A computational analysis of proof transformation by forcing

Alexandre Miquel

June 1st, 2010 – LAMA, Chambéry

Introduction

- The forcing technique :
  - Introduced by Cohen to prove $\text{Cons}(\text{ZFC} + \neg\text{HC})$
  - Formulæ interpreted as sets of conditions (belonging to a fixed poset C)
  - Formula translation : $A \mapsto p \Vdash A$ $(p \in C)$
- Krivine's interpretation of forcing (in 2nd/3rd order arithmetic)
  - Underlying program transformation $t \mapsto t^*$ (on Curry-style proof-terms)
  - Correctness expressed via generalized realizability structures
- The aims of this talk :
  - Rephrase the translation in $\text{PA}\omega^+$ (independently from realizability)
  - Present the underlying program transformation $t \mapsto t^*$ and study its computational contents
  - Reveal the underlying computation model (i.e. abstract machine)

Plan

Plan

Higher-order arithmetic ($PA\omega^+$)

- A multi-sorted language that allows to express
  - Individuals (sort ι)
  - Propositions (sort o)
  - Functions over individuals ($\iota \to \iota, \quad \iota \to \iota \to \iota, \quad \ldots$)
  - Predicates over individuals ($\iota \to o, \quad \iota \to \iota \to o, \quad \ldots$)
  - Predicates over predicates... ($(\iota \to o) \to o, \quad \ldots$)

Syntax of sorts (kinds) and higher-order terms

Sorts $\quad \tau, \sigma \quad ::= \quad \iota \quad | \quad o \quad | \quad \tau \to \sigma$

Terms $\quad M, N, A, B \quad ::= \quad x^\tau \quad | \quad \lambda x^\tau . M \quad | \quad MN \quad | \quad 0 \quad | \quad \mathsf{s} \quad | \quad \mathrm{rec}_\tau$
$\quad | \quad A \Rightarrow B \quad | \quad \forall x^\tau \, A \quad | \quad \langle M = M' \rangle A$

- Implication without computational contents : $\langle M = M' \rangle A$
  - Means : A if $M = M'$ (equality of denotations)
    $\top$ otherwise ($\top$ = type of all proofs)
  - Provably equivalent to : $M =_\tau M' \Rightarrow A$ (Leibniz equality)

Conversion (1/2)

- Conversion $M \cong_{\mathcal{E}} M'$ parameterized by a (finite) set of equations

$$\mathcal{E} \equiv M_1 = M_1', \ldots, M_k = M_k' \qquad \text{(non oriented, well sorted)}$$

- Reflexivity, symmetry, transitivity + base case :

$$\frac{}{M \cong_{\mathcal{E}} M'}\ (M=M')\in\mathcal{E}$$

- β-conversion, recursion :

$$\begin{array}{rcl} (\lambda x^{\tau} . M)N & \cong_{\mathcal{E}} & M\{x := N\} \\ \mathrm{rec}_{\tau}\, M\, M'\, 0 & \cong_{\mathcal{E}} & M \\ \mathrm{rec}_{\tau}\, M\, M'\, (\mathrm{s}\, N) & \cong_{\mathcal{E}} & M'\, N\, (\mathrm{rec}_{\tau}\, M\, M'\, N) \end{array}$$

- Usual context rules + extended rule for $\langle M = M' \rangle A$:

$$\frac{A \cong_{\mathcal{E}, M=M'} A'}{\langle M = M' \rangle A \cong_{\mathcal{E}} \langle M = M' \rangle A'}$$

Conversion (2/2)

- Rules for identifying (computationally equivalent) propositions :

$$
\begin{array}{rcll}
\forall x^\tau \, \forall y^\sigma \, A & \cong_{\mathcal{E}} & \forall y^\sigma \, \forall x^\tau \, A & \\
\forall x^\tau \, A & \cong_{\mathcal{E}} & A & x^\tau \notin FV(A) \\
A \Rightarrow \forall x^\tau \, B & \cong_{\mathcal{E}} & \forall x^\tau \, (A \Rightarrow B) & x^\tau \notin FV(A) \\
\langle M = M' \rangle \langle N = N' \rangle A & \cong_{\mathcal{E}} & \langle N = N' \rangle \langle M = M' \rangle A & \\
\langle M = M \rangle A & \cong_{\mathcal{E}} & A & \\
A \Rightarrow \langle M = M' \rangle B & \cong_{\mathcal{E}} & \langle M = M' \rangle (A \Rightarrow B) & \\
\forall x^\tau \langle M = M' \rangle A & \cong_{\mathcal{E}} & \langle M = M' \rangle \forall x^\tau A & x^\tau \notin FV(M, M')
\end{array}
$$

