature\ (mkpair\ (nil\ Ind\_sorts)\ (nil\ Ind\_ops))$$

$$(mkpair\ (nil\ Ind\_sorts)\ (nil\ Ind\_ops))$$

$$gencs\_Sams : \Pi s : Ind\_sorts.\Pi sign, sign' : Signature.\Pi sams : Same\_signature\ sign\ sign'.$$

$$Same\_signature\ (sort\_sl\ (cons\ s\ (fst\ sign), (snd\ sign)))$$

$$(sort\_sl\ (cons\ s\ (fst\ sign'), (snd\ sign')))$$

$$gencop\_Sams : \Pi op : Ops.\Pi sign, sign' : Signature.\Pi sams : Same\_signature\ sign\ sign'.$$

$$Same\_signature\ (fst\ sign, (sort\_opl\ (consop\ (snd\ sign)))$$

$$(fst\ sign, (sort\_opl\ (cons\ op\ (snd\ sign)))$$

Definition G.14 *The inductive relation* $Subsignature : \Pi sign, sign' : Signature.Prop$ *is defined by the following set of constructors:*

$$basec\_Subsign : \Pi sign : Signature.Subsignature\ (mkpair\ (nil\ Ind\_sorts)\ (nil\ Ind\_ops))\ sign$$

$$gencs\_Subsign : \Pi s : Ind\_sorts.\Pi sign, sign' : Signature.$$

$$\Pi isins : Is\_in\_List\ s\ (fst\ sign').$$

$$Subsignature\ (sort\_sl\ (cons\ s\ (fst\ sign), (snd\ sign)))\ sign'$$

$$gencop\_Subs : \Pi op : Ops.\Pi sign, sign' : Signature.\Pi isins : Is\_in\_List\ op\ (snd\ sign').$$

$$Subsignature\ (fst\ sign, (sort\_opl\ (cons\ op\ (snd\ sign)))\ sign'$$

Definition G.15 *The inductive relation* $Subsorts : \Pi sl : List\ Ind\_sorts.sign' : Signature.Prop$ *is defined by the following set of constructors:*

$$basec\_Subs : \Pi sign : Signature.Subsorts\ (nil\ Ind\_sorts)\ sign$$

$$gencs\_Subs : \Pi s : Ind\_sorts.\Pi sl : List\ Ind\_sorts.\Pi sign : Signature.$$

$$\Pi isins : Is\_in\_List\ s\ (fst\ sign').$$

$$Subsorts\ (sort\_sl\ (cons\ s\ sl))\ sign'$$

Definition G.16 *The inductive relation*

$$Bijective : \Pi sign : Signature.\Pi signm : Signature\_morphism.Prop$$

is defined by the following constructors:

$bij\_ctr : \Pi sign : Signature.\Pi signm : Signature\_morphism.$

$\Pi norepssd : Norep\_list\ Ind\_sorts\ Eqbool\_Isrts\ (fst\ (get\_dom\_sm\ signm)).$

$\Pi norepsst : Norep\_list\ Ind\_sorts\ Eqbool\_Isrts\ (fst\ (get\_ran\_sm\ signm)).$

$\Pi norepsopd : Norep\_list\ Ind\_ops\ Eqbool\_Iops\ (snd\ (get\_dom\_sm\ signm)).$

$\Pi norepsopt : Norep\_list\ Ind\_ops\ Eqbool\_Iops\ (snd\ (get\_ran\_sm\ signm)).$

$\Pi samesignd : Same\_signature\ sign\ (get\_dom\_sm\ signm).$

$\Pi samesignt : Same\_signature\ (first\ signm)\ (get\_ran\_sm\ signm).$

$Bijective\ sign\ signm$

G.4 Operations associated to the pushouts morphisms of structured specifications

Definition G.17 *The function* $inl\_sums : Specification \rightarrow Signature \rightarrow Specification \rightarrow Signature$ *is defined as follows:*

$$inl\_sums\ sp\ sign\ sp' = Signature\_sp\ sp$$

Definition G.18 *The function* $inr\_sums : Specification \rightarrow Signature \rightarrow Specification \rightarrow Signature$ *is defined as follows:*

