Formalization of Pure Type System

1. Definition

(i) A *pure type system* (*PTS*) is a triple tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ where

   (a) $\mathcal{S}$ is a set of *sorts*;

   (b) $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ is a set of *axioms*;

   (c) $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ is a set of *rules*.

(ii) *Raw expressions* A and *raw environments* Γ are defined by

$$
\begin{aligned}
A &::= x \mid s \mid AA \mid \lambda x : A.\ A \mid \Pi x : A.\ A \\
\Gamma &::= \varnothing \mid \Gamma, x : A
\end{aligned}
$$

where we use s, t, u, etc., to range over sorts, x, y, z, etc., to range over variables, and A, B, C, a, b, c, etc., to range over expressions.

(iii) Π and λ are used to bind variables. Let $\mathrm{FV}(A)$ denote free variable set of A. Let $A[x := B]$ denote the substitution of x in A with B. Standard notational conventions are applied here. Besides we also let $A \to B$ be an abbreviation for $(\Pi_{\_} : A.\ B)$.

(iv) The relation $\to_\beta$ is the smallest binary relation on raw expressions satisfying

$$(\lambda x : A.\ M)N \to_\beta M[x := N]$$

which can be used to define the notation $\twoheadrightarrow_\beta$ and $=_\beta$ by convention.

(v) Type assignment rules for $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ are given in Table 1. Particularly, the rule *(Conv)* is needed to make everything work.

2. Examples of PTSs

(i) The λ-cube (Table 2) consists of eight PTSs, where

   (a) $\mathcal{S} = \{\star, \square\}$

   (b) $\mathcal{A} = \{(\star, \square)\}$

   (c) $\{(\star, \star)\} \subseteq \mathcal{R} \subseteq \{(\star, \star), (\star, \square), (\square, \star), (\square, \square)\}$

   Note that here we slightly abuse the notation of the set of rules $\mathcal{R}$, since in PTSs, $\mathcal{R}$ is a ternary relation, while in the λ-cube, $\mathcal{R}$ is a binary relation ($\Pi x : A.\ B$ has the same sorts as B).

(ii) An extension of $\lambda\omega$ that supports "polymorphic identity function on types", where

$$\text{(Ax)} \quad \frac{}{\vdash s : t} \quad (s,t) \in \mathcal{A}$$

$$\text{(Var)} \quad \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \quad x \notin \mathrm{dom}(\Gamma)$$

$$\text{(Weak)} \quad \frac{\Gamma \vdash b : B \qquad \Gamma \vdash A : s}{\Gamma, x : A \vdash b : B} \quad x \notin \mathrm{dom}(\Gamma)$$

$$\text{(App)} \quad \frac{\Gamma \vdash f : (\Pi x : A.\, B) \qquad \Gamma \vdash a : A}{\Gamma \vdash fa : B[x := a]}$$

$$\text{(Lam)} \quad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (\Pi x : A.\, B) : t}{\Gamma \vdash (\lambda x : A.\, b) : (\Pi x : A.\, B)}$$

$$\text{(Pi)} \quad \frac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : t}{\Gamma \vdash (\Pi x : A.\, B) : u} \quad (s,t,u) \in \mathcal{R}$$

$$\text{(Conv)} \quad \frac{\Gamma \vdash a : A \qquad \Gamma \vdash B : s \qquad A =_\beta B}{\Gamma \vdash a : B}$$

Table 1. Typing rules for pure type system

System	Set of Rules $\mathcal{R}$			
$\lambda_\rightarrow$	$(\star,\star)$			
$\lambda 2$	$(\star,\star)$	$(\Box,\star)$		
$\lambda\underline{\omega}$	$(\star,\star)$		$(\Box,\Box)$	
$\lambda\omega$	$(\star,\star)$	$(\Box,\star)$	$(\Box,\Box)$	
λP	$(\star,\star)$			$(\star,\Box)$
λP2	$(\star,\star)$	$(\Box,\star)$		$(\star,\Box)$
λP$\underline{\omega}$	$(\star,\star)$		$(\Box,\Box)$	$(\star,\Box)$
λC	$(\star,\star)$	$(\Box,\star)$	$(\Box,\Box)$	$(\star,\Box)$

Table 2. The systems of the λ-cube

(a) $\mathcal{S} = \{\star, \Box, \Box'\}$

(b) $\mathcal{A} = \{(\star,\Box), (\Box,\Box')\}$

(c) $\mathcal{R} = \{(\star,\star), (\Box,\star), (\Box,\Box), (\Box',\Box')\}$

in which we can have $\vdash (\lambda\kappa : \Box.\ \lambda\alpha : \kappa.\ \alpha) : (\Pi\kappa : \Box.\ \kappa \rightarrow \kappa)$, justified as follows:

