

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

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

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 $\text{FV}(A)$ 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 $A \rightarrow B$ be an abbreviation for $(\Pi x : A. B)$.

(iv) The relation \rightarrow_β is the smallest binary relation on raw expressions satisfying

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

which can be used to define the notation \twoheadrightarrow_β 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

(Ax)	$\frac{}{\vdash s : t}$	$(s, t) \in \mathcal{A}$
(Var)	$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$	$x \notin \text{dom}(\Gamma)$
(Weak)	$\frac{\Gamma \vdash b : B \quad \Gamma \vdash A : s}{\Gamma, x : A \vdash b : B}$	$x \notin \text{dom}(\Gamma)$
(App)	$\frac{\Gamma \vdash f : (\Pi x : A. B) \quad \Gamma \vdash a : A}{\Gamma \vdash fa : B[x := a]}$	
(Lam)	$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x : A. B) : t}{\Gamma \vdash (\lambda x : A. b) : (\Pi x : A. B)}$	
(Pi)	$\frac{\Gamma \vdash A : s \quad \Gamma, x : A \vdash B : t}{\Gamma \vdash (\Pi x : A. B) : u}$	$(s, t, u) \in \mathcal{R}$
(Conv)	$\frac{\Gamma \vdash a : A \quad \Gamma \vdash B : s \quad A =_{\beta} B}{\Gamma \vdash a : B}$	

Table 1. Typing rules for pure type system

System	Set of Rules \mathcal{R}
λ_{\rightarrow}	(\star, \star)
λ_2	$(\star, \star) \quad (\square, \star)$
λ_{ω}	$(\star, \star) \quad (\square, \square)$
λ_{ω}	$(\star, \star) \quad (\square, \star) \quad (\square, \square)$
λ_P	$(\star, \star) \quad (\star, \square)$
λ_{P2}	$(\star, \star) \quad (\square, \star) \quad (\star, \square)$
$\lambda_{P\omega}$	$(\star, \star) \quad (\square, \square) \quad (\star, \square)$
λ_C	$(\star, \star) \quad (\square, \star) \quad (\square, \square) \quad (\star, \square)$

Table 2. The systems of the λ -cube

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

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

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

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

$$\frac{\frac{\frac{\mathcal{B}}{\kappa : \square, \alpha : \kappa \vdash \alpha : \kappa} \text{Var} \quad \mathcal{A}}{\kappa : \square \vdash (\lambda \alpha : \kappa. \alpha) : (\Pi \alpha : \kappa. \kappa)} \text{Lam} \quad \frac{\frac{\frac{}{\vdash \square : \square'} \text{Ax} \quad \mathcal{A}}{\vdash (\Pi \kappa : \square. \Pi \alpha : \kappa. \kappa) : \square} \text{Pi}}{\vdash (\lambda \kappa : \square. \lambda \alpha : \kappa. \alpha) : (\Pi \kappa : \square. \Pi \alpha : \kappa. \kappa)} \text{Lam}$$

$$\mathcal{A} = \frac{\mathcal{B} \quad \frac{\frac{\mathcal{B}}{\kappa : \square, \alpha : \kappa \vdash \kappa : \square} \text{Weak} \quad \mathcal{B}}{\kappa : \square \vdash (\Pi \alpha : \kappa. \kappa) : \square} \text{Pi}}$$

$$\mathcal{B} = \frac{\frac{}{\vdash \square : \square'} \text{Ax}}{\kappa : \square \vdash \kappa : \square} \text{Var}$$

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:

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

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 ($1 \leq j \leq n$) 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: $\vec{u} = [u_1, \dots, u_k]$, $\vec{t}_j = [t_{j1}, \dots, t_{jk_j}]$, etc. If $\vec{a} = [a_1, \dots, a_n]$ and $\vec{A} = [A_1, \dots, A_n]$, then $\Pi \vec{a} : \vec{A}. B$ denotes $\Pi a_1 : A_1 \dots \Pi a_n : A_n. B$. Let τ_1, \dots, τ_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 $1 \leq j \leq n$
 - $T' = \Pi \vec{u} : \vec{\tau}. \star$
 - $K'_j = \Pi \vec{u} : \vec{\tau}. \Pi \vec{a} : \vec{t}_j. (T \ \vec{u})$, for every $1 \leq j \leq n$
 - $T : T' \vdash K_j : K'_j : \star$

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.

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.