A computational analysis of the proof transformation by forcing

Alexandre Miquel
ENS de Lyon – LIP/Plume team

June 22th, 2011 – LICS'11
Fields Institute, Toronto

Introduction

- **What is forcing ?**
  - A technique invented by Cohen ('63) to prove the independence of continuum hypothesis (CH) w.r.t. ZFC
  - Cohen forcing can be understood as
    - A technique to transform models of ZFC, using generic sets
    - A translation of formulas and proofs (proof theorist's point of view)

Introduction

- **What is forcing ?**
  - A technique invented by Cohen ('63) to prove the independence of continuum hypothesis (CH) w.r.t. ZFC
  - Cohen forcing can be understood as
    - A technique to transform models of ZFC, using generic sets
    - A translation of formulas and proofs (proof theorist's point of view)
- **Curry-Howard correspondence in classical logic**
  - Classical reasoning principles as control operators [Griffin'90]

$$\mathsf{call/cc} \quad : \quad ((A \Rightarrow B) \Rightarrow A) \Rightarrow A$$

  - The theory of classical realizability [Krivine '01, '03, '09]
    - Complete reformulation of Kleene's realizability by hard-wiring Friedman's A-translation in the engine (and even more)
    - Works in expressive frameworks : PA2, PAω, ZF + DC

The big picture

- **Krivine's realizability interpretation of forcing** [Krivine '09, '10]
  - Introduces generalized realizability structures
  - Defines iterated forcing/realizability + case studies
  - Existence of an underlying program transformation...
- **Aims of the talk :**
  - Rephrase translation in PAω, using typing rather than realizability
  - Present program transformation + underlying computation model

The big picture

- **Krivine's realizability interpretation of forcing** [Krivine '09, '10]
  - Introduces generalized realizability structures
  - Defines iterated forcing/realizability + case studies
  - Existence of an underlying program transformation...
- **Aims of the talk :**
  - Rephrase translation in PAω, using typing rather than realizability
  - Present program transformation + underlying computation model
- **Underlying methodology :**

Translation of formulas & proofs	$\rightsquigarrow$	Program transform	$\rightsquigarrow$	Abstract machine (transform becomes identity)

Example : $\neg\neg$-translation $\rightsquigarrow$ CPS transform $\rightsquigarrow$ stack based machine

Plan

Plan

Higher-order arithmetic (tuned)

- System PAω^+ : a multi-sorted language to express
  - Individuals (kind ι)
  - Propositions (kind o)
  - Predicates over individuals ($\iota \to o$, $\iota \to \iota \to o$, ...)
  - Predicates over predicates... ($(\iota \to o) \to o$, ...)

Syntax of kinds and higher-order terms (a.k.a. constructors)

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

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

Proof terms $\quad$ (postponed)

- Proposition $\langle M = M' \rangle A$ is provably equivalent to $M =_\tau M' \Rightarrow A$ (where $=_\tau$ is Leibniz equality), but has more compact proof terms

The relation of conversion

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

$$\mathcal{E} \equiv \{M_1 = M'_1, \ldots, M_k = M'_k\} \qquad \text{(non oriented, well `kinded')}$$

- Base case : if $(M = M') \in \mathcal{E}$, then $M \cong_{\mathcal{E}} M'$
- Contains β-conversion, η-conversion, recursion (as usual)
- Many rules to identify semantically equivalent propositions :

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

$\rightsquigarrow$ Conversion emphasizes the Curry-style setting

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 kind o)

Deduction system (typing)

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

Deduction/typing rules

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

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

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

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

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

From operational semantics...

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

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

  - 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)

(Push)	tu	$\star$	π	$\succ$	t	$\star$	$u \cdot \pi$
(Grab)	$\lambda x\,.\,t$	$\star$	$u \cdot \pi$	$\succ$	$t\{x := u\}$	$\star$	π
(Save)	æ	$\star$	$t \cdot \pi$	$\succ$	t	$\star$	$\mathsf{k}_\pi \cdot \pi$
(Restore)	k_π	$\star$	$t \cdot \pi'$	$\succ$	t	$\star$	π

... to classical realizability semantics

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

... to classical realizability semantics

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

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

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

- **Notation :** $\quad t \Vdash A \;\equiv\; t \in [\![A]\!]^{\perp\!\!\!\perp} \;\equiv\; \forall \pi \in [\![A]\!]\ \ (t \star \pi) \in \perp\!\!\!\perp$
- **Adequacy :** $\quad$ If $\;\vdash t : A$, then $\; t \Vdash A$

