Formalization of Pure Type Systems

Last modified: March 22, 2015 at 1:03am

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*.

  Following standard practice, we use (s_1, s_2) to denote rules of the form (s_1, s_2, s_2).

(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 3. Particularly, the rule *(Conv)* is needed to make everything work.

2. Examples of PTSs

(i) Here we present the formal definition of a type system called *the calculus of construction (λC)*, where

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

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

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

and the typing relation is shown in Table 1.

(Ax)	$\dfrac{}{\vdash \star : \Box}$	
(Var)	$\dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$	$x \notin \mathrm{dom}(\Gamma)$
(Weak)	$\dfrac{\Gamma \vdash b : B \qquad \Gamma \vdash A : s}{\Gamma, x : A \vdash b : B}$	$x \notin \mathrm{dom}(\Gamma)$
(App)	$\dfrac{\Gamma \vdash f : (\Pi x : A.\, B) \qquad \Gamma \vdash a : A}{\Gamma \vdash f a : B[x := a]}$	
(Lam)	$\dfrac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (\Pi x : A.\, B) : t}{\Gamma \vdash (\lambda x : A.\, b) : (\Pi x : A.\, B)}$	$t \in \{\star, \Box\}$
(Pi)	$\dfrac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : t}{\Gamma \vdash (\Pi x : A.\, B) : t}$	$(s, t) \in \mathcal{R}$
(Conv)	$\dfrac{\Gamma \vdash a : A \qquad \Gamma \vdash B : s \qquad A =_\beta B}{\Gamma \vdash a : B}$	

Table 1. Typing rules for λC

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

(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 $\vdash (\lambda\kappa : \Box.\, \lambda\alpha : \kappa.\, \alpha) : (\Pi\kappa : \Box.\, \kappa \to \kappa)$, 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

3.1 Recursive types

3.1.1 Definition

We extend Calculus of Constructions (λC, see Section 2) with recursive types, namely λC_μ. The raw expressions are extended as follows:

$$
\begin{aligned}
A ::= {} & x \mid \star \mid \square \\
& \mid AA \mid \lambda x : A.A \mid \Pi x : A.A \\
& \mid \mu x.A \mid \mathsf{fold}[A]\, A \mid \mathsf{unfold}[A]\, A \\
& \mid \mathsf{beta}\, A
\end{aligned}
$$

We introduce a new reduction rule for unfold and fold:

$$\mathsf{unfold}[A]\,(\mathsf{fold}[B]\,a) \to a$$

The extended typing rules are shown in Table 2. Compared with λC, the original *Conv* rule is replaced by the new *Beta* rule where the latter only performs one step of β-reduction.

(Ax)	$\dfrac{}{\vdash \star : \square}$	
(Var)	$\dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$	$x \notin \mathrm{dom}(\Gamma)$
(Weak)	$\dfrac{\Gamma \vdash b : B \qquad \Gamma \vdash A : s}{\Gamma, x : A \vdash b : B}$	$x \notin \mathrm{dom}(\Gamma)$
(App)	$\dfrac{\Gamma \vdash f : (\Pi x : A.\ B) \qquad \Gamma \vdash a : A}{\Gamma \vdash fa : B[x := a]}$	
(Lam)	$\dfrac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (\Pi x : A.\ B) : t}{\Gamma \vdash (\lambda x : A.\ b) : (\Pi x : A.\ B)}$	$t \in \{\star, \square\}$
(Pi)	$\dfrac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : t}{\Gamma \vdash (\Pi x : A.\ B) : t}$	$(s, t) \in \mathcal{R}$
(Mu)	$\dfrac{\Gamma, x : s \vdash A : s}{\Gamma \vdash (\mu x.A) : s}$	
(Fold)	$\dfrac{\Gamma \vdash a : (A[x := \mu x.A]) \qquad \Gamma \vdash \mu x.A : s}{\Gamma \vdash (\mathsf{fold}[\mu x.A]\, a) : \mu x.A}$	
(Unfold)	$\dfrac{\Gamma \vdash a : \mu x.A \qquad \Gamma \vdash A[x := \mu x.A] : s}{\Gamma \vdash (\mathsf{unfold}[\mu x.A]\, a) : A[x := \mu x.A]}$	
(Beta)	$\dfrac{\Gamma \vdash a : A \qquad \Gamma \vdash B : s \qquad A \to_\beta B}{\Gamma \vdash (\mathsf{beta}\, a) : B}$	

Table 2. Typing rules for λC_μ

3.1.2 Examples of typable terms

By convention, we can abbreviate a product $\Pi x : A.B$ to $A \to B$ when $x \notin \mathrm{FV}(B)$.

- A polymorphic fixed-point constructor $\mathsf{fix} : (\Pi\alpha : \star.(\alpha \to \alpha) \to \alpha)$ can be defined as follows:

$$\begin{aligned}\mathsf{fix} =& \lambda\alpha : \star.\lambda f : \alpha \to \alpha. \\ & (\lambda x : (\mu\sigma.\sigma \to \alpha).f((\mathsf{unfold}[\mu\sigma.\sigma \to \alpha]\, x)x)) \\ & (\mathsf{fold}[\mu\sigma.\sigma \to \alpha]\, (\lambda x : (\mu\sigma.\sigma \to \alpha).f((\mathsf{unfold}[\mu\sigma.\sigma \to \alpha]\, x)x)))\end{aligned}$$

- Using fix, we can build recursive functions. For example, given a "hungry" type $H = \mu\sigma.\alpha \to \sigma$, the "hungry" function h where

$$h = \lambda\alpha : \star.\mathsf{fix}\,(\alpha \to H)\,(\lambda f : \alpha \to H.\lambda x : \alpha.\mathsf{fold}[H]\, f)$$

  can take arbitrary number of arguments.

3.2 Encoding of Datatypes

- We can encode the type of natural numbers as follow:

$$\mathsf{Nat} = \mu X.\, \Pi(a : \star).\, a \to (X \to a) \to a$$

  then we can define "zero" and "succ" as follows:

$$\begin{aligned}& zero : \mathsf{Nat} \\ & zero = \mathsf{fold}[\mathsf{Nat}]\, \lambda(a : \star)(z : a)(f : \mathsf{Nat} \to a).\, z \\ & succ : \mathsf{Nat} \to \mathsf{Nat} \\ & succ = \lambda(n : \mathsf{Nat}).\, \mathsf{fold}[\mathsf{Nat}]\, \lambda(a : \star)(z : a)(f : \mathsf{Nat} \to a).\, f\, n\end{aligned}$$

  Using fix, we can define a recursive function "plus" as follow:

$$\begin{aligned}plus :\ & \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat} \\ plus =\ & \mathsf{fix}\,(\mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat})\,(\lambda(p : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat})(n : \mathsf{Nat})(m : \mathsf{Nat}). \\ & \quad (\mathsf{unfold}[\mathsf{Nat}]\, n)\, \mathsf{Nat}\, m\, (\lambda(n' : \mathsf{Nat}).\, \mathsf{succ}\,(p\, n'\, m)))\end{aligned}$$

- We can encode the type of lists of natural numbers as follows:

$$\mathsf{List} = \mu X.\, \Pi(a : \star).\, a \to (\mathsf{Nat} \to X \to a) \to a$$

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.

A. Appendix

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

Table 3. Typing rules for a PTS