A Classical Realisability Model for PML$_2$

with Semantical Value Restriction

Inria Saclay 22/02/2017

Rodolphe Lepigre (rodolphe.lepigre@univ-smb.fr)

Programs and Proofs

```
type rec N = [ Z | S of N ]
val rec add : N ⇒ N ⇒ N =
  fun n m →
    match n with
    | Z     → m
    | S[k]  → S[add k m]
```

```
type rec N = [ Z | S of N ]
val rec add : N ⇒ N ⇒ N =
  fun n m →
    match n with
    | Z    → m
    | S[k] → S[add k m]

val addZN : ∀n (add Z n ≡ n) = {}
```

```
type rec N = [ Z | S of N ]
val rec add : N ⇒ N ⇒ N =
  fun n m →
    match n with
    | Z    → m
    | S[k] → S[add k m]

val addZN : ∀n (add Z n ≡ n) = {}

// val addNZ : ∀n (add n Z ≡ n) = ...
// Cannot be proved.
```

Proofs and Typed Quantification

```
val rec addNZ : (n:N) ⇒ (add n Z ≡ n) =
  fun n →
    match n with
    | Z    → {}
    | S[k] → addNZ k; {}
```

```
val rec addNZ : (n:N) ⇒ (add n Z ≡ n) =
  fun n →
    match n with
    | Z    → {}
    | S[k] → addNZ k; {}

val rec addNSM : (n:N) ⇒ (m:N) ⇒ (add n S[m] ≡ S[add n m]) =
  fun n m →
    match n with
    | Z    → {}
    | S[k] → addNSM k m; {}
```

Mixing Proofs and Programs

```
val rec addComm : (n:N) ⇒ (m:N) ⇒ (add n m ≡ add m n) =
  fun n m →
    match n with
    | Z    → addNZ m; {}
    | S[k] → addComm k m; addNSM m; {}
```

```
val rec addComm : (n:N) ⇒ (m:N) ⇒ (add n m ≡ add m n) =
  fun n m →
    match n with
    | Z    → addNZ m; {}
    | S[k] → addComm k m; addNSM m; {}

val add : (n:N) ⇒ (m:N) ⇒ N | (add n m ≡ add m n) =
  fun n m →
    addComm n m; add n m
```

```
val rec addComm : (n:N) ⇒ (m:N) ⇒ (add n m ≡ add m n) =
  fun n m →
    match n with
    | Z    → addNZ m; {}
    | S[k] → addComm k m; addNSM m; {}

val add : (n:N) ⇒ (m:N) ⇒ N | (add n m ≡ add m n) =
  fun n m →
    addComm n m; add n m

val add' : N ⇒ N ⇒ N = add
```

$$
\begin{aligned}
v\,,\,w &::= x \mid \lambda x.t \mid \{(l_i = v_i)_{i\in I}\} \mid C_k[v] \mid \square \\
t\,,\,u &::= a \mid v \mid t\ u \mid \mu\alpha.t \mid [\pi]t \mid v.l_k \mid [v \mid (C_i[x_i] \to t_i)_{i\in I}] \mid F_{v,t} \mid R_{v,t} \mid \delta_{v,w} \\
\pi\,,\,\rho &::= \alpha \mid \varepsilon \mid v\cdot\pi \mid [t]\pi \\
p\,,\,q &::= t * \pi
\end{aligned}
$$

Evaluation in the Machine (1/2)

$$
\begin{aligned}
t\,u * \pi &\succ u * [t]\pi \\
v * [t]\pi &\succ t * v \cdot \pi \\
\lambda x.t * v \cdot \pi &\succ t[x := v] * \pi \\
\mu\alpha.t * \pi &\succ t[\alpha := \pi] * \pi \\
[\pi]t * \xi &\succ t * \pi \\
\{(l_i = v_i)_{i \in I}\}.l_k * \pi &\succ v_k * \pi && (k \in I) \\
[C_k[v] \mid (C_i[x_i] \to t_i)_{i \in I}] * \pi &\succ t_k[x_k := v] * \pi && (k \in I)
\end{aligned}
$$

