Toward an Adequation Lemma for PML2

LAMA
Laboratoire de Mathématiques

Rodolphe Lepigre - Montevideo - 03/12/2014

Why another proof assistant?

Proof assistants usually come with two languages:

- Formulas (e.g. specifications)
- Proof-terms (e.g. pure λ-calculus)
- An optional proof construction language (e.g. tactics)

Our aim: build a programing language centered system

What about other systems?

- *Coq*: hidden proof-terms (use of tactics)
- *Agda*: proof-terms with a limited syntax (explicited directly)
- *HOL light, HOL, Isabelle/HOL*: no proof-terms

The ingredients

Programming side:

- Full-featured ML-like language
- Evaluation strategy: call-by-value
- Curry-style language (no types in terms)
- Proofs are programs

Logic side:

- Higher-order types
- Classical logic
- Program values are the individuals of the logic
- Contain the equational theory of the programming language

Example using the equational theory

```
type rec nat = [ Z[] | S[nat] ]

val rec (+) : nat => nat => nat =
  fun m n -> match n with
             | Z[]   -> m
             | S[n'] -> S[m + n']

val rec assoc : l:nat => m:nat => n:nat => (l+m)+n == l+(m+n) =
  fun l m n -> match n with
              | Z[]   -> show (l+m)+Z[] == l+(m+Z[]);
                         show l+m == l+m;
                         8<
              | S[n'] -> show (l+m)+S[n'] == l+(m+S[n']);
                         show S[(l+m)+n'] == l+S[m+n'];
                         show S[(l+m)+n'] == S[l+(m+n')];
                         show (l+m)+n' == l+(m+n');
                         use (assoc l m n'); 8<
```

Every “`show ... == ...;`” is only added for clarity

Values and terms

Call-by-value λ-calculus has two syntactic entities:

$$v, w ::= x \mid \lambda x\, t$$

$$t, u ::= v \mid t\, u$$

Remarks:

- Values are terms
- In call-by-name values and terms are collapsed

Why do we want a call-by-value language?

- Quantifiers are more symmetric
- Works well in practice (*OCaml*)
- Simon Peyton Jones regrets not using call-by-value for *Haskell*

Going ML-like

We add case analysis, records and a fixpoint operator:

$$v, w \ ::= \ \cdots \mid C[v] \mid \{ \cdots \ l_i = v_i \, ; \ \cdots \}$$

$$t, u \ ::= \ \cdots \mid Y(t, v) \mid v \, . \, l \mid \mathsf{case}\ v\ \mathsf{of}\ [\ \cdots \ C_i[x] \rightarrow t_i \, ; \ \cdots \ ]$$

We enforce values in many places to simplify the calculus

We can define syntactic sugars:

$$C[t] ::= (\lambda x\, C[x])\, t \qquad\qquad t\, .\, l ::= (\lambda x\, x\, .\, l)\, t$$

Let's make the calculus classical

One possibility is to add a μ binder ($\lambda\mu$-calculus):

$$t, u ::= \cdots \mid \mu\alpha\, t \mid t * \pi$$

$$\pi, \rho ::= \alpha \mid v \cdot \pi \mid [t]\,\pi$$

Stacks can be manipulated as first-class objects

Remarks:

- A stack can be seen as an evaluation context
- Intuition: it stores function arguments
- In call-by-value we need stack-frames ($[t]\,\pi$)

Summary of the syntax: Values, Terms, Stacks and Processes

$$v, w \ ::= \ x \mid \lambda x\, t \mid C[v] \mid \{ \cdots \ l_i = v_i \,; \ \cdots \} \qquad (\Lambda_v)$$

$$t, u \ ::= \ v \mid t\, u \mid \mu\alpha\, t \mid p \mid Y(t, v) \mid v.l \mid \text{case}\, v\, \text{of}\, [\, \cdots \,] \qquad (\Lambda)$$

$$\pi, \rho \ ::= \ \alpha \mid v \cdot \pi \mid [t]\, \pi \qquad (\Pi)$$

$$p, s \ ::= \ t * \pi \qquad (\Lambda * \Pi)$$

A process forms the internal state of a *Krivine Machine*

It can be thought of as a term in its environment

Operational semantics - reduction relation

Call-by-value β-reduction:

$$\begin{aligned}(t\,u)*\pi &\twoheadrightarrow u*[t]\pi\\ v*[t]\pi &\twoheadrightarrow t*v\cdot\pi\\ (\lambda x\,t)*v\cdot\pi &\twoheadrightarrow t[x \leftarrow v]*\pi\end{aligned}$$