Plan

Representing conditions

- **Intuition :** Represent the set of conditions (in system PAω^+) as an upwards closed subset of a meet-semilattice
- Take :
  - A kind κ of conditions, equipped with
  - A binary product $(p, q) \mapsto pq$ (kind $\kappa \to \kappa \to \kappa$)
  - A unit 1 (kind κ)
  - A predicate $p \mapsto C[p]$ of well-formedness (kind $\kappa \to o$)

Representing conditions

- **Intuition :** Represent the set of conditions (in system PAω^+) as an upwards closed subset of a meet-semilattice

- Take :
  - A kind κ of conditions, equipped with
  - A binary product $(p, q) \mapsto pq$ (kind $\kappa \to \kappa \to \kappa$)
  - A unit 1 (kind κ)
  - A predicate $p \mapsto C[p]$ of well-formedness (kind $\kappa \to o$)

- **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)
  - $1 \equiv \lambda x^\tau y^\sigma . \perp$ (empty relation)
  - $C[p] \equiv$ "p is a finite function from τ to σ"

Combinators

The forcing translation is parameterized by

- The kind κ + closed terms $\cdot$, 1, C (logical level)
- 9 closed proof terms $\alpha_*, \alpha_1, \ldots, \alpha_8$ (computational level)

Primitive combinators

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

Combinators

The forcing translation is parameterized by

- The kind κ + closed terms $\cdot$, 1, C (logical level)
- 9 closed proof terms $\alpha_*, \alpha_1, \ldots, \alpha_8$ (computational level)

Primitive combinators

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

Derived combinators (from $\alpha_*, \alpha_1, \ldots, \alpha_8$)

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

Ordering

Let $\quad p \le q \quad := \quad \forall r^\kappa (C[pr] \Rightarrow C[qr])$

- $\le$ is a preorder with greatest element 1 :

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

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

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

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

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

- Bad conditions are smallest elements :

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

The forcing translation in PAω^+ : logical level

Translating kinds : $\tau \mapsto \tau^*$

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

Intuition : Propositions become sets of conditions

Auxiliary translation on HO terms of all kinds : $M : \tau \;\mapsto\; M^* : \tau^*$

$$\begin{array}{rcl}
(x^\tau)^* & \equiv & x^{\tau^*} \\
(\lambda x^\tau . M)^* & \equiv & \lambda x^{\tau^*} . M^* \\
(MN)^* & \equiv & M^* N^* \\
\ldots & & \ldots \\
(\forall x^\tau A)^* & \equiv & \lambda r^\kappa . \forall x^{\tau^*} A^* r \\
(\langle M_1 = M_2 \rangle A)^* & \equiv & \lambda r^\kappa . \langle M_1^* = M_2^* \rangle A^* r \\
(A \Rightarrow B)^* & \equiv & \lambda r^\kappa . \forall q^\kappa \forall r'^\kappa \langle r = qr' \rangle ((q \Vdash A) \Rightarrow B^* r')
\end{array}$$

Forcing translation on propositions : ($p : \kappa$ fixed condition)

$$p \Vdash A \quad \equiv \quad \forall r^\kappa (C[pr] \Rightarrow A^* r)$$

Properties of the forcing translation

General properties

$$\begin{array}{llll} \beta_1 & := & \lambda xyc\,.\,y\,(x\,c) & : \quad \forall p^\kappa\ \forall q^\kappa\ (q \le p \Rightarrow (p \Vdash A) \Rightarrow (q \Vdash A)) \\ \beta_2 & := & \lambda xc\,.\,x\,(\alpha_1\,c) & : \quad \forall p^\kappa\ (\neg C[p] \Rightarrow p \Vdash A) \end{array}$$

Properties of the forcing translation

General properties

$$
\begin{array}{lcccl}
\beta_1 & := & \lambda xyc\,.\,y\,(x\,c) & : & \forall p^\kappa\ \forall q^\kappa\ (q \le p \Rightarrow (p \Vdash A) \Rightarrow (q \Vdash A)) \\
\beta_2 & := & \lambda xc\,.\,x\,(\alpha_1\ c) & : & \forall p^\kappa\ (\neg C[p] \Rightarrow p \Vdash A)
\end{array}
$$