- Example : $\top \quad := \quad \langle \mathrm{tt} = \mathrm{ff} \rangle \bot$ (type of all proof-terms)

  where $\mathrm{tt} \equiv \lambda x^o y^o \, . \, x$, $\mathrm{ff} \equiv \lambda x^o y^o \, . \, y$ and $\bot \equiv \forall z^o \, z$

Deduction system (typing)

- Proof terms : $t, u \quad ::= \quad x \mid \lambda x\,.\,t \mid tu \mid \text{œ}$ (Curry-style)
- Contexts : $\Gamma \quad ::= \quad x_1 : A_1, \ldots, x_n : A_n$ (A_i of sort o)

Deduction/typing rules

$$\frac{}{\mathcal{E};\Gamma \vdash x : A}\ {\scriptstyle (x:A)\in\Gamma} \qquad \frac{\mathcal{E};\Gamma \vdash t : A}{\mathcal{E};\Gamma \vdash t : A'}\ {\scriptstyle A\cong_{\mathcal{E}} A'}$$

$$\frac{\mathcal{E};\Gamma, x : A \vdash t : B}{\mathcal{E};\Gamma \vdash \lambda x\,.\,t : A \Rightarrow B} \qquad \frac{\mathcal{E};\Gamma \vdash t : A \Rightarrow B \qquad \mathcal{E};\Gamma \vdash u : A}{\mathcal{E};\Gamma \vdash tu : B}$$

$$\frac{\mathcal{E}, M = M';\Gamma \vdash t : A}{\mathcal{E};\Gamma \vdash t : \langle M = M' \rangle A} \qquad \frac{\mathcal{E};\Gamma \vdash t : \langle M = M \rangle A}{\mathcal{E};\Gamma \vdash t : A}$$

$$\frac{\mathcal{E};\Gamma \vdash t : A}{\mathcal{E};\Gamma \vdash t : \forall x^\tau A}\ {\scriptstyle x^\tau \notin FV(\mathcal{E};\Gamma)} \qquad \frac{\mathcal{E};\Gamma \vdash t : \forall x^\tau A}{\mathcal{E};\Gamma \vdash t : A\{x := N^\tau\}}$$

$$\frac{}{\mathcal{E};\Gamma \vdash \text{œ} : ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$$

Remark : All proof-terms have type $\top \equiv \langle \mathrm{tt} = \mathrm{ff} \rangle \bot$ (normalization fails)

From operational semantics...

- Krivine's λ_c-calculus
  - λ-calculus with call/cc and continuation constants :

$$t, u \quad ::= \quad x \quad | \quad \lambda x\,.\,t \quad | \quad tu \quad | \quad \text{\ae} \quad | \quad \mathsf{k}_\pi$$

  - An abstract machine with explicit stacks :
    - Stack = list of closed terms (notation : π, π')
    - Process = closed term $\star$ stack

- Evaluation rules (weak head normalization, call by name)

(Grab)	$\lambda x\,.\,t$	$\star$	$u \cdot \pi$	$\succ$	$t\{x := u\}$	$\star$	π
(Push)	tu	$\star$	π	$\succ$	t	$\star$	$u \cdot \pi$
(Call/cc)	$\text{\ae}$	$\star$	$t \cdot \pi$	$\succ$	t	$\star$	$\mathsf{k}_\pi \cdot \pi$
(Resume)	k_π	$\star$	$t \cdot \pi'$	$\succ$	t	$\star$	π

... to classical realizability semantics

- Interpreting higher-order terms :
  - Individuals interpreted as natural numbers $\quad [\![\iota]\!] = \mathbb{N}$
  - Propositions interpreted as falsity values $\quad [\![o]\!] = \mathfrak{P}(\Pi)$
  - Functions interpreted set-theoretically $\quad [\![\tau \to \sigma]\!] = [\![\sigma]\!]^{[\![\tau]\!]}$
- Parameterized by a pole $\quad \perp\!\!\!\perp \subseteq \Lambda_c \star \Pi \quad$ (closed under anti-evaluation)
- Interpreting logical constructions :

$$[\![\forall x^\tau A]\!]_\rho \;=\; \bigcup_{e \in [\![\tau]\!]} [\![A]\!]_{\rho, x \leftarrow e} \qquad\qquad [\![A \Rightarrow B]\!]_\rho \;=\; [\![A]\!]_\rho^{\perp\!\!\!\perp} \cdot [\![B]\!]_\rho$$

$$[\![\langle M = M' \rangle A]\!]_\rho \;=\; \begin{cases} [\![A]\!]_\rho & \text{if } [\![M]\!]_\rho = [\![M']\!]_\rho \\ \varnothing & \text{otherwise} \end{cases}$$

