A Formalisation of the Metatheory of Pure Type Systems in Coq

Robin Adams
Department of Computer Science
University of Manchester

Pure Type Systems

- A general theory covering a large number of type systems.

- A generalization of the lambda cube.

- Invented independently by Berardi and Terlouw in 1989.

H. Barendregt, 1992. "Lambda Calculi with Types". In Abramsky, Gabbat and Maibaum (eds.): *Handbook of Logic in Computer Science*, Vol II. Oxford University Press.

Pure Type Systems

A *specification* of a Pure Type System (PTS) consists of:

- a set $\mathcal{S}$ of *sorts*;
- a set $\mathcal{A} \subseteq \mathcal{S}^2$ of *axioms*;
- a set $\mathcal{R} \subseteq \mathcal{S}^3$ of *rules*.

The set of *terms* is given by the grammar

$$\text{Term } M ::= x \mid s \mid MM \mid \Pi x : M.M \mid \lambda x : M.M$$

A *judgement* has the form

$$x_1 : A_1, ..., x_n : A_n \vdash M : B$$

The derivable judgements are given by seven rules of deduction.

Theorem
A PTS has the *Uniqueness of Types* property:

$$\Gamma \vdash M : A, \Gamma \vdash M : B \Rightarrow A \simeq_\beta B$$

iff the specification is *functional*:

$$\begin{aligned} (s,t) \in \mathcal{A}, (s,t') \in \mathcal{A} &\Rightarrow t = t' \\ (s_1,s_2,s_3) \in \mathcal{R}, (s_1,s_2,s_3') \in \mathcal{R} &\Rightarrow s_3 = s_3' \end{aligned}$$

Other Approaches to Formalization

De Bruijn Indices
Bruno Barras. *Coq in Coq*
Bruno Barras, 1999. *Auto-validation d'un système de preuves avec familles inductives*. PhD thesis, Université Paris 7

- Uses `nat` for free and bound variables.
- α-convertible terms become equal.
- Many technical lemmas about lifting and substitution operations needed.

Parameters and Bound Variables
James McKinna and Robert Pollack, 1999. *Some Lambda Calculus and Type Theory Formalized*. Journal of Automated Reasoning 23 (3–4), pp. 373–409.

- Uses separate types P and V for parameters (free variables) and bound variables.
- Very close to implementation.
- Many technicalities — such as variant induction principles for some relations — needed.

Terms as a Nested Datatype

Francoise Bellegarde and James Hook, 1994. *Substitution: A formal methods case study using monads and transformations*. Science of Computer Programming 23 (2–3), pp. 287–311.

Define the inductive family $\mathcal{T}$

$$(V : \text{Type})\ \mathcal{T}_V : \text{Type}$$

with constructors

$$\frac{x : V}{\text{var}\, x : \mathcal{T}_V} \qquad \frac{s : \mathcal{S}}{\text{srt}\, s : \mathcal{T}_V} \qquad \frac{M : \mathcal{T}_V \quad N : \mathcal{T}_V}{MN : \mathcal{T}_V}$$

$$\frac{A : \mathcal{T}_V \quad B : \mathcal{T}_{V_\perp}}{\Pi AB : \mathcal{T}_V} \qquad \frac{A : \mathcal{T}_V \quad M : \mathcal{T}_{V_\perp}}{\lambda AM : \mathcal{T}_V}$$

$\mathcal{T}_V$ is the type of terms formed using the objects of V as free variables.

```
Inductive term :  Set -> Type :=
var :  (V : Set) V -> term V |
srt :  (V : Set) sort -> term V |
app :  (V : Set) term V -> term V -> term V |
pi  :  (V : Set) term V -> term (maybe V) -> term V |
lda :  (V : Set) term V -> term (maybe V) -> term V.
```

Nat-indexed Family of Terms

Define an inductive family $\mathcal{T}$

$$(n : \mathbb{N})\ \mathcal{T}_n : \mathsf{Set}$$

with constructors

$$\frac{x : \mathcal{F}_n}{\mathsf{var}\, x : \mathcal{T}_n} \qquad \frac{s : \mathcal{S}}{\mathsf{srt}\, s : \mathcal{T}_n} \qquad \frac{M : \mathcal{T}_n \quad N : \mathcal{T}_n}{MN : \mathcal{T}_n}$$

$$\frac{A : \mathcal{T}_n \quad B : \mathcal{T}_{n+1}}{\Pi AB : \mathcal{T}_n} \qquad \frac{A : \mathcal{T}_n \quad M : \mathcal{T}_{n+1}}{\lambda AM : \mathcal{T}_n}$$

where $\mathcal{F}_n$ is a type with n objects.