Evaluation in the Machine (2/2)

$$\begin{aligned}
\square * v\,.\,\pi &\succ \square * \pi \\
\square.l_i * \pi &\succ \square * \pi \\
[\square \mid (C_i[x_i] \to t_i)_{i \in I}] * \pi &\succ \square * \pi \\
F_{\lambda x.u,t} * \pi &\succ t * \pi \\
R_{\{(l_i = v_i)_{i \in I}\},t} * \pi &\succ t * \pi
\end{aligned}$$

$$
\begin{aligned}
\mathrm{not}\;\; C_1[\{\}] * \varepsilon &= (\lambda x.[x \mid C_1[y] \to C_0[\{\}] \mid C_0[y] \to C_1[\{\}]])\;\; C_1[\{\}] * \varepsilon \\
&\succ C_1[\{\}] * [\lambda x.[x \mid C_1[y] \to C_0[\{\}] \mid C_0[y] \to C_1[\{\}]]]\varepsilon \\
&\succ \lambda x.[x \mid C_1[y] \to C_0[\{\}] \mid C_0[y] \to C_1[\{\}]] * C_1[\{\}] \,.\, \varepsilon \\
&\succ [C_1[\{\}] \mid C_1[y] \to C_0[\{\}] \mid C_0[y] \to C_1[\{\}]] * \varepsilon \\
&\succ C_0[\{\}] * \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\Omega * \varepsilon &= (\lambda x.x\;\, x)\;\; (\lambda x.x\;\, x) * \varepsilon \\
&\succ \lambda x.x\;\, x * [\lambda x.x\;\, x]\varepsilon \\
&\succ \lambda x.x\;\, x * \lambda x.x\;\, x \,.\, \varepsilon \\
&\succ (\lambda x.x\;\, x)\;\; (\lambda x.x\;\, x) * \varepsilon \\
&\succ \ldots
\end{aligned}
$$

Observational Equivalence over Programs

Observational Equivalence over Programs

It is easy to quantify over evaluation contexts (i.e. stacks).

It is easy to quantify over evaluation contexts (i.e. stacks).

We define $p\Downarrow$ as $\exists\, v\,,\; p \succ^* v * \varepsilon$.

It is easy to quantify over evaluation contexts (i.e. stacks).

We define $p\Downarrow$ as $\exists\, v\,,\ p \succ^* v * \varepsilon$.

$$(\equiv) = \{(t\,,\ u) \mid \forall\,\pi, \forall\,\rho\,,\ t\rho * \pi\Downarrow \Leftrightarrow u\rho * \pi\Downarrow\}$$

It is easy to quantify over evaluation contexts (i.e. stacks).

We define $p \Downarrow$ as $\exists\, v,\ p \succ^* v * \varepsilon$.

$$(\equiv) = \{(t\,,\,u) \mid \forall\,\pi, \forall\,\rho\,,\ t\rho * \pi \Downarrow\ \Leftrightarrow\ u\rho * \pi \Downarrow\}$$

We quantify over substitutions to handle free variables.

Example of Derivable Equivalences

Example of Derivable Equivalences

For all x, v, t we have $(\lambda x.t)\ v \equiv t[x := v]$.

For all x, v, t we have $(\lambda x.t)\ v \equiv t[x := v]$.

$$
\begin{aligned}
((\lambda x.t)\ v)\rho * \pi &= (\lambda x.t\rho)\ v\rho * \pi \\
&\succ v\rho * [\lambda x.t\rho]\pi \\
&\succ \lambda x.t\rho * v\rho \,.\, \pi \\
&\succ t\rho[x := v\rho] * \pi \\
&= (t[x := v])\rho * \pi
\end{aligned}
$$

More Equivalences: Canonical Values

$$x \equiv v \quad \Leftrightarrow \quad v = x$$

$$\square \equiv v \quad \Leftrightarrow \quad v = \square$$

$$C_k[v_k] \equiv v \quad \Leftrightarrow \quad v = C_k[w_k] \text{ and } v_k \equiv w_k$$

$$\{(l_i = v_i)_{i \in I}\} \equiv v \quad \Leftrightarrow \quad v = \{(l_i = w_i)_{i \in I}\} \text{ and } \forall\, i \in I,\ v_i \equiv w_i$$

$$\lambda x.t \equiv v \quad \Leftrightarrow \quad v = \lambda y.u \text{ and } t \equiv u[y := x]$$

Value Interpretation of Types

A type A is interpreted as a set of values $[\![A]\!]$.