Forcing logical constructions (computationally irrelevant)

$$
\begin{array}{rcl}
p \Vdash \langle M = M' \rangle A & \cong & \langle M^* = M'^* \rangle (p \Vdash A) \\
p \Vdash \forall x^\tau A & \cong & \forall x^{\tau^*} (p \Vdash A)
\end{array}
$$

Properties of the forcing translation

General properties

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

Forcing logical constructions (computationally irrelevant)

$$p \Vdash \langle M = M' \rangle A \quad \cong \quad \langle M^* = M'^* \rangle (p \Vdash A)$$

$$p \Vdash \forall x^\tau A \quad \cong \quad \forall x^{\tau^*} (p \Vdash A)$$

Forcing an implication (computationally relevant)

$$p \Vdash A \Rightarrow B \quad \Leftrightarrow \quad \forall q^\kappa\ ((q \Vdash A) \Rightarrow (pq \Vdash B))$$

since :

$$\begin{array}{llll} \gamma_1 & := & \lambda xcy\,.\,x\,y\,(\alpha_6\ c) & : \quad \forall q\,((q \Vdash A) \Rightarrow (pq \Vdash B)) \Rightarrow (p \Vdash A \Rightarrow B) \\ \gamma_2 & := & \lambda xyc\,.\,x\,(\alpha_5\ c)\,y & : \quad (p \Vdash A \Rightarrow B) \Rightarrow \forall q\ ((q \Vdash A) \Rightarrow (pq \Vdash B)) \end{array}$$

The forcing translation in PAω^+ : level of proof terms

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

$$
\begin{aligned}
x^* &\equiv x & \text{æ}^* &\equiv \lambda cx\,.\,\text{æ}\,(\lambda k\,.\,x\,(\alpha_{14}\ c)\,(\gamma_4\ k)) \\
(t\ u)^* &\equiv \gamma_3\ t^*\ u^* \\
(\lambda x\,.\,t)^* &\equiv \gamma_1\,(\lambda x\,.\,t^*\,\underbrace{\{x := \beta_4 x\}}_{\text{bounded var}}\,\underbrace{\{x_i := \beta_3 x_i\}_{i=1}^{n}}_{\text{other free vars of } t})
\end{aligned}
$$

$\gamma_4 \equiv \lambda xcy\,.\,x\,(y\,(\alpha_{15}\ c))$

$\gamma_3 \equiv \lambda xyc\,.\,x\,(\alpha_{11}\ c)\ y$

$\gamma_1 \equiv \lambda xcy\,.\,x\ y\,(\alpha_6\ c)$

$\beta_3 \equiv \lambda xc\,.\,x\,(\alpha_9\ c)$

$\beta_4 \equiv \lambda xc\,.\,x\,(\alpha_{10}\ c)$

- The translation inserts : γ_1 ("fold") in front of each λ
  γ_3 ("apply") in front of each app.
- 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

The forcing translation in PAω^+ : level of proof terms

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

$$
\begin{aligned}
x^* &\equiv x & \text{œ}^* &\equiv \lambda cx \,.\, \text{œ}\,(\lambda k \,.\, x\,(\alpha_{14}\; c)\,(\gamma_4\; k))\\
(t\,u)^* &\equiv \gamma_3\; t^*\; u^* \\
(\lambda x\,.\,t)^* &\equiv \gamma_1\,(\lambda x\,.\,t^*\,\underbrace{\{x := \beta_4 x\}}_{\text{bounded var}}\,\underbrace{\{x_i := \beta_3 x_i\}_{i=1}^{n}}_{\text{other free vars of } t})
\end{aligned}
$$

$\gamma_4 \equiv \lambda xcy \,.\, x\,(y\,(\alpha_{15}\; c))$

$\gamma_3 \equiv \lambda xyc \,.\, x\,(\alpha_{11}\; c)\; y$

$\gamma_1 \equiv \lambda xcy \,.\, x\; y\,(\alpha_6\; c)$

$\beta_3 \equiv \lambda xc \,.\, x\,(\alpha_9\; c)$

$\beta_4 \equiv \lambda xc \,.\, x\,(\alpha_{10}\; c)$

- The translation inserts : γ_1 ("fold") in front of each λ
  γ_3 ("apply") in front of each app.
- 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 $\quad \mathcal{E};\; x_1 : A_1,\; \ldots,\; x_n : A_n \;\vdash\; t \;:\; B$

