Toward an Adequation Lemma for PML2

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 (explicated 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:

\[ \nu, \omega ::= x \mid \lambda x \, t \]

\[ t, u ::= \nu \mid t \, u \]

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:

\[ \nu, \omega ::= \ldots \mid C[\nu] \mid \{ \ldots l_i = \nu_i ; \ldots \} \]

\[ t, u ::= \ldots \mid Y(t, \nu) \mid \nu. l \mid \text{case } \nu \text{ of } [ \ldots C_i[x] \rightarrow t_i ; \ldots ] \]

We enforce values in many places to simplify the calculus.

We can define syntactic sugars:

\[ C[t] ::= (\lambda x C[x]) t \quad t \cdot l ::= (\lambda x \cdot l) t \]
Let's make the calculus classical

One possibility is to add a μ binder (λμ-calculus):

\[
\begin{align*}
t, u & ::= \ldots \mid \mu \alpha \ t \mid t \cdot \pi \\
\pi, \rho & ::= \alpha \mid \nu \cdot \pi \mid [t] \pi
\end{align*}
\]

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] π)
Summary of the syntax: Values, Terms, Stacks and Processes

\[
\begin{align*}
\nu, \omega & ::= \ x \mid \lambda x \ t \mid C[\nu] \mid \{ \ldots \ l_i = \nu_i \, ; \, \ldots \} & (\Lambda_{\nu}) \\
\tau, \rho & ::= \alpha \mid \nu \cdot \tau \mid [t] \tau & (\Pi) \\
\rho, s & ::= t^* \tau & (\Lambda^*\Pi)
\end{align*}
\]

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 $\beta$-reduction:

\[
(t \ u) \ast \pi \rightarrow u \ast [t] \pi \\
\nu \ast [t] \pi \rightarrow t \ast \nu \cdot \pi \\
(\lambda x \ t) \ast \nu \cdot \pi \rightarrow t[x \leftarrow \nu] \ast \pi
\]

Capturing and restoring the evaluation context:

\[
(\mu \alpha \ t) \ast \pi \rightarrow t[\alpha \leftarrow \pi] \ast \pi \\
p \ast \pi \rightarrow p
\]

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

Given a process \( p \) we write:

- \( p \downarrow \) if \( \exists v, \exists \alpha, p \Rightarrow^* v \alpha \)
- \( p \uparrow \) 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 $F$:

\[
A, B ::= \ \\tau \\
  \quad | \quad A \Rightarrow B \\
  \quad | \quad \forall X \ A \\
  \quad | \quad \exists X \ A
\]

We extend it to an ML-like system:

\[
A, B ::= \ldots \\
  \quad | \quad [ \ldots \ C_i[A_i]; \ \ldots \ ] \\
  \quad | \quad \{ \ldots \ t_i : A_i; \ \ldots \ \} \\
  \quad | \quad \mu X_n A
\]
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:

\[
A, B ::= \cdots
\]

\[
| X_n(t_1, \ldots, t_n)
|
| \forall X_n \ A
|
| \exists X_n \ A
\]

The variables of System \( F \) can be seen as predicates of arity 0
Full second-order type system

\[ A, B ::= X_n(t_1, \ldots, t_n) \]

\[ \mid A \Rightarrow B \]

\[ \mid \forall X_n A \mid \exists X_n A \]

\[ \mid [ \ldots C_i[A_i]; \ldots ] \]

\[ \mid \{ \ldots l_i : A_i; \ldots \} \]

\[ \mid \mu X_n A \]

\[ \mid \forall x A \mid \exists x A \]

\[ \mid t \in A \]

\[ \mid A \triangleright t \equiv u \]

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

We interpret terms and values as their equivalence classes
- $[[\nu]] = \{w \in \Lambda_{\nu} \mid \nu \equiv w\}$
- $[[t]] = \{u \in \Lambda \mid t \equiv u\}$