$inr\_sums\ sp\ sign\ sp' =$

$union\_sign\ (new\_index\ (nameclash\_sign\ (Signature\_sp\ sp)\ sign\ (Signature\_sp\ sp'))$

$(next\_Vi\ (maxind\_Si\ (snd\ (Signature\_ind\_sp\ sp\ first\_Vi))$

$(snd\ (Signature\_ind\_sp\ sp'\ first\_Vi)))))$

$(diff\_sign\ (Signature\_sp\ sp')\ (nameclash\_sign\ (Signature\_sp\ sp)\ sign(Signature\_sp\ sp')))$

Definition G.19 *The function* $inlsm\_sums : Specification \rightarrow Signature \rightarrow$

$Specification \ \rightarrow \ Signature\_morphism$ *is defined as follows:*

$inlsm\_sums\ sp\ sign\ sp' \ =$

$(join\ Ind\_sorts\ Ind\_sorts\ (fst\ (Signature\_sp\ sp))\ \ (fst\ (Signature\_sp\ sp)),$

$join\ Ind\_ops\ Ind\_ops\ (snd\ (Signature\_sp\ sp))(snd\ (Signature\_sp\ sp)))$

Definition G.20 *The function* $inrsm\_sums : Specification \ \rightarrow \ Signature \ \rightarrow$ $Specification \ \rightarrow \ Signature\_morphism$ *is defined as follows:*

$inrsm\_sums\ sp\ sign\ sp' \ =$

$(concat\ (prod\ Ind\_sorts\ Ind\_sorts)\ (join\ Ind\_sorts\ Ind\_sorts$

$(Fst\ (nameclash\_sign\ (Signature\_sp\ sp)\ sign\ (Signature\_sp\ sp')))$

$(Fst\ (new\_index\ (nameclash\_sign\ (Signature\_sp\ sp)\ sign\ (Signature\_sp\ sp'))$

$(next\_Vi\ (maxind\_Si\ (snd\ (Signature\_ind\_sp\ sp\ first\_Vi))$

$(snd\ (Signature\_ind\_sp\ sp'\ first\_Vi)))))))$

$(join\ Ind\_sorts\ Ind\_sorts\ (Fst\ (diff\_sign\ (Signature\_sp\ sp'\ first\_Si)$

$(nameclash\_sign\ (Signature\_sp\ sp\ first\_Si)\ sign\ (Signature\_sp\ sp'\ first\_Si))))$

$(Fst\ (diff\_sign\ (Signature\_sp\ sp'\ first\_Si)\ (nameclash\_sign$

$(Fst\ (Signature\_sp\ sp\ first\_Si))\ sign\ (Signature\_sp\ sp'\ first\_Si)))))),$

$(concat\ (prod\ Ind\_ops\ Ind\_ops)\ (join\ Ind\_ops\ Ind\_ops$

$(snd\ (nameclash\_sign\ (Signature\_sp\ sp\ first\_Si)\ sign\ (Signature\_sp\ sp'\ first\_Si)))$

$(snd\ (new\_index\ (nameclash\_sign\ (Signature\_sp\ sp\ first\_Si)$

$sign\ (Signature\_sp\ sp'\ first\_Si))$

$(next\_Vi\ (maxind\_Si\ (snd\ (Signature\_ind\_sp\ sp\ first\_Vi))$

$(snd\ (Signature\_ind\_sp\ sp'\ first\_Vi)))))))$

$(join\ Ind\_ops\ Ind\_ops\ (snd\ (diff\_sign\ (Signature\_sp\ sp'\ first\_Si)$

$(nameclash\_sign\ (Signature\_sp\ sp\ first\_Si)\ sign\ (Signature\_sp\ sp'\ first\_Si))))$

$(snd\ (diff\_sign\ (Signature\_sp\ sp'\ first\_Si)\ (nameclash\_sign$

$(Signature\_sp\ sp\ first\_Si)\ sign\ (Signature\_sp\ sp'\ first\_Si))))))$