We require $[\![A]\!]$ to be closed under $(\equiv)$.

We require $\square \in [\![A]\!]$.

We have $[\![A]\!] \in \{\{\square\} \subseteq \Phi \subseteq \Lambda_\iota \mid v \in \Phi \wedge w \equiv v \Rightarrow w \in \Phi\}$.

(Λ_ι is the set of all the values.)

Value Interpretation of Pure Types

$$\left[\!\left[\{(l_i : A_i)_{i\in I}\}\right]\!\right] = \left\{\{(l_i = v_i)_{i\in I}\} \mid \forall\, i \in I,\, v_i \in [\![A_i]\!]\right\} \cup \{\square\}$$

$$\left[\!\left[[(C_i : A_i)_{i\in I}]\right]\!\right] = \cup_{i\in I}\{C_i[v] \mid v \in [\![A_i]\!]\} \cup \{\square\}$$

$$[\![\forall X.A]\!] = \cap_{\Phi}[\![A[X := \Phi]]\!]$$

$$[\![\exists X.A]\!] = \cup_{\Phi}[\![A[X := \Phi]]\!]$$

$$[\![\forall a.A]\!] = \cap_{t\in\Lambda}[\![A[a := t]]\!]$$

$$[\![\exists a.A]\!] = \cup_{t\in\Lambda}[\![A[a := t]]\!]$$

Function Type and Terms

Function Type and Terms

$$[\![A \Rightarrow B]\!] = \{\lambda x.w \mid \forall\, v \in [\![A]\!]\,,\, w[x := v] \in [\![B]\!]\} \cup \{\square\}$$

Function Type and Terms

$$\llbracket A \Rightarrow B \rrbracket = \{\lambda x.w \mid \forall\, v \in \llbracket A \rrbracket\,,\, w[x := v] \in \llbracket B \rrbracket\} \cup \{\square\}$$

What about programs that actually compute something?

$$[\![A \Rightarrow B]\!] = \{\lambda x.w \mid \forall\, v \in [\![A]\!]\,,\, w[x := v] \in [\![B]\!]\} \cup \{\square\}$$

What about programs that actually compute something?

We define a completion operation $[\![A]\!] \mapsto [\![A]\!]^{\perp\!\!\!\perp\perp\!\!\!\perp}$.

$$[\![A \Rightarrow B]\!] = \{\lambda x.w \mid \forall\, v \in [\![A]\!]\,,\, w[x := v] \in [\![B]\!]\} \cup \{\square\}$$

What about programs that actually compute something?

We define a completion operation $[\![A]\!] \mapsto [\![A]\!]^{\perp\!\!\perp}$.

The set $[\![A]\!]^{\perp\!\!\perp}$ contains terms "behaving" as values of $[\![A]\!]$.

$$\llbracket A \Rightarrow B \rrbracket = \{\lambda x.w \mid \forall\, v \in \llbracket A \rrbracket\,,\, w[x := v] \in \llbracket B \rrbracket\} \cup \{\square\}$$

What about programs that actually compute something?

We define a completion operation $\llbracket A \rrbracket \mapsto \llbracket A \rrbracket^{\perp\!\!\!\perp\perp\!\!\!\perp}$.

The set $\llbracket A \rrbracket^{\perp\!\!\!\perp\perp\!\!\!\perp}$ contains terms "behaving" as values of $\llbracket A \rrbracket$.

We can then take $\llbracket A \Rightarrow B \rrbracket = \{\lambda x.t \mid \forall\, v \in \llbracket A \rrbracket\,,\, t[x := v] \in \llbracket B \rrbracket^{\perp\!\!\!\perp\perp\!\!\!\perp}\} \cup \{\square\}$

Pole and Orthogonality

The definition of $[\![A]\!]^{\perp\!\!\!\perp\perp\!\!\!\perp}$ is parametrised by a set $\perp\!\!\!\perp \subseteq \Lambda \times \Pi$.

The definition of $[\![A]\!]^{\perp\!\!\!\perp\perp\!\!\!\perp}$ is parametrised by a set $\perp\!\!\!\perp \subseteq \Lambda \times \Pi$.