Capturing and restoring the evaluation context:

$$\begin{aligned}(\mu\alpha\,t)*\pi &\twoheadrightarrow t[\alpha \leftarrow \pi]*\pi\\ p*\pi &\twoheadrightarrow p\end{aligned}$$

There are also rules for projection, case analysis and the fixpoint operator

Equivalence relation

Given a process p we write:

- $p \Downarrow$ if $\exists \nu, \exists \alpha, p \twoheadrightarrow^* \nu * \alpha$
- $p \Uparrow$ otherwise

Intuitively $p \Downarrow$ means that the evaluation of p is successful

We write $t \equiv u$ if $\forall \pi, t * \pi \Downarrow \Leftrightarrow u * \pi \Downarrow$

$\equiv$ is an equivalence relation over terms

Type system

We start from *System F*:

$$\begin{array}{rcl} A\,,B & ::= & X \\ & | & A \Rightarrow B \\ & | & \forall X\ A \\ & | & \exists X\ A \end{array}$$

We extend it to an ML-like system:

$$\begin{array}{rcl} A\,,B & ::= & \dots \\ & | & [\ \dots\ C_i[A_i]\,;\ \dots\ ] \\ & | & \{\ \dots\ l_i : A_i\,;\ \dots\ \} \\ & | & \mu X_n A \end{array}$$

Allowing formulas to talk about terms

We add four type constructors:

- $t \in A$ meaning "t is a term of type A"
- $A \upharpoonright t \equiv u$ meaning "A and $t \equiv u$"
- $\forall x\ A$ and $\exists x\ A$ quantifying over values

We also add n-ary predicates over terms:

$$
\begin{aligned}
A, B \quad ::= &\quad \cdots \\
\mid &\quad X_n(t_1, \ldots, t_n) \\
\mid &\quad \forall X_n\ A \\
\mid &\quad \exists X_n\ A
\end{aligned}
$$

The variables of *System F* can be seen as predicates of arity 0

Full second-order type system

$$
\begin{array}{rcl}
A, B & ::= & X_n(t_1, \dots, t_n) \\
& | & A \Rightarrow B \\
& | & \forall X_n\ A \quad | \quad \exists X_n\ A \\
& | & [\ \dots\ C_i[A_i];\ \dots\ ] \\
& | & \{\ \dots\ l_i : A_i;\ \dots\ \} \\
& | & \mu X_n A \\
& | & \forall x\ A \quad | \quad \exists x\ A \\
& | & t \in A \\
& | & A \upharpoonright t \equiv u
\end{array}
$$

It is possible to extend this type system to higher-order

Semantics

We interpret terms and values as their equivalence classes

- $[\![v]\!] = \{w \in \Lambda_v \mid v \equiv w\}$
- $[\![t]\!] = \{u \in \Lambda \mid t \equiv u\}$

Raw semantics of formulas:

- $[\![A \Rightarrow B]\!] = \{\lambda x\, t \mid \forall\, v \in [\![A]\!], t[x \leftarrow v] \in [\![B]\!]^{\perp\perp}\}$
- $[\![\forall X_n\ A]\!] = \cap_{P_n} [\![A[X_n \leftarrow P_n]]\!]$
- $[\![\forall x\ A]\!] = \cap_{v \in \Lambda_v} [\![A[x \leftarrow v]]\!]$
- $[\![t \in A]\!] = \{v \in [\![A]\!] \mid v \equiv t\}$
- $[\![A \upharpoonright t \equiv u]\!] = [\![A]\!]$ if $t \equiv u$ and $\emptyset$ otherwise
- ...