Adequacy

If
- $\mathcal{E}; x_1 : A_1, \ldots, x_n : A_n \vdash t : B \qquad$ (in $\mathrm{PA}\omega^+$)
- $\rho \models \mathcal{E}, \quad u_1 \in [\![A_1]\!]_\rho^{\perp\!\!\!\perp}, \ \ldots, \ u_n \in [\![A_n]\!]_\rho^{\perp\!\!\!\perp}$

then : $\quad t\{x_1 := u_1; \ldots; x_n := u_n\} \in [\![B]\!]_\rho^{\perp\!\!\!\perp}$

Plan

Representing conditions

- **Intuition :** Represent the set of conditions as an upwards closed subset of a meet-semilattice
- Take :
  - A sort κ of conditions, equipped with
  - A binary product $(p, q) \mapsto pq$ (of sort $\kappa \to \kappa \to \kappa$)
  - A unit 1 (of sort κ)
  - A predicate $p \mapsto C[p]$ of well-formedness (of sort $\kappa \to o$)
- **Typical example :** finite functions from τ to σ are modelled by
  - $\kappa \equiv \tau \to \sigma \to o$ (binary relations $\subseteq \tau \times \sigma$)
  - $pq \equiv \lambda x^{\tau} y^{\sigma} . p\,x\,y \vee q\,x\,y$ (union of relations p and q)
  - $1 \equiv \lambda x^{\tau} y^{\sigma} . \bot$ (empty relation)
  - $C[p] \equiv$ "p is a finite function from τ to σ"

Combinators

- The forcing translation is parameterized by
  - The sort κ + closed terms $\cdot$, 1, C (logical level)
  - 9 closed proof terms $\alpha_*, \alpha_1, \ldots, \alpha_8$ (computational level)

$$\begin{array}{lcl}
\alpha_* & : & C[1] \\
\alpha_1 & : & \forall p^\kappa\ \forall q^\kappa\ (C[pq] \Rightarrow C[p]) \\
\alpha_2 & : & \forall p^\kappa\ \forall q^\kappa\ (C[pq] \Rightarrow C[q]) \\
\alpha_3 & : & \forall p^\kappa\ \forall q^\kappa\ (C[pq] \Rightarrow C[qp]) \\
\alpha_4 & : & \forall p^\kappa\ (C[p] \Rightarrow C[pp]) \\
\alpha_5 & : & \forall p^\kappa\ \forall q^\kappa\ \forall r^\kappa\ (C[(pq)r] \Rightarrow C[p(qr)]) \\
\alpha_6 & : & \forall p^\kappa\ \forall q^\kappa\ \forall r^\kappa\ (C[p(qr)] \Rightarrow C[(pq)r]) \\
\alpha_7 & : & \forall p^\kappa\ (C[p] \Rightarrow C[p1]) \\
\alpha_8 & : & \forall p^\kappa\ (C[p] \Rightarrow C[1p])
\end{array}$$

This set is not minimal. One can take $\alpha_*, \alpha_1, \alpha_3, \alpha_4, \alpha_5, \alpha_7$ and define :
$\alpha_2 := \alpha_1 \circ \alpha_3, \quad \alpha_6 := \alpha_3 \circ \alpha_5 \circ \alpha_3 \circ \alpha_5 \circ \alpha_3, \quad \alpha_8 := \alpha_3 \circ \alpha_7$

Derived combinators

- The combinators $\alpha_1, \ldots, \alpha_8$ can be composed :

  Example : $\alpha_1 \circ \alpha_6 \circ \alpha_3 \quad : \quad \forall p^\kappa\ \forall q^\kappa\ \forall r^\kappa\ (C[(pq)r] \Rightarrow C[rp])$

- We will also use the following derived combinators :