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

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

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

$$(\lambda x\,.\,t)^* \quad \star \quad c \cdot u \cdot \pi$$

$$(tu)^* \quad \star \quad c \cdot \pi$$

$$\text{cc}^* \quad \star \quad c \cdot u \cdot \pi$$

Evaluation combinators

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

$$\begin{array}{rcl} (\lambda x\,.\,t)^* \star c \cdot u \cdot \pi & \succ & t^*\{x := \beta_4 u\} \star \alpha_6\, c \cdot \pi \\ (tu)^* \star c \cdot \pi & & \\ \text{œ}^* \star c \cdot u \cdot \pi & & \end{array}$$

Evaluation combinators

$$\alpha_6 \;:\; C[p(qr)] \;\Rightarrow\; C[(pq)r]$$

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

$$
\begin{array}{rcl}
(\lambda x\,.\,t)^* \star c \cdot u \cdot \pi & \succ & t^*\{x := \beta_4 u\} \star \alpha_6\, c \cdot \pi \\
(tu)^* \star c \cdot \pi & \succ & t^* \star \alpha_{11}\, c \cdot u^* \cdot \pi \\
\text{œ}^* \star c \cdot u \cdot \pi & &
\end{array}
$$

Evaluation combinators

$$
\begin{array}{rcccl}
\alpha_6 & : & C[p(qr)] & \Rightarrow & C[(pq)r] \\
\alpha_{11} & : & C[pr] & \Rightarrow & C[p(pr)]
\end{array}
$$

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

$$
\begin{array}{rclcrcl}
(\lambda x\,.\,t)^* & \star & c \cdot u \cdot \pi & \succ & t^*\{x := \beta_4 u\} & \star & \alpha_6\, c \cdot \pi \\
(tu)^* & \star & c \cdot \pi & \succ & t^* & \star & \alpha_{11}\, c \cdot u^* \cdot \pi \\
\text{œ}^* & \star & c \cdot u \cdot \pi & \succ & u & \star & \alpha_{14}\, c \cdot \mathrm{k}_\pi^* \cdot \pi
\end{array}
$$

where : $\mathrm{k}_\pi^* \;\equiv\; \gamma_4\,\mathrm{k}_\pi \quad (\approx \lambda cx\,.\,\mathrm{k}_\pi\,(x\,(\alpha_{15}\,c)))$

Evaluation combinators

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

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

$$
\begin{array}{rclcrcl}
(\lambda x\,.\,t)^* & \star & c \cdot u \cdot \pi & \succ & t^*\{x := \beta_4 u\} & \star & \alpha_6\, c \cdot \pi \\
(tu)^* & \star & c \cdot \pi & \succ & t^* & \star & \alpha_{11}\, c \cdot u^* \cdot \pi \\
\text{cc}^* & \star & c \cdot u \cdot \pi & \succ & u & \star & \alpha_{14}\, c \cdot \mathrm{k}^*_\pi \cdot \pi \\
\mathrm{k}^*_\pi & \star & c \cdot t \cdot \pi' & & & &
\end{array}
$$

where : $\mathrm{k}^*_\pi \;\equiv\; \gamma_4\, \mathrm{k}_\pi \quad (\approx \lambda cx\,.\,\mathrm{k}_\pi\,(x\,(\alpha_{15}\, c)))$

Evaluation combinators

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

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

$$
\begin{array}{rcllcrcl}
(\lambda x\,.\,t)^* & \star & c \cdot u \cdot \pi & \succ & t^*\{x := \beta_4 u\} & \star & \alpha_6\, c \cdot \pi \\
(tu)^* & \star & c \cdot \pi & \succ & t^* & \star & \alpha_{11}\, c \cdot u^* \cdot \pi \\
\text{œ}^* & \star & c \cdot u \cdot \pi & \succ & u & \star & \alpha_{14}\, c \cdot \mathrm{k}_\pi^* \cdot \pi \\
\mathrm{k}_\pi^* & \star & c \cdot t \cdot \pi' & \succ & t & \star & \alpha_{15}\, c \cdot \pi
\end{array}
$$

where : $\mathrm{k}_\pi^* \;\equiv\; \gamma_4\, \mathrm{k}_\pi \quad (\approx \lambda cx\,.\,\mathrm{k}_\pi\,(x\,(\alpha_{15}\, c)))$