The set $[\![A]\!]$ is closed under $\equiv$ for all A (by construction)

Pole, Falsity Values and Truth Values

We define a family of poles $\perp\!\!\!\perp_{(V_i,\alpha_i)_{i\in I}}$:

$$\perp\!\!\!\perp_{(V_i,\alpha_i)_{i\in I}} = \{p \mid \exists\, i \in I, \exists\, v \in V_i, \exists\, w \equiv v, p \twoheadrightarrow^* w * \alpha_i\}$$

Properties of a pole $\perp\!\!\!\perp$:

- They are closed under $(\twoheadrightarrow)^{-1}$
- And closed under $(\twoheadrightarrow)$
- If $v * \alpha \in \perp\!\!\!\perp$ and $v \equiv w$ then $w * \alpha \in \perp\!\!\!\perp$

For every formula A we define:

$$\llbracket A \rrbracket^{\perp} = \{\pi \in \Pi \mid \forall\, v \in \llbracket A \rrbracket, v * \pi \in \perp\!\!\!\perp\}$$

$$\llbracket A \rrbracket^{\perp\perp} = \{t \in \Lambda \mid \forall\, \pi \in \llbracket A \rrbracket^{\perp}, t * \pi \in \perp\!\!\!\perp\}$$

Typing judgements and Adequation Lemma

We have two forms of typing judgements (collapsed in call-by-name):

$$\Gamma \vdash v : A \qquad\qquad \Gamma \vdash t : A$$

A context Γ contain:

- Type assignments of the form $x : A$
- Type assignments of the form $\alpha : A^{\perp}$
- Equivalences / inequivalences of the form $t \equiv u$ / $t \not\equiv u$

Theorem 1.

$$\Gamma \vdash v : A \Rightarrow v' \in [\![A]\!] \qquad\qquad \Gamma \vdash t : A \Rightarrow t' \in [\![A]\!]^{\perp\perp}$$

Adding adequate typing rules to the system

We can add any rule provided that it is adequate

Examples of adequate rules:

$$\frac{}{\Gamma, x : A \vdash x : A}\text{Ax} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x\, t : A \Rightarrow B}\Rightarrow_i \qquad \frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}\Rightarrow_e$$

$$\frac{\Gamma, \alpha : A^\perp \vdash t : A}{\Gamma \vdash \mu\alpha\, t : A}\mu \qquad \frac{\Gamma, \alpha : A^\perp \vdash t : A}{\Gamma, \alpha : A^\perp \vdash t * \alpha : B}*$$

Proof of adequacy of $(\Rightarrow_e)$

We suppose $t' \in [\![A \Rightarrow B]\!]$ and $u' \in [\![B]\!]$

We need to show $(t'u') \in [\![B]\!]^{\perp\perp}$

We take $\pi \in [\![B]\!]^{\perp}$ and show $(t'u') * \pi \in \perp\!\!\!\perp$

It is enough to show $u' * [t']\pi \in \perp\!\!\!\perp$

It is enough to show $[t']\pi \in [\![B]\!]^{\perp}$

We take $v \in [\![B]\!]$ and show $v * [t']\pi \in \perp\!\!\!\perp$

It is enough to show $t' * v . \pi \in \perp\!\!\!\perp$

It is enough to show $v . \pi \in [\![A \Rightarrow B]\!]^{\perp}$

We take $\lambda x\, m \in [\![A \Rightarrow B]\!]$ and show $\lambda x\, m * v . \pi \in \perp\!\!\!\perp$

It is enough to show $m[x \leftarrow v] * \pi \in \perp\!\!\!\perp$

It is enough to show $m[x \leftarrow v] \in [\![B]\!]^{\perp\perp}$

This is true by definition of $[\![A \Rightarrow B]\!]$

Rules of *System F*

$$\frac{\Gamma \vdash v : A}{\Gamma \vdash v : \forall X\ A}\ \forall_i \qquad \frac{\Gamma \vdash t : \forall X_n\ A}{\Gamma \vdash t : A[X_n \leftarrow P_n]}\ \forall_e$$

$$\frac{\Gamma \vdash t : A[X_n \leftarrow P_n]}{\Gamma \vdash t : \exists X_n\ A}\ \exists_i \qquad \frac{\Gamma, x : A[X_n \leftarrow P_n] \vdash t : B}{\Gamma, x : \exists X_n\ A \vdash t : B}\ \exists_e$$

