Pure Type Systems in Rewriting Logic

Specifying Typed Higher-Order Languages in a First-Order Logical Framework

Presenter: Baqiao Liu

Stehr, Mark-Oliver, and José Meseguer. "Pure type systems in rewriting logic: Specifying typed higher-order languages in a first-order logical framework."

All images come from the original paper

Outline

- Overview
- Background
- Main Contents
- Discussion
  - My Project
- Conclusion

Overview

- Previously in class...
  - Lambda calculus (λ) semantics in rewriting logic
  - Lecture 14b. direct implementation of λ
  - Lecture 15. λ with de Bruijn indices → RWL.
  - Lecture 16. λ-CINNI
    - Machine friendly and user friendly!

Overview cont.

- Useful typed versions of λ
  - Simply typed λ-calculus, System F, Calculus of Constructions, etc.
  - Lambda Cube (Barendregt 89), Pure Type Systems (Bardendregt 90)
- PTS in rewriting logic (RWL)
  - Evolution of PTS all formalized in Maude
  - The formalization is "nice"

PTS → UPTS → □ → □

Background

STLC → System F.

"type"

- Typed Lambda Calculi $I := \underline{\lambda x.\, x}$, $B : Type$, $\to I : B \to B$. $A : Type$, $\to I : A \to A$. $I : \forall \alpha : Type,\ \alpha \to \alpha$.
  - Going Up the Cube

$$\begin{array}{lll} \textit{terms} & M, N & ::= \; x \mid \lambda x : T.M \mid M\ N \\ \textit{types} & T, U & ::= \; X \mid T \to U \end{array}$$

$$\begin{array}{lll} \textit{terms} & M, N & ::= \; x \mid \lambda x : T.M \mid M\ N \mid \lambda X : \texttt{Prop}.M \mid M\ T \\ \textit{types} & T, U & ::= \; X \mid T \to U \mid \forall X : \texttt{Prop}.T \end{array}$$

  - Generalize notion of "dependencies"
    - → Terms dependent on terms $\underbrace{(\lambda x.\, x)\,(v)}$, STLC.
    - → Types dependent on types Generics Array<Integer>
    - → Terms dependent on types
    - → Types dependent on terms Vector n

      $n : \mathbb{N}$

Coq CoC

Background cont.

- Pure Type Systems {Prop, Type}
  - A PTS is parameterized by a triple (S, A, R)
    - S : a set of *sorts*, all constants, also called *universes* Prop : Type.
    - A : a subset of S^2 called *axioms* , denoted as c : s Prop, Type, Type
    - R : a subset of S^3 called *rules* "dependencies".
  - Terms are X | (M N) | [X : A] M | {X : A} M | s
    - range over sorts
    - range over terms
    - names

$\lambda x : A . M \mid \Pi_{x:A} M . \mid s,$

  - Equal modulo alpha equivalence

Background cont.

- Natural deduction rules

$$\frac{}{[] \vdash s_1 : s_2} \quad (s_1, s_2) \in \mathcal{A} \qquad \frac{}{0 : \mathbb{N}} \qquad \frac{n : \mathbb{N}}{succ(n) : \mathbb{N}} \qquad \text{(Axiom)}$$

$$\frac{\Gamma \vdash A : s}{\Gamma, X : A \vdash X : A} \quad X \notin \Gamma \qquad \text{(Start)}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma, X : B \vdash M : A} \quad X \notin \Gamma \qquad \text{(Weakening)}$$

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, X : A \vdash B : s_2}{\Gamma \vdash \{X : A\}B : s_3} \quad (s_1, s_2, s_3) \in \mathcal{R} \qquad \text{(Pi)}$$

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, X : A \vdash M : B \quad \Gamma, X : A \vdash B : s_2}{\Gamma \vdash [X : A]M : \{X : A\}B} \quad (s_1, s_2, s_3) \in \mathcal{R} \qquad \text{(Lda)}$$

λ-abstraction.

$$\frac{\Gamma \vdash M : \{X : A\}B \quad \Gamma \vdash N : A}{\Gamma \vdash (M\ N) : [X{:=}A]B} \qquad \text{(App)}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma \vdash M : B} \quad A \equiv_\beta B \qquad \text{(Conv)}$$

Background cont.

- Generality

  $T : \mathcal{U}$.

  - if we instantiate S = {T, U}, and A = {(T, U)} with R = {(T,T,T)} we recover STLC
  - If R = {(T,T,T),(U,U,U),(T,U,U),(U,T,T)} we recover the calculus of constructions

Background cont.

- We can now naively implement PTS semantics in Maude
- What about alpha equivalence?
- Representational distance
  - If we represent "faithfully" one logical system L in G by some correspondence: informally, how complex is this correspondence?

