

# Formalization of Pure Type Systems

*Last modified: March 25, 2015 at 1:17am*

## 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*  $\Gamma$  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)  $\Pi$  and  $\lambda$  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  $(\Pi_-. A. B)$ .

(iv) The relation  $\rightarrow_\beta$  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  $\twoheadrightarrow_\beta$  and  $=_\beta$  by convention.

(v) Type assignment rules for  $(\mathcal{S}, \mathcal{A}, \mathcal{R})$  are given in Table 3. Particularly, the rule  $(\text{Conv})$  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* ( $\lambda C$ ), where

- (a)  $\mathcal{S} = \{\star, \square\}$
- (b)  $\mathcal{A} = \{(\star, \square)\}$
- (c)  $\mathcal{R} = \{(\star, \star), (\star, \square), (\square, \star), (\square, \square)\}$

and the typing relation is shown in Table 1.

(Ax)	$\frac{}{\vdash \star : \square}$	
(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)	$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x : A. B) : t}{\Gamma \vdash (\lambda x : A. b) : (\Pi x : A. B)}$	$t \in \{\star, \square\}$
(Pi)	$\frac{\Gamma \vdash A : s \quad \Gamma, x : A \vdash B : t}{\Gamma \vdash (\Pi x : A. B) : t}$	$(s, t) \in \mathcal{R}$
(Conv)	$\frac{\Gamma \vdash a : A \quad \Gamma \vdash B : s \quad A =_{\beta} B}{\Gamma \vdash a : B}$	

**Table 1.** Typing rules for  $\lambda C$

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

- (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:

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

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

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

### 3. Extending PTSs

#### 3.1 Recursive types

##### 3.1.1 Definition

We extend Calculus of Constructions ( $\lambda C$ , see Section 2) with recursive types, namely  $\lambda C_\mu$ . The raw expressions are extended as follows:

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

We introduce a new reduction rule for unfold and fold:

$$\text{unfold}[A] (\text{fold}[B] a) \rightarrow a$$

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

(Ax)	$\frac{}{\vdash \star : \square}$	
(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)	$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x : A. B) : t}{\Gamma \vdash (\lambda x : A. b) : (\Pi x : A. B)}$	$t \in \{\star, \square\}$
(Pi)	$\frac{\Gamma \vdash A : s \quad \Gamma, x : A \vdash B : t}{\Gamma \vdash (\Pi x : A. B) : t}$	$(s, t) \in \mathcal{R}$
(Mu)	$\frac{\Gamma, x : s \vdash A : s}{\Gamma \vdash (\mu x. A) : s}$	
(Fold)	$\frac{\Gamma \vdash a : (A[x := \mu x. A]) \quad \Gamma \vdash \mu x. A : s}{\Gamma \vdash (\text{fold}[\mu x. A] a) : \mu x. A}$	
(Unfold)	$\frac{\Gamma \vdash a : \mu x. A \quad \Gamma \vdash A[x := \mu x. A] : s}{\Gamma \vdash (\text{unfold}[\mu x. A] a) : A[x := \mu x. A]}$	
(Beta)	$\frac{\Gamma \vdash a : A \quad \Gamma \vdash B : s \quad A \rightarrow_\beta B}{\Gamma \vdash (\text{beta } a) : B}$	

**Table 2.** Typing rules for  $\lambda C_\mu$

##### 3.1.2 Examples of typable terms

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

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

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

- Using  $\text{fix}$ , we can build recursive functions. For example, given a “hungry” type  $H = \mu \sigma. \alpha \rightarrow \sigma$ , the “hungry” function  $h$  where

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

can take arbitrary number of arguments.

### 3.2 Encoding of Datatypes

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

$$\text{Nat} = \mu X. \Pi(a : \star). a \rightarrow (X \rightarrow a) \rightarrow a$$

then we can define zero and suc as follows:

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

Using  $\text{fix}$ , we can define a recursive function  $\text{plus}$  as follow:

$$\begin{aligned} \text{plus} &: \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat} \\ \text{plus} &= \text{fix} (\text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat}) (\lambda(p : \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat})(n : \text{Nat})(m : \text{Nat}). \\ &\quad (\text{unfold}[\text{Nat}] n) \text{Nat } m (\lambda(n' : \text{Nat}). \text{suc } (p n' m))) \end{aligned}$$

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

$$\text{List} = \mu X. \Pi(a : \star). a \rightarrow (\Pi(b : \star). b \rightarrow X \rightarrow a) \rightarrow a$$

then we can define  $\text{nil}$  and  $\text{cons}$  as follows:

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

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

```
length : List → Nat
length = fix (List → Nat) (λ(l : List → Nat)(xs : List).
  (unfold[List] xs) Nat zero (λ(b : ★)(y : b)(ys : List). suc (l ys)))
```

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

### 3.3 Proof of soundness

#### Lemma 3.3.1 ( $\lambda C_\mu$ Substitutions)

Assume we have

$$\Gamma, x : A \vdash B : C \quad (1)$$

$$\Gamma \vdash D : A, \quad (2)$$

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.

$$\frac{\Gamma, x : A \vdash B_1 : (\Pi y : C_1. C_2) \quad \Gamma, x : A \vdash B_2 : C_1}{\Gamma, x : A \vdash (B_1 B_2) : C_2[y := B_2]}.$$

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 ( $\lambda C_\mu$ Subject Reduction)

If  $\Gamma \vdash A : B$  and  $A \rightarrow_\beta A'$  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  $A \rightarrow_\beta A'$ .

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

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

$$\frac{\frac{\Gamma, x : A \vdash M : A'}{\Gamma \vdash (\lambda x : A.M) : (\Pi x : A.A')} \text{Lam} \quad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x : A.M)N : A'} \text{App}$$

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

**case Lam:**  $\frac{M \rightarrow_{\beta} M'}{\lambda x : A.M \rightarrow_{\beta} \lambda x : A.M'}$ .

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

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

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

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

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

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

$$\frac{\Gamma \vdash M : (\Pi x : A.A') \quad \Gamma \vdash N : A}{\Gamma \vdash MN : A'} \text{App}$$

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

$$\frac{\Gamma \vdash M' : (\Pi x : A.A') \quad \Gamma \vdash N : A}{\Gamma \vdash M'N : A'} \text{App}$$

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

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

$$\frac{\Gamma \vdash v : (\Pi x : A.A') \quad \Gamma \vdash M : A}{\Gamma \vdash vM : A'} \text{App}$$

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

$$\frac{\Gamma \vdash v : (\Pi x : A.A') \quad \Gamma \vdash M' : A}{\Gamma \vdash vM' : A'} \text{App}$$

□

### Theorem 3.3.3 ( $\lambda C_{\mu}$ Progress)

If  $\cdot \vdash A : B$  then either  $A$  is a value or  $A \rightarrow_{\beta} A'$ .

*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

(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 3.** Typing rules for a PTS