α_9	$:=$	$\alpha_3 \circ \alpha_1 \circ \alpha_6 \circ \alpha_3$	:	$\forall p^\kappa\ \forall q^\kappa\ \forall r^\kappa\ (C[(pq)r] \Rightarrow C[pr])$
α_{10}	$:=$	$\alpha_2 \circ \alpha_5$	:	$\forall p^\kappa\ \forall q^\kappa\ \forall r^\kappa\ (C[(pq)r] \Rightarrow C[qr])$
α_{11}	$:=$	$\alpha_9 \circ \alpha_4$	:	$\forall p^\kappa\ \forall q^\kappa\ (C[pq] \Rightarrow C[p(pq)])$
α_{12}	$:=$	$\alpha_5 \circ \alpha_3$	:	$\forall p^\kappa\ \forall q^\kappa\ \forall r^\kappa\ (C[p(qr)] \Rightarrow C[q(rp)])$
α_{13}	$:=$	$\alpha_3 \circ \alpha_{12}$	:	$\forall p^\kappa\ \forall q^\kappa\ \forall r^\kappa\ (C[p(qr)] \Rightarrow C[(rp)q])$
α_{14}	$:=$	$\alpha_5 \circ \alpha_3 \circ \alpha_{10} \circ \alpha_4 \circ \alpha_2$	:	$\forall p^\kappa\ \forall q^\kappa\ \forall r^\kappa\ (C[p(qr)] \Rightarrow C[q(rr)])$
α_{15}	$:=$	$\alpha_9 \circ \alpha_3$	:	$\forall p^\kappa\ \forall q^\kappa\ \forall r^\kappa\ (C[p(qr)] \Rightarrow C[qp])$

- **Important remark :**
  - $C[pq] \Rightarrow C[p] \wedge C[q]$, but $C[p] \wedge C[q] \not\Rightarrow C[pq]$ (in general)
  - Two conditions p and q are compatible when $C[pq]$

Ordering

- Let $p \leq q \;\; := \;\; \forall r^{\kappa}(C[pr] \Rightarrow C[qr])$
- $\leq$ is a preorder with greatest element 1 :

$$\begin{array}{lcl} \lambda c\,.\,c & : & \forall p^{\kappa}\ (p \leq p) \\ \lambda xyc\,.\,y(xc) & : & \forall p^{\kappa}\ \forall q^{\kappa}\ \forall r^{\kappa}\ (p \leq q \Rightarrow q \leq r \Rightarrow p \leq r) \\ \alpha_8 \circ \alpha_2 & : & \forall p^{\kappa}\ (p \leq 1) \end{array}$$

- Product pq is the l.u.b. of p and q :

$$\begin{array}{lcl} \alpha_9 & : & \forall p^{\kappa}\ \forall q^{\kappa}\ (pq \leq p) \\ \alpha_{10} & : & \forall p^{\kappa}\ \forall q^{\kappa}\ (pq \leq q) \\ \lambda xy\,.\,\alpha_{13} \circ y \circ \alpha_{12} \circ x \circ \alpha_{11} & : & \forall p^{\kappa}\ \forall q^{\kappa}\ \forall r^{\kappa}\ (r \leq p \Rightarrow r \leq q \Rightarrow r \leq pq) \end{array}$$

- C (set of 'good' conditions) is upwards closed :

$$\lambda xc\,.\,\alpha_1\,(x\,(\alpha_7\,c)) \quad : \quad \forall p^{\kappa}\ \forall q^{\kappa}\ (p \leq q \Rightarrow C[p] \Rightarrow C[q])$$

- Bad conditions are smallest elements :

$$\lambda xc\,.\,x\,(\alpha_1\,c) \quad : \quad \forall p^{\kappa}\ (\neg C[p] \Rightarrow \forall q^{\kappa}\ p \leq q)$$

The auxiliary translation $(\_)^*$

- Translating sorts : $\tau \mapsto \tau^*$

$$\iota^* \equiv \iota \qquad o^* \equiv \kappa \to o \qquad (\tau \to \sigma)^* \equiv \tau^* \to \sigma^*$$