Main Contents

Roadmap

- PTS
- PTS-CINNI?

Go deeper

- PTS
- UPTS (basically PTS-CINNI)
- OUPTS (explicit judgements for well-typed contexts)
- ROUPTS (type-checking of OUPTS formalized in RWL)

PTS in RWL

Do the naïve implementation in Maude

- Family of PTS parameterized by their signature
  - Represented as a parameterized theory in Maude
- Terms
  - Expressed as ADTs in Maude
    - Minor point: distinguishes between names being declared and names being used
- Substitution
  - Sets, set inclusion
- Beta-convertability (used in the Conv rule)
- Context, Judgements, Inference Rules

Actual Implementation Code

Signatures

```
fth PTS-SIG is
sorts Sorts Axioms Axioms? Rules Rules? .
subsort Axioms < Axioms? .
subsort Rules < Rules? .
op (_,_) : Sorts Sorts -> Axioms? .
op (_,_,_) : Sorts Sorts Sorts -> Rules? .
endfth
```

Actual Implementation Code

The parameterized theory, and the terms as algebraic data types

```
fmod PTS[S :: PTS-SIG] is
   sorts Var Trm .
   subsort Qid < Var .
   subsort Var < Trm .
   subsort Sorts < Trm .
   op __ : Trm Trm -> Trm .
   op [_:_]_ : Qid Trm Trm -> Trm .
   op {_:_}_ : Qid Trm Trm -> Trm .

   vars s s1 s2 s3 : Sorts .
   vars X Y Z : Qid .
   vars A B M N O P Q R T A' B' M' N' T' : Trm .
```

Actual Implementation Code

Set inclusion, free variables (implementation not explicitly given). Substitution operator

```
op _in_ : Qid QidSet -> Bool .
op FV : Trm -> QidSet .
op [_:=_]_ : Qid Trm Trm -> Trm .
```

```
ceq [X : A] M = [Y : A] ([X := Y] M) if not(Y in FV(M)) .
ceq {X : A} M = {Y : A} ([X := Y] M) if not(Y in FV(M)) .
```

Actual Implementation Code

Beta convertibility (a and b are convertible if they are beta-equivalent)

```
sorts Convertible Convertible? .
subsort Convertible < Convertible? .

op _<->_ : Trm Trm -> Convertible? .

mb  M <-> M : Convertible .

cmb M <-> N : Convertible
    if N <-> M : Convertible .

cmb P <-> R : Convertible if
    P <-> Q : Convertible and Q <-> R : Convertible .

cmb (M N) <-> (M' N') : Convertible if
    M <-> M' : Convertible and N <-> N' : Convertible .

cmb ([X : A] M) <-> ([X : A'] M') : Convertible if
    A <-> A' : Convertible and M <-> M' : Convertible .

cmb ({X : A} B) <-> ({X : A'} B') : Convertible if
    A <-> A' : Convertible and B <-> B' : Convertible .

mb  (([X : A] M) N) <-> ([X := N] M) : Convertible .
```

Actual Implementation Code

Context, Judgements

```
sorts Context Judgement .
op emptyContext : -> Context .
op _:_ : Qid Trm -> Context .
op _,_ : Context Context -> Context [assoc id : emptyContext] .

var G : Context .
```



```
op _|-_:_ : Context Trm Trm -> Judgement .

op _in_ : Qid Context -> Bool .
eq X in emptyContext = false .
eq X in (G,(Y : A)) = (X in G) or (X == Y) .
```

Actual Implementation Code

Inference Rules

```
sort Derivable .
subsort Derivable < Judgement .

cmb (emptyContext |- s1 : s2) : Derivable if (s1,s2) : Axioms .

cmb (G,(X : A) |- X : A) : Derivable if
    (G |- A : s) : Derivable /\ not(X in G) .

cmb (G,(X : B) |- M : A) : Derivable if
    (G |- M : A) : Derivable /\
    (G |- B : s) : Derivable /\ not(X in G) .

cmb (G |- {X : A} B : s3) : Derivable if
    (G |- A : s1) : Derivable /\
    (G,(X : A) |- B : s2) : Derivable /\ (s1,s2,s3) : Rules .

cmb (G |- [X : A] M : {X : A} B) : Derivable if
    (G |- A : s1) : Derivable /\
    (G,(X : A) |- M : B) : Derivable /\
    (G,(X : A) |- B : s2) : Derivable /\ (s1,s2,s3) : Rules .

cmb (G |- (M N) : [X := A] B) : Derivable if
    (G |- M : {X : A} B) : Derivable /\
    (G |- N : A) : Derivable .

cmb (G |- M : B) : Derivable if
    (G |- M : A) : Derivable /\
    (G |- B : s) : Derivable /\ A <-> B : Convertible .

endfm
```