Raw semantics of formulas:
- $[[A \Rightarrow B]] = \{\lambda x \, t \mid \forall \nu \in [[A]], \, t[x \leftarrow \nu] \in [[B]] \}$
- $[[\forall X_n \, A]] = \bigcap_{p_n} [[A[X_n \leftarrow p_n]]]$
- $[[\forall x \, A]] = \bigcap_{\nu \in \Lambda_x} [[A[x \leftarrow \nu]]]$
- $[[t \in A]] = \{\nu \in [[A]] \mid \nu \equiv t\}$
- $[[A \upharpoonright t \equiv u]] = [[A]]$ if $t \equiv u$ and $\emptyset$ otherwise
- $\ldots$

The set $[[A]]$ is closed under $\equiv$ for all $A$ (by construction)
Pole, Falsity Values and Truth Values

We define a family of poles $\mathcal{U}_{(\forall_i, \alpha_i)_{i \in I}}$:

$$\mathcal{U}_{(\forall_i, \alpha_i)_{i \in I}} = \{ p \mid \exists i \in I, \exists v \in V_i, \exists w \equiv v, p \rightarrow^* w \alpha_i \}$$

Properties of a pole $\mathcal{U}$:
- They are closed under $(\rightarrow)^{-1}$
- And closed under $(\rightarrow)$
- If $v \alpha \in \mathcal{U}$ and $v \equiv w$ then $w \alpha \in \mathcal{U}$

For every formula $A$ we define:

$$\llbracket A \rrbracket_{\perp} = \{ \pi \in \Pi \mid \forall v \in \llbracket A \rrbracket, v \alpha \pi \in \mathcal{U} \}$$

$$\llbracket A \rrbracket_{\perp \perp} = \{ t \in \Lambda \mid \forall \pi \in \llbracket A \rrbracket_{\perp}, t \alpha \pi \in \mathcal{U} \}$$
Typing judgements and Adequation Lemma

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

\[ \Gamma \vdash v : A \quad \Gamma \vdash t : A \]

A context \( \Gamma \) contain:
- Type assignments of the form \( x : A \)
- Type assignments of the form \( \alpha : A' \)
- Equivalences / inequivalences of the form \( t \equiv u / t \not\equiv u \)

Theorem 1.

\[ \Gamma \vdash v : A \Rightarrow v' \in \llbracket A \rrbracket \quad \Gamma \vdash t : A \Rightarrow t' \in \llbracket A \rrbracket^{\perp\perp} \]
Adding adequate typing rules to the system

We can add any rule provided that it is adequate

Examples of adequate rules:

\[ \frac{}{\Gamma, x : A \vdash x : A}^{\text{Ax}} \]
\[ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x \cdot t : A \Rightarrow B}^{i} \]
\[ \frac{\Gamma \vdash t : A \Rightarrow B}{\Gamma \vdash u : A \Rightarrow e} \]
\[ \frac{\Gamma, \alpha : \downarrow A \vdash t : A}{\Gamma \vdash \mu \alpha \cdot t : A}^{\mu} \]
\[ \frac{\Gamma, \alpha : \downarrow A \vdash t : A}{\Gamma, \alpha : \downarrow A \vdash t \ast \alpha : B}^{*} \]
Proof of adequacy of \( \Rightarrow_c \)

We suppose \( t' \in \llbracket A \Rightarrow B \rrbracket \) and \( u' \in \llbracket B \rrbracket \)
We need to show \( (t' u') \in \llbracket B \rrbracket^\perp \)
We take \( \pi \in \llbracket B \rrbracket^\perp \) and show \( (t' u') \ast \pi \in \perp \)
It is enough to show \( u' \ast [t'] \pi \in \perp \)
It is enough to show \( [t'] \pi \in \llbracket B \rrbracket^\perp \)
We take \( v \in \llbracket B \rrbracket \) and show \( v \ast [t'] \pi \in \perp \)
It is enough to show \( t' \ast v \cdot \pi \in \perp \)
It is enough to show \( v \cdot \pi \in \llbracket A \Rightarrow B \rrbracket^\perp \)
We take \( \lambda x \ m \in \llbracket A \Rightarrow B \rrbracket \) and show \( \lambda x \ m \ast v \cdot \pi \in \perp \)
It is enough to show \( m[x \leftarrow v] \ast \pi \in \perp \)
It is enough to show \( m[x \leftarrow v] \in \llbracket B \rrbracket^\perp \)
This is true by definition of \( \llbracket A \Rightarrow B \rrbracket \)
Rules of System F