Records and case analysis

$$\frac{\Gamma \vdash v : \{\ \cdots\ \ l_i : A_i\,;\ \ \cdots\ \}}{\Gamma \vdash v\,.\,l_i : A_i}\times_e \qquad \frac{\cdots \quad \Gamma \vdash v_i : A_i \ \ \cdots}{\Gamma \vdash \{\ \cdots\ \ l_i = v_i\,;\ \ \cdots\ \} : \{\ \cdots\ \ l_i : A_i\,;\ \ \cdots\ \}}\times_i$$

$$\frac{\Gamma \vdash v : A_i}{\Gamma \vdash C_i[v] : [\ \cdots\ \ C_i[A_i];\ \ \cdots\ ]}+_i$$

$$\frac{\Gamma \vdash v : [\ \cdots\ \ C_i[A_i];\ \ \cdots\ ] \qquad \cdots \quad \Gamma, x : A_i\,, C_i[x] \equiv v \vdash t_i : B \quad \cdots}{\Gamma \vdash \mathsf{case}\ v\ \mathsf{of}\ [\ \cdots\ \ C_i[x] \rightarrow t_i\,;\ \ \cdots\ ] : B}+_e$$

Remark: equivalence in the premise of $+_e$

Quantification over individuals

$$\frac{\Gamma \vdash v : A}{\Gamma \vdash v : \forall x\ A}\forall_i \qquad \frac{\Gamma \vdash t : \forall x\ A}{\Gamma \vdash t : A[x \leftarrow v]}\forall_e$$

$$\frac{\Gamma \vdash t : A[x \leftarrow v]}{\Gamma \vdash t : \exists x\ A}\exists_i \qquad \frac{\Gamma, x : A[y \leftarrow v] \vdash t : B}{\Gamma, x : \exists y\ A \vdash t : B}\exists_e$$

Belonging and Restriction

$$\frac{\Gamma \vdash v : A \qquad \Gamma \vdash t \equiv v}{\Gamma \vdash v : t \in A}\in_i \qquad \frac{\Gamma, x : A, x \equiv u \vdash t : B}{\Gamma, x : u \in A \vdash t : B}\in$$

$$\frac{\Gamma, x : A, u_1 \equiv u_2 \vdash t : C}{\Gamma, x : A \upharpoonright u_1 \equiv u_2 \vdash t : C}\upharpoonright_l \qquad \frac{\vdash \mathcal{E}(\Gamma, u_1 \not\equiv u_2) \qquad \Gamma, u_1 \equiv u_2 \vdash t : A}{\Gamma \vdash t : A \upharpoonright u_1 \equiv u_2}\upharpoonright_r$$

Dependent product

The usual dependent product $\Pi x : A\, B$ can be encoded:

$$\Pi x : A\, B \quad := \quad \forall x\ (x \in A \Rightarrow B)$$

For instance the elimination rule

$$\frac{\Gamma \vdash t : \Pi_{x:A} B \qquad \Gamma \vdash v : A}{\Gamma \vdash t\, v : B[x \leftarrow v]}\,\Pi_e$$

can be derived:

$$\dfrac{\dfrac{\Gamma \vdash t : \forall x\ (x \in A \Rightarrow B)}{\Gamma \vdash t : v \in A \Rightarrow B[x \leftarrow v]}\,\forall_e \qquad \dfrac{\Gamma \vdash v \in A}{\Gamma \vdash v : v \in A}\,\in_i}{\Gamma \vdash (t\, v) : B[x \leftarrow v]}\,\Rightarrow_e$$

Value restriction

In call-by-value with classical logic we need value restriction:

$$\frac{\Gamma \vdash t : \Pi_{x:A} B \qquad \Gamma \vdash v : A}{\Gamma \vdash t\,v : B[x \leftarrow v]}\,\Pi_e$$

The following rule is not valid:

$$\frac{\Gamma \vdash t : \Pi_{x:A} B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B[x \leftarrow u]}\,\Pi_e$$

We would like to have at least:

$$\frac{\Gamma, y \equiv u \vdash t : \Pi_{x:A} B \qquad \Gamma, y \equiv u \vdash u : A}{\Gamma, y \equiv u \vdash t\,u : B[x \leftarrow u]}\,\Pi_e$$

Derivation of Π_e

Provided that we have:

$$\dfrac{\Gamma, t_1 \equiv t_2 \vdash u : A[t_1]}{\Gamma, t_1 \equiv t_2 \vdash u : A[t_2]}\equiv_r \qquad \dfrac{\Gamma, t_1 \equiv t_2 \vdash t_1 : A}{\Gamma, t_1 \equiv t_2 \vdash t_2 : A}\equiv_l$$