Intuition : Propositions become sets of conditions

- Translating terms : $M \mapsto M^*$

$$\begin{array}{rcl} (x^\tau)^* & \equiv & x^{\tau^*} \\ (\lambda x^\tau . M)^* & \equiv & \lambda x^{\tau^*} . M^* \\ (MN)^* & \equiv & M^* N^* \end{array} \qquad \begin{array}{rcl} 0^* & \equiv & 0 \\ \mathsf{s}^* & \equiv & \mathsf{s} \\ \mathsf{rec}_\tau^* & \equiv & \mathsf{rec}_{\tau^*} \end{array}$$

$$\begin{array}{rcl} (A \Rightarrow B)^* & \equiv & \lambda r^\kappa . \forall q^\kappa \forall r'^\kappa \langle r = qr' \rangle (\forall s^\kappa (C[qs] \Rightarrow A^* s) \Rightarrow B^* r') \\ (\forall x^\tau A)^* & \equiv & \lambda r^\kappa . \forall x^{\tau^*} A^* r \\ (\langle M_1 = M_2 \rangle A)^* & \equiv & \lambda r^\kappa . \langle M_1^* = M_2^* \rangle (A^* r) \end{array}$$

Lemma

- $(M\{x^\tau := N\})^* \equiv M^*\{x^{\tau^*} := N^*\}$ (substitutivity)
- If $M_1 \cong_{\mathcal{E}} M_2$, then $M_1^* \cong_{\mathcal{E}^*} M_2^*$ (compatibility with conversion)

The forcing translation

- Given a proposition A and a condition p, let :

$$p \Vdash A \quad := \quad \forall r^{\kappa}(C[pr] \Rightarrow A^{*} r)$$

  - The forcing translation is trivial on $\forall$ and $\langle \_ = \_ \rangle \_$:

$$\begin{aligned} p \Vdash \forall x^{\tau} A \quad & \cong_{\varnothing} \quad \forall x^{\tau^{*}} (p \Vdash A) \\ p \Vdash \langle M_1 = M_2 \rangle A \quad & \cong_{\varnothing} \quad \langle M_1^{*} = M_2^{*} \rangle (p \Vdash A) \end{aligned}$$

  - All the complexity lies in implication ! (cf next slide)

General properties

$$\begin{aligned} \beta_1 \quad &:= \quad \lambda xyc\,.\,y\,(x\,c) &:& \quad \forall p^{\kappa}\ \forall q^{\kappa}\ (q \leq p \Rightarrow (p \Vdash A) \Rightarrow (q \Vdash A)) \\ \beta_2 \quad &:= \quad \lambda xc\,.\,x\,(\alpha_1\,c) &:& \quad \forall p^{\kappa}\ (\neg C[p] \Rightarrow p \Vdash A) \\ \beta_3 \quad &:= \quad \lambda xc\,.\,x\,(\alpha_9\,c) &:& \quad \forall p^{\kappa}\ \forall q^{\kappa}\ ((p \Vdash A) \Rightarrow (pq \Vdash A)) \\ \beta_4 \quad &:= \quad \lambda xc\,.\,x\,(\alpha_{10}\,c) &:& \quad \forall p^{\kappa}\ \forall q^{\kappa}\ ((q \Vdash A) \Rightarrow (pq \Vdash A)) \end{aligned}$$

Forcing an implication

- Definition of $p \Vdash A \Rightarrow B$ looks strange :

$$
\begin{aligned}
p \Vdash A \Rightarrow B \quad &\equiv \quad \forall r^{\kappa}(C[pr] \Rightarrow (A \Rightarrow B)^{*} r) \\
&\cong_{\varnothing} \quad \forall r^{\kappa}(C[pr] \Rightarrow \forall q^{\kappa} \forall r'^{\kappa} \langle r = qr' \rangle ((q \Vdash A) \Rightarrow B^{*} r'))
\end{aligned}
$$

- But it is equivalent to

$$
\forall q\,((q \Vdash A) \Rightarrow (pq \Vdash B)) \qquad \left(\textbf{Hint :} \quad \frac{p \Vdash A \Rightarrow B \qquad q \Vdash A}{pq \Vdash B}\right)
$$