\[
\Gamma \vdash v : A \\
\frac{}{\Gamma \vdash \forall X A} \forall_i
\]

\[
\Gamma \vdash t : \forall X_n A \\
\frac{}{\Gamma \vdash t : A[X_n \leftarrow P_n]} \forall_e
\]

\[
\Gamma \vdash t : A[X_n \leftarrow P_n] \\
\frac{}{\Gamma \vdash t : \exists X_n A} \exists_i
\]

\[
\Gamma, x : A[X_n \leftarrow P_n] \vdash t : B \\
\frac{}{\Gamma, x : \exists X_n A \vdash t : B} \exists_e
\]
Records and case analysis

\[ \begin{align*}
\Gamma \vdash v : \{ \ldots l_i : A_i ; \ldots \} & \quad \times_e \\
\Gamma \vdash v \cdot l_i : A_i & \\
\Gamma \vdash \{ \ldots l_i = v_i ; \ldots \} : \{ \ldots l_i : A_i ; \ldots \} & \times_i \\
\Gamma \vdash v : A_i & +_i \\
\Gamma \vdash C_i[v] : [ \ldots C_i[A_i] ; \ldots ] & \\
\Gamma \vdash v : [ \ldots C_i[A_i] ; \ldots ] & \quad \Gamma, x : A_i, C_i[x] \equiv v \vdash t_i : B & \quad +_e \\
\Gamma \vdash \text{case } v \text{ of } [ \ldots C_i[x] \rightarrow t_i ; \ldots ] : B & \\
\end{align*} \]

Remark: equivalence in the premise of \( +_e \)
Quantification over individuals

\[
\Gamma \vdash v : A \\
\Gamma \vdash v : \forall x \ A \\
\Gamma \vdash t : \forall x \ A \\
\Gamma \vdash t : \exists x \ A \\
\Gamma \vdash t : A[x \leftarrow v] \\
\Gamma \vdash t : \exists x \ A \\
\Gamma, x : A[y \leftarrow v] \vdash t : B \\
\Gamma, x : \exists y \ A \vdash t : B
\]
Belonging and Restriction

\[ \frac{\Gamma \vdash v : A \quad \Gamma \vdash t \equiv v}{\Gamma \vdash v : t \in A} \quad \varepsilon \]

\[ \frac{\Gamma, x : A, x \equiv u \vdash t : B}{\Gamma, x : u \in A \vdash t : B} \quad \varepsilon \]

\[ \frac{\Gamma, x : A, u_1 \equiv u_2 \vdash t : C}{\Gamma, x : A \uparrow u_1 \equiv u_2 \vdash t : C} \quad \lVERT \]

\[ \frac{\vdash \varepsilon(\Gamma, u_1 \not\equiv u_2) \quad \Gamma, u_1 \equiv u_2 \vdash t : A}{\Gamma \vdash t : A \uparrow u_1 \equiv u_2} \quad \lVERT \]
**Dependent product**

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

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

For instance the elimination rule

$$\frac{\Gamma \vdash t : \Pi x : A \ B \quad \Gamma \vdash \nu : A}{\Gamma \vdash t \nu : B[x \leftarrow \nu]}$$

can be derived:

$$\frac{\Gamma \vdash t : \forall x \ (x \in A \Rightarrow B) \quad \Gamma \vdash \nu \in A}{\Gamma \vdash (t \nu) : B[x \leftarrow \nu]}$$
Value restriction

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

\[
\frac{\Gamma \vdash t : \Pi_x:A \ B \quad \Gamma \vdash v : A}{\Pi_e} \quad \Gamma \vdash t \ v : B[x \leftarrow v]
\]

The following rule is not valid:

\[
\frac{\Gamma \vdash t : \Pi_x:A \ B \quad \Gamma \vdash u : A}{\Pi_e} \quad \Gamma \vdash t \ u : B[x \leftarrow u]
\]