$\mathcal{T}_n$ is the type of terms formed using the objects of $\mathcal{F}_n$ as free variables; i.e. the type of terms with at most n free variables.

```
Inductive term :  nat -> Set :=
var :  (n :  nat) fin n -> term n |
srt :  (n :  nat) sort -> term n |
app :  (n :  nat) term n -> term n -> term n |
pi  :  (n :  nat) term n -> term (S n) -> term n |
lda :  (n :  nat) term n -> term (S n) -> term n.
```

Substitution on the Nested Datatype

We would like to define

$$(M : \mathcal{T}_U, \rho : U \to \mathcal{T}_V)\ \ M[\rho] : \mathcal{T}_V$$

as follows:

$$\begin{array}{rcl}
(\mathsf{var}\, x)[\rho] & = & \rho x \\
(\mathsf{srt}\, s)[\rho] & = & \mathsf{srt}\, s \\
(MN)[\rho] & = & (M[\rho])(N[\rho]) \\
(\Pi AB)[\rho] & = & \Pi(A[\rho])\left(B\left[\begin{array}{ccc} \bot & \mapsto & \bot \\ \uparrow x & \mapsto & (\rho x)[\uparrow] \end{array}\right]\right) \\
(\lambda AM)[\rho] & = & \lambda(A[\rho])\left(M\left[\begin{array}{ccc} \bot & \mapsto & \bot \\ \uparrow x & \mapsto & (\rho x)[\uparrow] \end{array}\right]\right)
\end{array}$$

The recursion can be shown to terminate.

Observe:

- $\uparrow$ maps variables to variables.
- If ρ maps variables to variables, so does

$$\begin{array}{ccc} \bot & \mapsto & \bot \\ \uparrow x & \mapsto & (\rho x)[\uparrow] \end{array}$$

Replacement — Substituting Variables for Variables

Define

$$(M : \mathcal{T}_U, \rho : U \to V)\ M\{\rho\} : \mathcal{T}_V$$

the result of replacing each variable x with ρx throughout M.

The definition is by *structural recursion* on M as follows:

$$\begin{array}{rcl}
(\mathsf{var}\, x)\{\rho\} & = & \mathsf{var}(\rho x) \\
(\mathsf{srt}\, s)\{\rho\} & = & \mathsf{srt}\, s \\
(MN)\{\rho\} & = & (M\{\rho\})(N\{\rho\}) \\
(\Pi AB)\{\rho\} & = & \Pi(A\{\rho\})\left(B\left\{\begin{array}{ccc} \bot & \mapsto & \bot \\ \uparrow x & \mapsto & \uparrow(\rho x) \end{array}\right\}\right) \\
(\lambda AM)\{\rho\} & = & \lambda(A\{\rho\})\left(M\left\{\begin{array}{ccc} \bot & \mapsto & \bot \\ \uparrow x & \mapsto & \uparrow(\rho x) \end{array}\right\}\right)
\end{array}$$

Substitution Defined Using Structural Recursion

We can now define

$$(M : \mathcal{T}_U, \rho : U \to \mathcal{T}_V)\ M[\rho] : \mathcal{T}_V$$

by structural recursion as follows:

$$\begin{array}{rcl}
(\mathsf{var}\, x)[\rho] & = & \rho x \\
(\mathsf{srt}\, s)[\rho] & = & \mathsf{srt}\, s \\
(MN)[\rho] & = & (M[\rho])(N[\rho]) \\
(\Pi AB)[\rho] & = & \Pi(A[\rho])\left(B\left[\begin{array}{ccc} \bot & \mapsto & \bot \\ \uparrow x & \mapsto & (\rho x)\{\uparrow\} \end{array}\right]\right) \\
(\lambda AM)[\rho] & = & \lambda(A[\rho])\left(M\left[\begin{array}{ccc} \bot & \mapsto & \bot \\ \uparrow x & \mapsto & (\rho x)\{\uparrow\} \end{array}\right]\right)
\end{array}$$

We can proceed to define β-reduction and β-convertibility, prove the Church-Rosser Theorem, etc.

