Formalization of Pure Type Systems

Last modified: March 25, 2015 at 1:18am

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.

$$\text{(Ax)} \qquad \frac{}{\vdash \star : \Box}$$

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

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

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

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

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

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

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:

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

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

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

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:

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

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)	$\dfrac{\Gamma \vdash f : (\Pi x : A.\ B) \qquad \Gamma \vdash a : A}{\Gamma \vdash f\,a : B[x := a]}$	
(Lam)	$\dfrac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (\Pi x : A.\ B) : t}{\Gamma \vdash (\lambda x : A.\ b) : (\Pi x : A.\ B)}$	$t \in \{\star, \square\}$
(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:

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

- 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 suc as follows:

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

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

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

- We can encode the type of lists of a certain type:

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

then we can define nil and cons as follows:

$$\begin{aligned}\mathsf{nil} &: \mathsf{List}\\ \mathsf{nil} &= \mathsf{fold}[\mathsf{List}]\,(\lambda(a : \star)(z : a)(f : \Pi(b : \star).\, b \to \mathsf{List} \to a).\, z)\\ \mathsf{cons} &: \Pi(b : \star).\, b \to \mathsf{List} \to \mathsf{List}\\ \mathsf{cons} &= \lambda(b : \star)(x : b)(xs : \mathsf{List}).\\ &\qquad \mathsf{fold}[\mathsf{List}]\,(\lambda(a : \star)(z : a)(f : \Pi(b : \star).\, b \to \mathsf{List} \to a).\, f\, b\, x\, xs)\end{aligned}$$

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

$$\begin{aligned}
&\mathsf{length} : \mathsf{List} \to \mathsf{Nat} \\
&\mathsf{length} = \mathsf{fix}\,(\mathsf{List} \to \mathsf{Nat})\,(\lambda(l : \mathsf{List} \to \mathsf{Nat})(xs : \mathsf{List}). \\
&\qquad\qquad (\mathsf{unfold}[\mathsf{List}]\; xs)\,\mathsf{Nat}\,\mathsf{zero}\,(\lambda(b : \star)(y : b)(ys : \mathsf{List}).\,\mathsf{suc}\,(l\; ys)))
\end{aligned}$$

- The rule (Mu) doesn't allow me to express something like $(\mu x.\, A) : \mathsf{Nat} \to \star$

3.3 Proof of soundness

Lemma 3.3.1 (λC_μ Substitutions)
Assume we have

$$\Gamma, x : A \vdash B : C \tag{1}$$

$$\Gamma \vdash D : A, \tag{2}$$

then

$$\Gamma[x := D] \vdash B[x := D] : C[x := D].$$

Proof. This is trivial by induction on the typing derivation of (1). We only discuss two cases for example. Let E^* denote $E[x := D]$. Consider following cases

- The last applied rule to obtain (1) is *Var*. There are 2 sub-cases:
  1. It is derived by
  $$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A},$$
  then we have $(B : C) \equiv (x : A)$. And $\Gamma \vdash (x : A)^* \equiv (D : A)$ which holds by (2).
  2. It is derived by
  $$\frac{\Gamma, x : A \vdash E : s}{\Gamma, x : A, y : E \vdash y : E},$$
  then we need to show $\Gamma^*, y : E^* \vdash y : E^*$. And it directly follows the induction hypothesis, i.e. $\Gamma^* \vdash E^* : s$.
- The last applied rule to obtain (1) is *App*, i.e.
$$\frac{\Gamma, x : A \vdash B_1 : (\Pi y : C_1.\, C_2) \qquad \Gamma, x : A \vdash B_2 : C_1}{\Gamma, x : A \vdash (B_1 B_2) : C_2[y := B_2]}.$$
By the induction hypothesis, we can obtain $\Gamma^* \vdash B_1^* : (\Pi y : C_1^*.C_2^*)$ and $\Gamma^* \vdash B_2^* : C_1^*$. Thus, $\Gamma^* \vdash (B_1^* B_2^*) : (C_2^*[y := B_2^*])$, i.e. $\Gamma^* \vdash (B_1 B_2)^* : (C_2[y := B_2])^*$.