Intuitively, $\perp\!\!\!\perp$ is a set of processes that "behave well".

Pole and Orthogonality

The definition of $[\![A]\!]^{\perp\!\!\!\perp\perp\!\!\!\perp}$ is parametrised by a set $\perp\!\!\!\perp \subseteq \Lambda \times \Pi$.

Intuitively, $\perp\!\!\!\perp$ is a set of processes that "behave well".

The set $\perp\!\!\!\perp = \{p \in \Lambda \times \Pi \mid \exists\, v \in \Lambda_\iota,\ p \succ^* v * \varepsilon\}$ is a good choice.

The definition of $[\![A]\!]^{\perp\!\!\!\perp\perp\!\!\!\perp}$ is parametrised by a set $\perp\!\!\!\perp \subseteq \Lambda \times \Pi$.

Intuitively, $\perp\!\!\!\perp$ is a set of processes that "behave well".

The set $\perp\!\!\!\perp = \{p \in \Lambda \times \Pi \mid \exists\, v \in \Lambda_\iota,\, p \succ^* v * \varepsilon\}$ is a good choice.

$$
\begin{aligned}
[\![A]\!] &\in \{\{\square\} \subseteq \Phi \subseteq \Lambda_\iota \mid v \in \Phi \wedge v \equiv w \Rightarrow w \in \Phi\} \\
[\![A]\!]^{\perp\!\!\!\perp} &= \{\pi \in \Pi \mid \forall\, v \in [\![A]\!],\, v * \pi \in \perp\!\!\!\perp\} \\
[\![A]\!]^{\perp\!\!\!\perp\perp\!\!\!\perp} &= \{t \in \Lambda \mid \forall\, \pi \in [\![A]\!]^{\perp\!\!\!\perp},\, t * \pi \in \perp\!\!\!\perp\}
\end{aligned}
$$

Typing judgments and adequacy

Typing judgments and adequacy

There are two forms of judgments: $\Xi \vdash_{\mathrm{val}} v : A$ and $\Xi \vdash \mathrm{t} : A$.

Typing judgments and adequacy

There are two forms of judgments: $\Xi \vdash_{\mathrm{val}} v : A$ and $\Xi \vdash t : A$.

The context Ξ contains only equivalences of the form $u_1 \equiv u_2$.

Typing judgments and adequacy

There are two forms of judgments: $\Xi \vdash_{\mathrm{val}} v : A$ and $\Xi \vdash t : A$.

The context Ξ contains only equivalences of the form $u_1 \equiv u_2$.

Everything is closed (choice operator / witness presentation).

Typing judgments and adequacy

There are two forms of judgments: $\Xi \vdash_{\mathrm{val}} v : A$ and $\Xi \vdash t : A$.

The context Ξ contains only equivalences of the form $u_1 \equiv u_2$.

Everything is closed (choice operator / witness presentation).

Adequacy for terms: if $\Xi \vdash t : A$ is derivable and Ξ is valid then $[\![t]\!] \in [\![A]\!]^{\perp\!\!\!\perp\perp\!\!\!\perp}$.

There are two forms of judgments: $\Xi \vdash_{\mathrm{val}} v : A$ and $\Xi \vdash t : A$.

The context Ξ contains only equivalences of the form $u_1 \equiv u_2$.

Everything is closed (choice operator / witness presentation).

Adequacy for terms: if $\Xi \vdash t : A$ is derivable and Ξ is valid then $[\![t]\!] \in [\![A]\!]^{\perp\!\!\!\perp}$.

Adequacy for values: if $\Xi \vdash_{\mathrm{val}} v : A$ is derivable and Ξ is valid then $[\![v]\!] \in [\![A]\!]$.

There are two forms of judgments: $\Xi \vdash_{\mathrm{val}} v : A$ and $\Xi \vdash t : A$.

The context Ξ contains only equivalences of the form $u_1 \equiv u_2$.

Everything is closed (choice operator / witness presentation).

Adequacy for terms: if $\Xi \vdash t : A$ is derivable and Ξ is valid then $[\![t]\!] \in [\![A]\!]^{\perp\!\!\!\perp}$.

Adequacy for values: if $\Xi \vdash_{\mathrm{val}} v : A$ is derivable and Ξ is valid then $[\![v]\!] \in [\![A]\!]$.