Coercions between $p \Vdash A \Rightarrow B$ and $\forall q\,((q \Vdash A) \Rightarrow (pq \Vdash B))$

$$
\begin{aligned}
\gamma_1 &:= \lambda xcy\,.\,x\,y\,(\alpha_6\,c) &&: \; (\forall q\,((q \Vdash A) \Rightarrow (pq \Vdash B)) \;\Rightarrow\; p \Vdash A \Rightarrow B) \\
\gamma_2 &:= \lambda xyc\,.\,x\,(\alpha_5\,c)\,y &&: \; (p \Vdash A \Rightarrow B) \;\Rightarrow\; \forall q\,((q \Vdash A) \Rightarrow (pq \Vdash B)) \\
\gamma_3 &:= \lambda xyc\,.\,x\,(\alpha_{11}\,c)\,y &&: \; (p \Vdash A \Rightarrow B) \;\Rightarrow\; (p \Vdash A) \;\Rightarrow\; (p \Vdash B) \\
\gamma_4 &:= \lambda xcy\,.\,x\,(y\,(\alpha_{15}\,c)) &&: \; \neg A^{*}\,p \;\Rightarrow\; p \Vdash A \Rightarrow B
\end{aligned}
$$

Translating proof-terms

- Krivine's program transformation $t \mapsto t^*$:

$$x^* \equiv x \qquad \text{cc}^* \equiv \lambda cx\,.\,\text{cc}\,(\lambda k\,.\,x\,(\alpha_{14}\,c)\,(\gamma_4\,k)) \qquad \gamma_4 \equiv \lambda xcy\,.\,x\,(y\,(\alpha_{15}\,c))$$

$$(t\,u)^* \equiv \gamma_3\,t^*\,u^* \qquad \gamma_3 \equiv \lambda xyc\,.\,x\,(\alpha_{11}\,c)\,y$$

$$(\lambda x\,.\,t)^* \equiv \gamma_1\,(\lambda x\,.\,t^*\,\underbrace{\{x := \beta_4 x\}}_{\text{bounded var}}\,\underbrace{\{x_i := \beta_3 x_i\}_{i=1}^{n}}_{\text{other free vars of } t}) \qquad \gamma_1 \equiv \lambda xcy\,.\,x\,y\,(\alpha_6\,c)$$

  - The translation inserts
    - γ_1 ("fold") in front of every abstraction
    - γ_3 ("apply") in front of every application
  - A bound occurrence of x in t is translated as $\beta_3^n(\beta_4 x)$, where n is the de Bruijn index of this occurrence

Soundness (in PAω^+)

If $\mathcal{E};\ x_1 : A_1,\ \ldots,\ x_n : A_n \ \vdash\ t \ :\ B$

then $\mathcal{E}^*;\ x_1 : (p \Vdash A_1),\ \ldots,\ x_n : (p \Vdash A_n) \ \vdash\ t^* \ :\ (p \Vdash B)$

Computational meaning of the transformation

- A proof of $p \Vdash A \equiv \forall r^{\kappa}(C[pr] \Rightarrow A^* r)$ is a function waiting an argument $c : C[pr]$ (for some r) $\rightsquigarrow$ computational condition

$$\begin{array}{rcllcrcl}
(\lambda x\,.\,t)^* & \star & c \cdot u \cdot \pi & \succ & t^\dagger\{x := u\} & \star & \alpha_6\, c \cdot \pi \\
(tu)^* & \star & c \cdot \pi & \succ & t^* & \star & \alpha_{11}\, c \cdot u^* \cdot \pi \\
\text{cc}^* & \star & c \cdot t \cdot \pi & \succ & t & \star & \alpha_{14}\, c \cdot \mathsf{k}_\pi^* \cdot \pi \\
\mathsf{k}_\pi^* & \star & c \cdot t \cdot \pi' & \succ & t & \star & \alpha_{15}\, c \cdot \pi
\end{array}$$

where :

$$\begin{array}{rcl}
t^\dagger & \equiv & t^*\{x := \beta_4\, x\}\{x_i := \beta_3\, x_i\}_{i=1}^n \\
\mathsf{k}_\pi^* & \equiv & \gamma_4\, \mathsf{k}_\pi
\end{array}$$

Evaluation combinators

$$\begin{array}{rcrcl}
\alpha_6 & : & C[p(qr)] & \Rightarrow & C[(pq)r] \\
\alpha_{11} & : & C[pr] & \Rightarrow & C[p(pr)] \\
\alpha_{14} & : & C[p(qr)] & \Rightarrow & C[q(rr)] \\
\alpha_{15} & : & C[p(qr)] & \Rightarrow & C[qp]
\end{array}$$

Plan

Krivine Forcing Abstract Machine (KFAM)