We would like to have at least:

\[
\frac{\Gamma, \ y \equiv u \vdash t : \Pi_x:A \ B \quad \Gamma, \ y \equiv u \vdash u : A}{\Pi_e} \quad \Gamma, \ y \equiv u \vdash \ t \ u : B[x \leftarrow u]
\]
**Derivation of \( \Pi_e \)**

Provided that we have:

\[
\frac{\Gamma, t_1 \equiv t_2 \vdash u : \Lambda[t_1]}{\Gamma, t_1 \equiv t_2 \vdash u : \Lambda[t_2]} \equiv_r \\
\frac{\Gamma, t_1 \equiv t_2 \vdash t_1 : \Lambda}{\Gamma, t_1 \equiv t_2 \vdash t_2 : \Lambda} \equiv_l
\]

We can derive the rule \( \Pi_e \) on \( t \) using \( x \equiv t \):

\[
\begin{array}{c}
P_1 \\
\Gamma, y \equiv u \vdash t : \Pi_{x:A} B \\
\Gamma, y \equiv u \vdash t y : B[x \leftarrow y] \equiv_l \\
\Gamma, y \equiv u \vdash t u : B[x \leftarrow u] \equiv_r \\
\end{array}
\]

\[
\begin{array}{c}
P_2 \\
\Gamma, y \equiv u \vdash u : A \\
\Gamma, y \equiv u \vdash y : A \equiv_l \\
\end{array}
\]

\[
\frac{\Gamma, y \equiv u \vdash y : A}{\Gamma, y \equiv u \vdash \Pi_{y:A} B[y]} \equiv_e
\]

\[
\frac{\Gamma, y \equiv u \vdash y : A}{\Gamma, y \equiv u \vdash \Pi_{y:A} B[y]} \equiv_e
\]
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:

\[
\text{Theorem 2.}
\]

If \( \Phi \subseteq \Lambda_v \) is closed under \( (\equiv) \) then \( \Phi = \Phi_{\uparrow\uparrow} \cap \Lambda_v \)

Direct consequence: \( v \in \llbracket A \rrbracket_{\uparrow\uparrow} \Rightarrow v \in \llbracket A \rrbracket \)

Remarks:
- \( \Phi \subseteq \Phi_{\uparrow\uparrow} \cap \Lambda_v \) is trivial
- \( \Phi \supseteq \Phi_{\uparrow\uparrow} \cap \Lambda_v \) is not true in general...
Main idea (sufficient condition)

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

\[ t, u ::= \ldots | \delta(v, w) \]

With the reduction rule:

\[ \delta(v, w) * \pi \rightarrow v * \pi \quad \text{if} \quad v \neq w \]

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

\[ \Phi \supseteq \Phi_{\perp} \cap \Lambda_v \]
Proof

Recall the definitions:

$$\Phi^\perp = \{ \pi \in \Pi \mid \forall \nu \in \Phi, \nu \ast \pi \in \bot \} \quad \Phi^{\perp \perp} = \{ t \in \Lambda \mid \forall \pi \in \Phi^\perp, \, t \ast \pi \in \bot \}$$

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

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

We need to find a stack $\pi_0 \in \Phi^\perp$ such that $\nu \ast \pi_0 \notin \bot$.

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

- $\forall \nu \in \Phi, \nu \ast \pi_0 \in \bot$
- $\nu \ast \pi_0 \notin \bot$

$\pi_0 = [\lambda x \, \delta(x, \nu)] \alpha$ is such a stack
A stratified model

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

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

\[
(\rightarrow_0) \quad = \quad (>)
\]

\[
(\rightarrow_{i+1}) \quad = \quad (\rightarrow_i) \cup \{\delta(v, w) * \pi, v * \pi | v \not\equiv_i w\}
\]

\[
(\equiv_i) \quad = \quad \{(t, u) | \forall j \leq i, \forall \pi \in \Pi, \forall \sigma, t\sigma * \pi \downarrow_j \iff u\sigma * \pi \downarrow_j\}
\]

We then take:

\[
(\equiv) = \bigcap_{i \in \mathbb{N}} (\equiv_i) \quad \quad \quad \quad \quad \quad (\rightarrow\rightarrow) = \bigcup_{i \in \mathbb{N}} (\rightarrow_i)
\]
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