Since $[\![A]\!] \subseteq [\![A]\!]^{\perp\!\!\!\perp}$ we have the rule $\dfrac{\Xi \vdash_{\mathrm{val}} v : A}{\Xi \vdash v : A}\uparrow$.

Rather Usual Typing Rules

Rather Usual Typing Rules

$$\frac{\Xi \vdash t[x := \varepsilon_{x\in A}(t \notin B)] : B}{\Xi \vdash_{\mathrm{val}} \lambda x.t : A \Rightarrow B}\Rightarrow_i \qquad \frac{\Xi \vdash t : A \Rightarrow B \qquad \Xi \vdash u : A}{\Xi \vdash t\, u : B}\Rightarrow_e$$

$$\frac{}{\Xi \vdash_{\mathrm{val}} \varepsilon_{x\in A}(t \notin B) : A}\mathrm{Ax}$$

Rather Usual Typing Rules

$$\frac{\Xi \vdash t[x := \varepsilon_{x \in A}(t \notin B)] : B}{\Xi \vdash_{\mathrm{val}} \lambda x.t : A \Rightarrow B} \Rightarrow_i \qquad \frac{\Xi \vdash t : A \Rightarrow B \qquad \Xi \vdash u : A}{\Xi \vdash t\ u : B} \Rightarrow_e$$

$$\frac{}{\Xi \vdash_{\mathrm{val}} \varepsilon_{x \in A}(t \notin B) : A} \mathrm{Ax}$$

$$\frac{(\Xi \vdash_{\mathrm{val}} v_i : A_i)_{i \in I}}{\Xi \vdash_{\mathrm{val}} \{(l_i = v_i)_{i \in I}\} : \{(l_i : A_i)_{i \in I}\}} \times_i \qquad \frac{\Xi \vdash_{\mathrm{val}} v : \{(l_i : A_i)_{i \in I}\} \qquad k \in I}{\Xi \vdash v.l_k : A_k} \times_e$$

Rather Usual Typing Rules

$$\frac{\Xi \vdash t[x := \varepsilon_{x\in A}(t \notin B)] : B}{\Xi \vdash_{\text{val}} \lambda x.t : A \Rightarrow B}\Rightarrow_i \qquad \frac{\Xi \vdash t : A \Rightarrow B \quad \Xi \vdash u : A}{\Xi \vdash t\,u : B}\Rightarrow_e$$

$$\frac{}{\Xi \vdash_{\text{val}} \varepsilon_{x\in A}(t \notin B) : A}\text{Ax}$$

$$\frac{(\Xi \vdash_{\text{val}} v_i : A_i)_{i\in I}}{\Xi \vdash_{\text{val}} \{(l_i = v_i)_{i\in I}\} : \{(l_i : A_i)_{i\in I}\}}\times_i \qquad \frac{\Xi \vdash_{\text{val}} v : \{(l_i : A_i)_{i\in I}\} \quad k \in I}{\Xi \vdash v.l_k : A_k}\times_e$$

$$\frac{\Xi \vdash_{\text{val}} v : A[X := \varepsilon_X(v \notin A)]}{\Xi \vdash_{\text{val}} v : \forall X.A}\forall_i \qquad \frac{\Xi \vdash t : \forall X.A}{\Xi \vdash t : A[X := B]}\forall_e$$

Equivalence types

$$[\![A \restriction u_1 \equiv u_2]\!] = \{v \in [\![A]\!] \mid u_1 \equiv u_2\} \cup \{\square\}$$

$u_1 \equiv u_2$ is defined as $\{\} \restriction u_1 \equiv u_2$.

Equivalence types

$$[\![A \upharpoonright u_1 \equiv u_2]\!] = \{v \in [\![A]\!] \mid u_1 \equiv u_2\} \cup \{\square\}$$

$u_1 \equiv u_2$ is defined as $\{\} \upharpoonright u_1 \equiv u_2$.

$$\frac{\Xi \vdash t : A \qquad \Xi \vdash u_1 \equiv u_2}{\Xi \vdash t : A \upharpoonright u_1 \equiv u_2}\upharpoonright_i \qquad \frac{\Xi\,,\, u_1 \equiv u_2 \vdash_{\mathrm{val}} \varepsilon_{x \in A}(t \notin B) : C}{\Xi \vdash_{\mathrm{val}} \varepsilon_{x \in A \upharpoonright u_1 \equiv u_2}(t \notin B) : C}\upharpoonright_e$$

Singleton and typed quantification

Singleton and typed quantification

$$[\![t \in A]\!] = \{ v \in [\![A]\!] \mid t \equiv v \} \cup \{ \square \}$$

$(a : A) \Rightarrow B$ is defined as $\forall a.(a \in A \Rightarrow B)$.