$$\dfrac{\dfrac{\dfrac{\mathcal{B}}{\kappa : \Box, \alpha : \kappa \vdash \alpha : \kappa}\,\mathit{Var} \qquad \mathcal{A}}{\kappa : \Box \vdash (\lambda\alpha : \kappa.\ \alpha) : (\Pi\alpha : \kappa.\ \kappa)}\,\mathit{Lam} \qquad \dfrac{\dfrac{}{\vdash \Box : \Box'}\,\mathit{Ax} \qquad \mathcal{A}}{\vdash (\Pi\kappa : \Box.\ \Pi\alpha : \kappa.\ \kappa) : \Box}\,\mathit{Pi}}{\vdash (\lambda\kappa : \Box.\ \lambda\alpha : \kappa.\ \alpha) : (\Pi\kappa : \Box.\ \Pi\alpha : \kappa.\ \kappa)}\,\mathit{Lam}$$

$$\mathcal{A} = \quad \dfrac{\mathcal{B} \qquad \dfrac{\mathcal{B} \qquad \mathcal{B}}{\kappa : \Box, \alpha : \kappa \vdash \kappa : \Box}\,\mathit{Weak}}{\kappa : \Box \vdash (\Pi\alpha : \kappa.\ \kappa) : \Box}\,\mathit{Pi}$$

$$\mathcal{B} = \quad \dfrac{\dfrac{}{\vdash \Box : \Box'}\,\mathit{Ax}}{\kappa : \Box \vdash \kappa : \Box}\,\mathit{Var}$$

3. Extending PTSs

This section investigates how to extend PTSs to have algebraic datatypes, case expressions, etc.

3.1 Algebraic Datatypes

An algebraic datatype has the form:

$$T\ u_1 \dots u_k = K_1\ t_{11} \dots t_{1k_1} \mid \cdots \mid K_n\ t_{n1} \dots t_{nk_n}$$

where T denotes a new *type constructor* with zero or more constituent *data constructors* $K_1, \dots, K_n$. We call $u_1, \dots, u_k$ the arguments of the type constructor T, and $t_{j1}, \dots, t_{jk_j}$ the types of the arguments of the $K_j (1 \leqslant j \leqslant n)$ data constructor. Each u_i is a variable of *sort type*, each t_{jk} is an expression of *kind type* (i.e., $t_{jk} : \star$), which may contain T and $u_1, \dots, u_k$ as free variables.

We use the following notation: $\vec{\mathbf{u}} = [u_1, \dots, u_k]$, $\vec{\mathbf{t_j}} = [t_{j1}, \dots t_{jk_j}]$, etc. If $\vec{\mathbf{a}} = [a_1, \dots, a_n]$ and $\vec{\mathbf{A}} = [A_1, \dots, A_n]$, then $\Pi\vec{\mathbf{a}} : \vec{\mathbf{A}}.\ B$ denotes $\Pi a_1 : A_1 \dots \Pi a_n : A_n.\ B$. Let $\tau_1, \dots, \tau_k$ be the types of $u_1, \dots, u_k$, respectively.

A PTS with ADTs is a tuple $(P, ADTS)$ where:

(i) P is a Pure Type System, let V, E be the sets of variables and expressions of P.

(ii) $ADTS$ is a set of ADTs, each consisting of $[T : T', K_1 : K'_1, \dots, K'_n]$ such that:

- $T, K_j \in V$ and $T', K'_j \in E$, for every $1 \leqslant j \leqslant n$
- $T' = \Pi\vec{\mathbf{u}} : \vec{\tau}.\ \star$
- $K'_j = \Pi\vec{\mathbf{u}} : \vec{\tau}.\ \Pi\vec{\alpha} : \vec{\mathbf{t_j}}.\ (T\ \vec{\mathbf{u}})$, for every $1 \leqslant j \leqslant n$
- $T : T' \vdash K_j : K'_j : \star$

Note that the use of the dependent product (Π) makes it possible to let the types of the data constructor arguments depend on other data constructor arguments.

(iii) (**Typability in a PTS with ADTS**) let $\Sigma = [c : ct \mid c : ct \leftarrow ADT, ADT \leftarrow ADTS]$, we say that $\Gamma \vdash_{(P,ADTS)} a : A$ if and only if $\Sigma \uplus \Gamma \vdash a : A$

3.1.1 An Example of PTSs with ADTs

Let $P = \lambda C$ and

$$\begin{aligned}
\Sigma_a &= [Int : \star, Zero : Int, Suc : Int \to Int,\\
&\qquad Bool : \star, True : Bool, False : Bool]\\
\Sigma_b &= [Vec : (\Pi n : Int.\ \Pi\alpha : \star.\ \star),\\
&\qquad Nil : (\Pi\alpha : \star.\ Vec\ Zero\ \alpha),\\
&\qquad Cons : (\Pi n : Int.\ \Pi\alpha : \star.\ \alpha \to Vec\ n\ \alpha \to Vec\ (Suc\ n)\ \alpha)]\\
ADTS &= [\Sigma_a, \Sigma_b]
\end{aligned}$$

then we can derive $\vdash_{(P,ADTS)}$ Cons Zero Bool True (Nil Bool) : Vec (Suc Zero) Bool

References

[1] Simon Peyton Jones and Erik Meijer. Henk: a typed intermediate language. *TIC*, 97, 1997.

[2] J-W Roorda and JT Jeuring. Pure type systems for functional programming. 2007.

[3] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*, volume 149. Elsevier, 2006.