Terms	t, u	$::=$	$x \mid \lambda x\,.\,t \mid tu \mid \text{œ}$
Environments	e	$::=$	$\emptyset \mid e, x = c$
Closures	c	$::=$	$(t\|e) \mid \mathsf{k}_\pi \mid \underbrace{(t\|e)^* \mid \mathsf{k}_\pi^*}_{\text{forcing closures}}$
Stacks	π	$::=$	$\diamond \mid c \cdot \pi$

- Real mode :

$$
\begin{array}{rcll}
(x|e, y = c) \star \pi & \succ & (x|e) \star \pi & (y \not\equiv x) \\
(x|e, x = c) \star \pi & \succ & c \star \pi & \\
(\lambda x\,.\,t|e) \star c \cdot \pi & \succ & (t|e, x = c) \star \pi & \\
(tu|e) \star \pi & \succ & (t|e) \star (u|e) \cdot \pi & \\
(\text{œ}|e) \star c \cdot \pi & \succ & c \star \mathsf{k}_\pi \cdot \pi & \\
\mathsf{k}_\pi \star c \cdot \pi' & \succ & c \star \pi &
\end{array}
$$

- Forcing mode :

$$
\begin{array}{rcll}
(x|e, y = c)^* \star c_0 \cdot \pi & \succ & (x|e)^* \star \alpha_9\, c_0 \cdot \pi & (y \not\equiv x) \\
(x|e, x = c)^* \star c_0 \cdot \pi & \succ & c \star \alpha_{10}\, c_0 \cdot \pi & \\
(\lambda x\,.\,t|e)^* \star c_0 \cdot c \cdot \pi & \succ & (t|e, x = c)^* \star \alpha_6\, c_0 \cdot \pi & \\
(tu|e)^* \star c_0 \cdot \pi & \succ & (t|e)^* \star \alpha_{11}\, c_0 \cdot (u|e)^* \cdot \pi & \\
(\text{œ}|e)^* \star c_0 \cdot c \cdot \pi & \succ & c \star \alpha_{14}\, c_0 \cdot \mathsf{k}_\pi^* \cdot \pi & \\
\mathsf{k}_\pi^* \star c_0 \cdot c \cdot \pi' & \succ & c \star \alpha_{15}\, c_0 \cdot \pi &
\end{array}
$$

Adequacy in real and forcing modes

- New abstract machine means :
  - New classical realizability model (based on the KFAM)
  - New adequacy results

Adequacy (real mode)

If
- $\mathcal{E}; x_1 : A_1, \ldots, x_n : A_n \vdash t : B$ (in $\mathrm{PA}\omega^+$)
- $\rho \models \mathcal{E}, \quad c_1 \in [\![A_1]\!]_\rho^{\perp\!\!\!\perp}, \ \ldots, \ c_n \in [\![A_n]\!]_\rho^{\perp\!\!\!\perp}$

then : $(t|x_1 = c_1, \ldots, x_n = c_n) \in [\![B]\!]_\rho^{\perp\!\!\!\perp}$

- Assuming that $\alpha_i \in [\![\text{type of } \alpha_i]\!]^{\perp\!\!\!\perp}$ (for $i = 6, 9, 10, 11, 14, 15$)

Adequacy (forcing mode)

If
- $\mathcal{E}; x_1 : A_1, \ldots, x_n : A_n \vdash t : B$ (in $\mathrm{PA}\omega^+$)
- $\rho \models \mathcal{E}^*, \quad c_1 \in [\![p_1 \Vdash A_1]\!]_\rho^{\perp\!\!\!\perp}, \ \ldots, \ c_n \in [\![p_n \Vdash A_n]\!]_\rho^{\perp\!\!\!\perp}$

then : $(t|x_1 = c_1, \ldots, x_n = c_n)^* \in [\![(p_0 p_1) \cdots p_n \Vdash B]\!]_\rho^{\perp\!\!\!\perp}$

Conclusion

Underlying methodology

Translation of formulas & proofs	$\rightsquigarrow$	Program transform	$\rightsquigarrow$	Computation model (transform becomes identity)

Example : Negative translation $\rightsquigarrow$ CPS transform $\rightsquigarrow$ stack based machine

- This methodology applies to the forcing translation
  - A new abstract machine : the KFAM
  - Reminiscent from well known tricks of computer architecture (protection rings, virtual memory, hardware tracing, ...)

1. How this computation model is used in particular cases of forcing ?
2. Use this methodology the other way around !
   - Deduce new logical translations from computation models borrowed to computer architecture, operating systems, ...