Singleton and typed quantification

$$[\![t \in A]\!] = \{v \in [\![A]\!] \mid t \equiv v\} \cup \{\square\}$$

$(a : A) \Rightarrow B$ is defined as $\forall a.(a \in A \Rightarrow B)$.

$$\frac{\Xi \vdash_{\text{val}} v : A}{\Xi \vdash_{\text{val}} v : v \in A}\in_i \qquad \frac{\Xi, \varepsilon_{x \in A}(t \notin B) \equiv u \vdash \varepsilon_{x \in A}(t \notin B) : C}{\Xi \vdash \varepsilon_{x \in (u \in A)}(t \notin B) : C}\in_e$$

Singleton and typed quantification

$$\llbracket t \in A \rrbracket = \{v \in \llbracket A \rrbracket \mid t \equiv v\} \cup \{\square\}$$

$(a : A) \Rightarrow B$ is defined as $\forall a.(a \in A \Rightarrow B)$.

$$\frac{\Xi \vdash_{\mathrm{val}} v : A}{\Xi \vdash_{\mathrm{val}} v : v \in A}\in_i \qquad \frac{\Xi, \varepsilon_{x\in A}(t \notin B) \equiv u \vdash \varepsilon_{x\in A}(t \notin B) : C}{\Xi \vdash \varepsilon_{x\in(u\in A)}(t \notin B) : C}\in_e$$

$$\frac{\Xi \vdash t[x := \varepsilon_{x\in A}(t \notin B[a := x])] : B[a := \varepsilon_{x\in A}(t \notin B[a := x])]}{\Xi \vdash_{\mathrm{val}} \lambda x.t : (a : A) \Rightarrow B}$$

$$\frac{\Xi \vdash t : (a : A) \Rightarrow B \qquad \Xi \vdash_{\mathrm{val}} v : A}{\Xi \vdash t\, v : B[a := v]}$$

Equivalence Learning and Congruence

$$\dfrac{\Xi \vdash v : [(C_i : A_i)_{i \in I}] \qquad \left(\Xi\,,\, v \equiv C_i[\varepsilon_{x_i \in A_i}(t_i \notin C)] \vdash t_i[x_i \coloneqq \varepsilon_{x_i \in A_i}(t_i \notin C)] : C\right)_{i \in I}}{\Xi \vdash [v \,|\, (C_i[x_i] \to t_i)_{i \in I}] : C}\,+_e$$

Equivalence Learning and Congruence

$$\frac{\Xi \vdash v : [(C_i : A_i)_{i \in I}] \qquad \left(\Xi, v \equiv C_i[\varepsilon_{x_i \in A_i}(t_i \notin C)] \vdash t_i[x_i := \varepsilon_{x_i \in A_i}(t_i \notin C)] : C\right)_{i \in I}}{\Xi \vdash [v \mid (C_i[x_i] \to t_i)_{i \in I}] : C} +_e$$

$$\frac{\Xi \vdash t : v \in A \Rightarrow B \qquad \Xi \vdash_{\mathrm{val}} v : A}{\Xi \vdash t\ v : B} \Rightarrow_{e,\varepsilon}$$

$$\frac{\Xi \vdash v : [(C_i : A_i)_{i \in I}] \qquad \left(\Xi, v \equiv C_i[\varepsilon_{x_i \in A_i}(t_i \notin C)] \vdash t_i[x_i := \varepsilon_{x_i \in A_i}(t_i \notin C)] : C\right)_{i \in I}}{\Xi \vdash [v \mid (C_i[x_i] \to t_i)_{i \in I}] : C} +_e$$

$$\frac{\Xi \vdash t : v \in A \Rightarrow B \qquad \Xi \vdash_{\mathrm{val}} v : A}{\Xi \vdash t\ v : B} \Rightarrow_{e,\varepsilon}$$

$$\frac{\Xi \vdash t[a := u_1] : A[a := u_1] \qquad \Xi \vdash u_1 \equiv u_2}{\Xi \vdash t[a := u_2] : A[a := u_2]} \equiv$$