Evaluation combinators

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

Plan

Krivine Abstract Machine (KAM)

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

- Real mode :

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

Krivine Forcing Abstract Machine (KFAM)

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

- Real mode :

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

Krivine Forcing Abstract Machine (KFAM)

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

- Real mode :

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

- Forcing mode :

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

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	$::=$	$(t\|e) \mid \mathrm{k}_\pi \mid \underbrace{(t\|e)^* \mid \mathrm{k}_\pi^*}_{\text{forcing closures}}$
Stacks	π	$::=$	$\diamond \mid c \cdot \pi$

- Real mode :

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

- Forcing mode :

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

Extracting programs from proofs by forcing

- New abstract machine (KFAM) means new realizability algebras, new realizability models and new adequacy results

- Two adequacy results for the KFAM :
  - Adequacy in real mode $\vdash t : A \quad \rightsquigarrow \quad (t|\emptyset) \Vdash A$
  - Adequacy in forcing mode $\vdash t : A \quad \rightsquigarrow \quad (t|\emptyset)^* \Vdash (p \Vdash A)$

Extracting programs from proofs by forcing

- New abstract machine (KFAM) means new realizability algebras, new realizability models and new adequacy results

- Two adequacy results for the KFAM :
  - Adequacy in real mode $\vdash t : A \quad \rightsquigarrow \quad (t|\emptyset) \Vdash A$
  - Adequacy in forcing mode $\vdash t : A \quad \rightsquigarrow \quad (t|\emptyset)^* \Vdash (p \Vdash A)$

Extracting programs from proofs 'by forcing'

Given two proof-terms u ('user') and s ('system') such that :

$$x : A \vdash u : B \qquad \text{and} \qquad \vdash s : (1 \Vdash A)$$

we get :

$$(u \mid x = (s|\emptyset))^*$$

Extracting programs from proofs by forcing

- New abstract machine (KFAM) means new realizability algebras, new realizability models and new adequacy results
- Two adequacy results for the KFAM :
  - Adequacy in real mode $\vdash t : A \quad \rightsquigarrow \quad (t|\emptyset) \Vdash A$
  - Adequacy in forcing mode $\vdash t : A \quad \rightsquigarrow \quad (t|\emptyset)^* \Vdash (p \Vdash A)$

Extracting programs from proofs 'by forcing'

Given two proof-terms u ('user') and s ('system') such that :

$$x : A \vdash u : B \qquad \text{and} \qquad \vdash s : (1 \Vdash A)$$

we get :

$$(u \mid x = (s|\emptyset))^* \quad \Vdash_{\text{real}} \quad (1 \Vdash B)$$

Extracting programs from proofs by forcing

- New abstract machine (KFAM) means new realizability algebras, new realizability models and new adequacy results
- Two adequacy results for the KFAM :
  - Adequacy in real mode $\vdash t : A \quad \rightsquigarrow \quad (t|\emptyset) \Vdash A$
  - Adequacy in forcing mode $\vdash t : A \quad \rightsquigarrow \quad (t|\emptyset)^* \Vdash (p \Vdash A)$

Extracting programs from proofs 'by forcing'

Given two proof-terms u ('user') and s ('system') such that :

$$x : A \vdash u : B \qquad \text{and} \qquad \vdash s : (1 \Vdash A)$$

we get :

$$(u \mid x = (s|\emptyset))^* \quad \Vdash_{\text{real}} \quad (1 \Vdash B)$$

$$((u|x = (s|\emptyset))^*,\ 1) \quad \Vdash_{\text{forcing}} \quad B$$

Conclusion

Underlying methodology

$$\boxed{\begin{array}{c}\text{Translation of}\\\text{formulas \& proofs}\end{array}} \rightsquigarrow \boxed{\begin{array}{c}\text{Program}\\\text{transform}\end{array}} \rightsquigarrow \boxed{\begin{array}{c}\text{Abstract machine}\\\text{(transform becomes identity)}\end{array}}$$

- 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, ...)
  - Computational condition treated as a reference (forcing mode)

Conclusion

Underlying methodology

Translation of formulas & proofs	$\rightsquigarrow$	Program transform	$\rightsquigarrow$	Abstract machine (transform becomes identity)

- 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, ...)
  - Computational condition treated as a reference (forcing mode)

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, ...