PTS: Takeaway

- RWL is powerful enough to make a straightforward representation of PTSs
  - without much change in syntax or rules
- One problem: alpha equivalence
  - Use CINNI?

$$\forall X \,.\, (A \;\wedge\; \forall Y \,.\, (B \Rightarrow \forall X.C(X_{1}^{2})))$$

UPTS

- PTS with CINNI, which has the advantages as listed in class:
  - Machine friendly
  - Canonical representation of equivalent terms
  - Avoiding the variable capture problem
- $\uparrow$ is the operation "shift" as defined in class
- $\Uparrow$ is the operation "lift" as defined in class

UPTS

- Terms $X_m \mid (M\ N) \mid [X : A]M \mid \{X : A\}M \mid s$

- New Substitution equations

$$
\begin{aligned}
S\ s &= s \\
S\ (MN) &= (SM)(SN) \\
S\ ([X : A]M) &= [X : (S\ A)](\Uparrow_X S\ M) \\
S\ (\{X : A\}M) &= \{X : (S\ A)\}(\Uparrow_X S\ M)
\end{aligned}
$$

- Change inference rules

$$\frac{\Gamma \vdash A : s}{\Gamma, X : A \vdash X_0\ : \uparrow_X A} \quad \text{(Start)}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma, X : B \vdash \uparrow_X M\ : \uparrow_X A} \quad \text{(Weak)}$$

UPTS

- Now we have explicit alpha equivalence

```
mb  [X : A] M <->
    [Y : A] ([X := Y{0}] [shift Y] M) : Convertible .

mb  {X : A} M <->
    {Y : A} ([X := Y{0}] [shift Y] M) : Convertible .
```

UPTS: Takeaway

Representational distance small. The equational nature is captured by the Maude specification.

We now have the advantages brought by CINNI.

OUPTS

Explicitly encodes the notion of a context being well typed.

$\vdash \Gamma$ $\qquad$ $\Gamma \vdash x : A$

New judgements: a context is well typed. Relative typing (M : A for some context if the context is well typed)

$\Gamma \vdash x : A$

$\Gamma \Vdash x : A$

OUPTS

New rules:

$$\frac{}{[] \Vdash} \quad \text{(Ctxt1)}$$

$$\frac{\Gamma \Vdash \quad \Gamma \vdash A : s}{\Gamma, X : A \Vdash} \quad \text{(Ctxt2)}$$

$$\frac{}{\Gamma \vdash X_m : \mathit{lookup}(\Gamma, X_m)} \text{ if } \mathit{lookup}(\Gamma, X_m) \text{ is defined} \quad \text{(Lookup)}$$

$$\frac{\Gamma \Vdash \quad \Gamma \vdash M : A}{\Gamma \Vdash M : A} \quad \text{(Main)}$$

$\Gamma \Vdash$

$\Gamma \Vdash M : A$

where $\mathit{lookup}(\Gamma, X_m)$ is a partial function defined by:

$$\begin{aligned}
\mathit{lookup}((\Gamma, X : A), X_0) &= \uparrow_X A \\
\mathit{lookup}((\Gamma, X : A), X_{m+1}) &= \uparrow_X \mathit{lookup}(\Gamma, X_m) \\
\mathit{lookup}((\Gamma, X : A), Y_m) &= \uparrow_X \mathit{lookup}(\Gamma, Y_m) \text{ if } X \neq Y
\end{aligned}$$

OUPTS

New versions of existing rules:

$$\frac{}{\Gamma \vdash s_1 : s_2} \quad (s_1, s_2) \in \mathcal{A} \qquad \text{(Ax)}$$

$$\frac{\Gamma \Vdash M : A}{\Gamma \Vdash_1 M : A} \quad \text{if no variable is declared in } \Gamma \text{ more than once.} \qquad \text{(Simple)}$$

The implementation still has a very low representational distance.

Discussions

- Even when generalized from the lambda calculus, representational distance from the implementation to the original formal system very small
- Generality of RWL: RWL is at least as general as the calculus of constructions. No "good" mappings from linear logic to system LF exists, but linear logic can be mapped to RWL quite well. Therefore RWL is more general than LF.

My Project

- Replicate this paper on Calculus of Inductive Constructions (CIC)
  - CIC is basically CoC with inductive types
  - CIC is the core calculus used in the Coq proof assistant
  - Goal: CIC→UCIC→OUCIC→ROUCIC
- CIC (compared to CoC) makes it easy to introduce new inference rules via inductive types
  - Can have natural numbers, ADTs defined organically
- Not a PTS; therefore I need to extend this paper's approach

Conclusion

Higher order logic with substitution, type-checking can be represented with low representational distance in RWL. This also has computational advantages.