□

Theorem 3.3.2 (λC_μ Subject Reduction)
If $\Gamma \vdash A : B$ *and* $A \twoheadrightarrow_\beta A'$ *then* $\Gamma \vdash A' : B$.

Proof. Let $\mathcal{D}$ be the derivation of $\Gamma \vdash A : B$. The proof is by induction on the derivation of $A \twoheadrightarrow_\beta A'$.

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

Derivation $\mathcal{D}$ has the following form

$$\dfrac{\dfrac{\Gamma, x : A \vdash M : A'}{\Gamma \vdash (\lambda x : A.M) : (\Pi x : A.A')}\,\textit{Lam} \qquad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x : A.M)N : A'}\,\textit{App}$$

Thus, by Lemma 3.3.1 we can obtain $\Gamma \vdash M[x := N] : A'$.

case *Lam*: $\dfrac{M \twoheadrightarrow_\beta M'}{\lambda x : A.M \twoheadrightarrow_\beta \lambda x : A.M'}$.

Derivation $\mathcal{D}$ has the following form

$$\dfrac{\Gamma, x : A \vdash M : A'}{\Gamma \vdash (\lambda x : A.M) : (\Pi x : A.A')}\,\textit{Lam}$$

By the induction hypothesis we have $\Gamma, x : A \vdash M' : A'$. Hence,

$$\dfrac{\Gamma, x : A \vdash M' : A'}{\Gamma \vdash (\lambda x : A.M') : (\Pi x : A.A')}\,\textit{Lam}$$

case *App (Left)*: $\dfrac{M \twoheadrightarrow_\beta M'}{MN \twoheadrightarrow_\beta M'N}$.

Derivation $\mathcal{D}$ has the following form

$$\dfrac{\Gamma \vdash M : (\Pi x : A.A') \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : A'}\,\textit{App}$$

By the induction hypothesis we have $\Gamma \vdash M' : (\Pi x : A.A')$. Hence,

$$\dfrac{\Gamma \vdash M' : (\Pi x : A.A') \qquad \Gamma \vdash N : A}{\Gamma \vdash M'N : A'}\,\textit{App}$$

case *App (Right)*: $\dfrac{M \twoheadrightarrow_\beta M'}{vM \twoheadrightarrow_\beta vM'}$.

Derivation $\mathcal{D}$ has the following form

$$\dfrac{\Gamma \vdash v : (\Pi x : A.A') \qquad \Gamma \vdash M : A}{\Gamma \vdash vM : A'}\,\textit{App}$$

By the induction hypothesis we have $\Gamma \vdash M' : A$. Hence,

$$\dfrac{\Gamma \vdash v : (\Pi x : A.A') \qquad \Gamma \vdash M' : A}{\Gamma \vdash vM' : A'}\,\textit{App}$$

□

Theorem 3.3.3 (λC_μ Progress)
If $\cdot \vdash A : B$ then either A is a value or $A \twoheadrightarrow_\beta A'$.

Proof. WIP. □

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

$$
\begin{array}{lcl}
\text{(Ax)} & \dfrac{}{\vdash s : t} & (s,t) \in \mathcal{A} \\[2ex]
\text{(Var)} & \dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} & x \notin \mathrm{dom}(\Gamma) \\[2ex]
\text{(Weak)} & \dfrac{\Gamma \vdash b : B \qquad \Gamma \vdash A : s}{\Gamma, x : A \vdash b : B} & x \notin \mathrm{dom}(\Gamma) \\[2ex]
\text{(App)} & \dfrac{\Gamma \vdash f : (\Pi x : A.\, B) \qquad \Gamma \vdash a : A}{\Gamma \vdash f a : B[x := a]} & \\[2ex]
\text{(Lam)} & \dfrac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (\Pi x : A.\, B) : t}{\Gamma \vdash (\lambda x : A.\, b) : (\Pi x : A.\, B)} & \\[2ex]
\text{(Pi)} & \dfrac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : t}{\Gamma \vdash (\Pi x : A.\, B) : u} & (s,t,u) \in \mathcal{R} \\[2ex]
\text{(Conv)} & \dfrac{\Gamma \vdash a : A \qquad \Gamma \vdash B : s \qquad A =_\beta B}{\Gamma \vdash a : B} &
\end{array}
$$

Table 3. Typing rules for a PTS