We can derive the rule Π_e on t using $x \equiv t$:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\mathcal{P}_1\\ \vdots\\ \Gamma, y \equiv u \vdash t : \Pi_{x:A} B\end{array} \qquad \dfrac{\begin{array}{c}\mathcal{P}_2\\ \vdots\\ \Gamma, y \equiv u \vdash u : A\end{array}}{\Gamma, y \equiv u \vdash y : A}\equiv_l}{\Gamma, y \equiv u \vdash t\,y : B[x \leftarrow y]}\Pi_e^v}{\Gamma, y \equiv u \vdash t\,u : B[x \leftarrow y]}\equiv_l}{\Gamma, y \equiv u \vdash t\,u : B[x \leftarrow u]}\equiv_r$$

Required property of the model

We need $\equiv$ to be extensional:

- $v \equiv w \Rightarrow E[x \leftarrow v] \equiv E[x \leftarrow w]$
- $t \equiv u \Rightarrow E[t] \equiv E[u]$

We also need:

Theorem 2.

If $\Phi \subseteq \Lambda_v$ is closed under $(\equiv)$ then $\Phi = \Phi^{\perp\perp} \cap \Lambda_v$

Direct consequence: $v \in [\![A]\!]^{\perp\perp} \Rightarrow v \in [\![A]\!]$

Remarks:

- $\Phi \subseteq \Phi^{\perp\perp} \cap \Lambda_v$ is trivial
- $\Phi \supseteq \Phi^{\perp\perp} \cap \Lambda_v$ is not true in general...

Main idea (sufficient condition)

We add a new term (or instruction) to the syntax:

$$t, u \quad ::= \quad \cdots \mid \delta(v, w)$$

With the reduction rule:

$$\delta(v, w) * \pi \twoheadrightarrow v * \pi \qquad \text{if} \qquad v \not\equiv w$$

In the presence of $\delta(v, w)$ we will obtain

$$\Phi \supseteq \Phi^{\perp\perp} \cap \Lambda_v$$

Proof

Recall the definitions:

$$\Phi^{\perp} = \{\pi \in \Pi \mid \forall\, v \in \Phi,\, v * \pi \in \perp\!\!\!\perp\} \qquad \Phi^{\perp\perp} = \left\{t \in \Lambda \mid \forall\, \pi \in \Phi^{\perp},\, t * \pi \in \perp\!\!\!\perp\right\}$$

We consider $\Phi \subseteq \Lambda_v$ closed under $(\equiv)$ and show $\Phi^{\perp\perp} \cap \Lambda_v \subseteq \Phi$

We assume that $v \notin \Phi$ and show that $v \notin \Phi^{\perp\perp}$

We need to find a stack $\pi_0 \in \Phi^{\perp}$ such that $v * \pi_0 \notin \perp\!\!\!\perp$.

We need to find a stack $\pi_0 \in \Pi$ such that:

- $\forall w \in \Phi,\, w * \pi_0 \in \perp\!\!\!\perp$
- $v * \pi_0 \notin \perp\!\!\!\perp$

$\pi_0 = [\lambda x\, \delta(x, v)]\, \alpha$ is such a stack

A stratified model

Problem: $(\twoheadrightarrow)$ and $(\equiv)$ are interdependent...

For all $i \in \mathbb{N}$ we define:

$$\begin{aligned}
(\twoheadrightarrow_0) &= (\succ) \\
(\twoheadrightarrow_{i+1}) &= (\twoheadrightarrow_i) \cup \left\{(\delta(v, w) * \pi, v * \pi) \mid v \not\equiv_i w\right\} \\
(\equiv_i) &= \left\{(t, u) \mid \forall j \leqslant i, \forall \pi \in \Pi, \forall \sigma, t\sigma * \pi \Downarrow_j \Leftrightarrow u\sigma * \pi \Downarrow_j\right\}
\end{aligned}$$

We then take:

$$(\equiv) = \bigcap_{i \in \mathbb{N}} (\equiv_i) \qquad (\twoheadrightarrow) = \bigcup_{i \in \mathbb{N}} (\twoheadrightarrow_i)$$

Future work

Check the full details of the adequation lemma

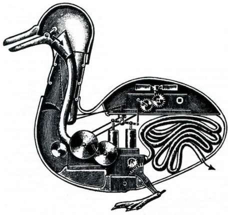
Add subtyping

Make sure we have enough rules

Implementation:

- Pseudo-algorithm for $\equiv$
- Hash-consing of the AST for efficiency
- Type checking
- ...

Thank you!



WWW.PATOLINE.ORG