Semantical Value Restriction

A Classical Realizability Model for a Semantical Value Restriction (ESOP 2016).

$$\frac{\Xi \vdash_{\mathrm{val}} v : A}{\Xi \vdash_{\mathrm{val}} v : v{\in}A}\in_i \qquad\qquad \frac{\Xi \vdash t : A \qquad \Xi \vdash v \equiv t}{\Xi \vdash t : t{\in}A}\in_i$$

A Classical Realizability Model for a Semantical Value Restriction (ESOP 2016).

$$\frac{\Xi \vdash_{\text{val}} v : A}{\Xi \vdash_{\text{val}} v : v \in A}\in_i \qquad\qquad \frac{\Xi \vdash t : A \qquad \Xi \vdash v \equiv t}{\Xi \vdash t : t \in A}\in_i$$

Having the rule $\dfrac{\Xi \vdash v : A}{\Xi \vdash_{\text{val}} v : A}\downarrow$ is enough.

A Classical Realizability Model for a Semantical Value Restriction (ESOP 2016).

$$\frac{\Xi \vdash_{\text{val}} v : A}{\Xi \vdash_{\text{val}} v : v \in A}\,\in_i \qquad\qquad \frac{\Xi \vdash t : A \qquad \Xi \vdash v \equiv t}{\Xi \vdash t : t \in A}\,\in_i$$

Having the rule $\dfrac{\Xi \vdash v : A}{\Xi \vdash_{\text{val}} v : A}\downarrow$ is enough.

Relaxed rules can be derived using $(\downarrow)$, $(\uparrow)$ and $(\equiv)$.

The new instruction trick

The property $[\![A]\!]^{\perp\!\!\!\perp\perp\!\!\!\perp} \cap \Lambda_\iota \subseteq [\![A]\!]$ is not true in every realizability model.

The new instruction trick

The property $[\![A]\!]^{\perp\!\!\!\perp\perp\!\!\!\perp} \cap \Lambda_\iota \subseteq [\![A]\!]$ is not true in every realizability model.

To obtain it we extend the system with a new term constructor $\delta_{v,w}$ with the rule $\delta_{v,w} * \pi \succ v * \pi$ when $v \not\equiv w$.

The new instruction trick

The property $[\![A]\!]^{\perp\!\!\!\perp\perp\!\!\!\perp} \cap \Lambda_\iota \subseteq [\![A]\!]$ is not true in every realizability model.

To obtain it we extend the system with a new term constructor $\delta_{v,w}$ with the rule $\delta_{v,w} * \pi \succ v * \pi$ when $v \not\equiv w$.

Idea of the proof:

- suppose $v \notin [\![A]\!]$ and show $v \notin [\![A]\!]^{\perp\!\!\!\perp\perp\!\!\!\perp}$,
- we need to find π such that $v * \pi \notin \perp\!\!\!\perp$ and $\forall\, w \in [\![A]\!],\, w * \pi \in \perp\!\!\!\perp$,
- we can take $\pi = [\lambda x.\delta_{x,v}]\varepsilon$,
- $v * [\lambda x.\delta_{x,v}]\varepsilon \succ \lambda x.\delta_{x,v} * v \,.\, \varepsilon \succ \delta_{v,v} * \varepsilon$,
- $w * [\lambda x.\delta_{x,v}]\varepsilon \succ \lambda x.\delta_{x,v} * w \,.\, \varepsilon \succ \delta_{w,v} * \varepsilon \succ w * \varepsilon$.

Stratified reduction and equivalence

The definitions of $(\succ)$ and $(\equiv)$ are circular.

The definitions of $(\succ)$ and $(\equiv)$ are circular.