Grammar Using the Nested Datatype

We define

$$(M : \mathcal{T}_U,\ \rho : U \to V)\ M\{\rho\} : \mathcal{T}_V$$

the result of replacing x with the variable ρx throughout M;

$$(M : \mathcal{T}_U,\ \rho : U \to \mathcal{T}_V)\ M[\rho] : \mathcal{T}_V$$

the result of substituting the term ρx for x throughout M.

We prove the various forms of the Substitution Lemma:

$$\begin{aligned} M\{\rho\}\{\sigma\} &= M\{\sigma \circ \rho\} \\ M\{\rho\}[\sigma] &= M[\sigma \circ \rho] \\ M[\rho]\{\sigma\} &= M[x \mapsto (\rho x)\{\sigma\}] \\ M[\rho][\sigma] &= M[x \mapsto (\rho x)[\sigma]] \end{aligned}$$

Richard S. Bird and Ross Paterson, 1999. *De Bruijn Notation as a Nested Datatype*. J. Functional Programming 9 (1): 77–91.

Thorsten Altenkirch and Bernhard Reus, 1999. *Monadic Presentations of Lambda Terms using Generalized Inductive Types*. CSL '99, LNCS 1683, pp. 453–468.

Rules of Deduction of a PTS

Define the type $\mathcal{C}_n$ of *contexts of length* n:

$$\begin{aligned} \mathcal{C}_0 &= \{*\} \\ \mathcal{C}_{n+1} &= \mathcal{C}_n \times \mathcal{T}_n \end{aligned}$$

Now we define the relation `Inductive PTS : (n : nat) ctxt n -> term n -> term n -> Prop :=` by simply writing down the rules of deduction of a PTS:

$$\text{(axioms)}\ \frac{}{\vdash s : t}\ (s : t \in \mathcal{A})$$

```
axioms : (s t : sort) ax s t ->
PTS 0 tt (srt s) (srt t)
```

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

```
weak : (n : nat) (Gamma : ctxt n) (A : term n)
(s : sort)
PTS n Gamma A (srt s) ->
PTS (S n) (Gamma, A) (var bot) (lift A)
```

etc.

"Big-Step" Weakening and Substitution Lemmas

For $\Gamma : \mathcal{C}_m$, $\Delta : \mathcal{C}_n$, define $\Gamma \subseteq_\rho \Delta$ to be

$$(\forall x : \mathcal{F}_m)\Delta_{\rho x} \equiv \Gamma_x\{\rho\}$$

where Γ_x is the type assigned to x by the context Γ.

Weakening Lemma If $\Gamma \subseteq_\rho \Delta$, $\Gamma \vdash M : A$, and Δ is a valid context, then $\Delta \vdash M\{\rho\} : A\{\rho\}$.

Substitution Lemma Let $\Gamma : \mathcal{C}_m$, $\rho : \mathcal{F}_m \to \mathcal{T}_n$, $\Delta : \mathcal{C}_n$. If $\Gamma \vdash M : A$ and

$$(\forall x : \mathcal{F}_m)\Delta \vdash \rho x : \Gamma_x[\rho]$$

then

$$\Delta \vdash M[\rho] : A[\rho]\,.$$

We proceed to prove Generation Lemmas, Subject Reduction, and Uniqueness of Types for functional PTSs.

Conclusion

Defining terms as a nested datatype is useful when formalizing the metatheory of a type theory:

- α-convertible terms become equal.
- "Big-step" definitions can be made, and "big-step" theorems proved.
- Technicalities are few, and have a type-theoretic flavour.

Coq scripts available at:
`http://www.cs.man.ac.uk/~radams/coqPTS`