We need to rely on a stratified construction of the two relations.

$$(\twoheadrightarrow_i) \;=\; (\succ) \cup \left\{(\delta_{v,w} * \pi, v * \pi) \mid \exists\, j < i\,,\, v \not\equiv_j w\right\}$$

$$(\cong_i) \;=\; \left\{(t\,,\, u) \mid \forall\, j \leqslant i\,, \forall\, \pi \in \Pi\,, \forall\, \rho\,,\, t\rho * \pi \Downarrow_j \Leftrightarrow u\rho * \pi \Downarrow_j\right\}$$

The definitions of $(\succ)$ and $(\equiv)$ are circular.

We need to rely on a stratified construction of the two relations.

$$(\twoheadrightarrow_i) \;=\; (\succ) \cup \left\{(\delta_{v,w} * \pi,\, v * \pi) \mid \exists\, j < i,\, v \not\equiv_j w\right\}$$

$$(\cong_i) \;=\; \left\{(t,\, u) \mid \forall\, j \leqslant i, \forall\, \pi \in \Pi, \forall\, \rho,\, t\rho * \pi \Downarrow_j \Leftrightarrow u\rho * \pi \Downarrow_j\right\}$$

We then take $(\twoheadrightarrow) = \cup_{i \in \mathbb{N}} (\twoheadrightarrow_i)$ and $(\cong) = \cap_{i \in \mathbb{N}} (\cong_i)$.

$$(\cong) \;=\; \left\{(t\,,\,u) \mid \forall\, i \in \mathbb{N}\,, \forall\, \pi \in \Pi\,, \forall\, \rho\,,\, t\rho * \pi \Downarrow_i \;\Leftrightarrow\; u\rho * \pi \Downarrow_i\right\}$$

$$(\twoheadrightarrow) \;=\; (\succ) \cup \{(\delta_{v,w} * \pi\,,\, v * \pi) \mid v \ncong w\}$$

Compatible equivalence

$$(\cong) \;=\; \big\{(t\,,\,u) \mid \forall\, i \in \mathbb{N}\,, \forall\, \pi \in \Pi\,, \forall\, \rho\,,\; t\rho * \pi \Downarrow_i \Leftrightarrow u\rho * \pi \Downarrow_i\big\}$$

$$(\twoheadrightarrow) \;=\; (\succ) \cup \{(\delta_{v,w} * \pi\,,\, v * \pi) \mid v \not\cong w\}$$

The relation $(\cong)$ is “compatible” with $(\equiv)$.

$$
\begin{aligned}
(\cong) &= \left\{(t\,,\,u) \mid \forall\, i \in \mathbb{N}, \forall\, \pi \in \Pi, \forall\, \rho\,,\, t\rho * \pi \Downarrow_i \Leftrightarrow u\rho * \pi \Downarrow_i\right\} \\
(\twoheadrightarrow) &= (\succ) \cup \{(\delta_{v,w} * \pi\,,\, v * \pi) \mid v \ncong w\}
\end{aligned}
$$

The relation $(\cong)$ is “compatible” with $(\equiv)$.

In particular we have $(\cong) \subseteq (\equiv)$.

$$(\cong) \;=\; \left\{(t\,,\,u) \mid \forall\, i \in \mathbb{N}\,, \forall\, \pi \in \Pi\,, \forall\, \rho\,,\ t\rho * \pi \Downarrow_i \Leftrightarrow u\rho * \pi \Downarrow_i\right\}$$

$$(\twoheadrightarrow) \;=\; (\succ) \cup \{(\delta_{v,w} * \pi\,,\ v * \pi) \mid v \not\cong w\}$$

The relation $(\cong)$ is “compatible” with $(\equiv)$.

In particular we have $(\cong) \subseteq (\equiv)$.

If for all π there is p such that $t * \pi \succ^* p$ and $u * \pi \succ^* p$ then $t \cong u$.

Work in progress and future work

Work in progress and future work

Implementation of the system (in progress).

Inductive and coinductive types (in progress).

Recursion, termination checking (in progress).

Work in progress and future work

Implementation of the system (in progress).

Inductive and coinductive types (in progress).

Recursion, termination checking (in progress).

Compile PML programs (future work).

Mixing terminating / non-terminating programs (future work).

Work in progress and future work

Implementation of the system (in progress).

Inductive and coinductive types (in progress).

Recursion, termination checking (in progress).

PhD thesis (coming soon).

Compile PML programs (future work).

Mixing terminating / non-terminating programs (future work).

Fin.