Recursion in the Call-by-Value Lambda-Calculus

Gérard Boudol and Pascal Zimmer

INRIA Sophia Antipolis – France

FICS'02 – July 21st, 2002

Motivation

Recursive non-functional values in a call-by-value language:

- mutually recursive modules,
- objects as recursive records, e.g.

$$\begin{aligned}\textsf{let rec}\ \mathit{self} &= \{ \mathit{pos} = \textsf{ref}\, 0, \\ &\qquad \mathit{move} = \lambda d(\mathit{self}.\mathit{pos} := \,!\mathit{self}.\mathit{pos} + d)\} \\ \textsf{in}\ \ldots\end{aligned}$$

In call-by-value languages, recursion is restricted to:

$$\textsf{let rec}\ f = \lambda x N\ \textsf{in}\ M$$

Why?

Contents

The Call-By-Value λ-Calculus

[Landin 64, Plotkin 75, Felleisen 86]

Syntax:

$$\begin{array}{rcl} M,\ N\ldots & ::= & V \mid (MN) \mid \cdots \\ V & ::= & x \mid \lambda x M \mid \cdots \end{array}$$

Evaluation contexts:

$$\mathbf{E} ::= [\cdot] \mid (\mathbf{E}N) \mid (V\mathbf{E})$$

Reduction:

$$(\lambda x M V) \rightarrow \{x \mapsto V\}M$$

$$M \rightarrow M' \Rightarrow \mathbf{E}[M] \rightarrow \mathbf{E}[M']$$

The Abstract Machine

Configurations:

$$\mathcal{M} = (\mathit{Stack}, \mathit{Environment}, \mathit{Code})$$

Control stack $\sim$ evaluation context

Environment $\sim$ substitution ($\rightsquigarrow$ closures)

$$\begin{array}{rcll} S & ::= & \varepsilon \mid S :: \mathbf{F} & \textit{stacks of} \\ \mathbf{F} & ::= & ([\cdot]\sigma M) \mid (\sigma A[\cdot]) & \textit{frames} \\ \sigma,\ \rho \ldots & ::= & \varepsilon \mid \sigma :: \{x \mapsto \rho A\} & \textit{environments} \\ A & ::= & \lambda x M & \textit{values} \end{array}$$

The Machine: Transitions

Application:

$$
\begin{aligned}
(S, \sigma, (M\,N)) &\;\rightarrow\; (S :: ([\cdot]\sigma N), \sigma, M) \\
(S :: ([\cdot]\rho N), \sigma, A) &\;\rightarrow\; (S :: (\sigma A[\cdot]), \rho, N) \\
(S :: (\sigma \lambda x M[\cdot]), \rho, A) &\;\rightarrow\; (S, \sigma :: \{x \mapsto \rho A\}, M)
\end{aligned}
$$

Variables:

$$
\begin{aligned}
(S, \sigma :: \{y \mapsto Z\}, x) &\;\rightarrow\; (S, \sigma, x) \quad x \neq y \\
(S, \sigma :: \{x \mapsto \rho A\}, x) &\;\rightarrow\; (S, \rho, A)
\end{aligned}
$$

Recursion

$$
\begin{array}{rcl}
M,\ N \ldots & ::= & \cdots \mid \mu x M \\
\mathbf{E} & ::= & \cdots \mid \mu x \mathbf{E} \\
\mu x V & \rightarrow & \{x \mapsto \mu x V\} V
\end{array}
$$

Landin: *"to compute $\mu x M$, allocate a new address ℓ, evaluate M in an environment where x is bound to ℓ, and assign to ℓ the result of this computation."*

Extended configurations:

$$\mathcal{M} = (\mathit{Stack}, \mathit{Environment}, \mathit{Code}, \mathit{Store})$$

Store: mapping from locations to values

$$\xi \ ::= \ \emptyset \mid \xi \uplus \{\ell \mapsto ?\} \mid \xi[\ell := \sigma A]$$

New stack frames and environments:

$$\begin{array}{rcl} \mathbf{F} & ::= & \cdots \mid \mu\ell[\cdot] \\ \sigma,\ \rho\ldots & ::= & \cdots \mid \sigma :: \{\mathsf{rec}\, x \mapsto A\} \mid \sigma :: \{x \mapsto \ell\} \end{array}$$

Transitions: as before (with an invariant store), plus

Recursion:

$$(S, \sigma, \mu x A, \xi) \to (S, \sigma :: \{\mathsf{rec}\, x \mapsto A\}, A, \xi)$$

$$(S, \sigma, \mu x M, \xi) \to (S :: \mu\ell[\cdot], \sigma :: \{x \mapsto \ell\}, M, \xi \uplus \{\ell \mapsto ?\})$$

$$(S :: \mu\ell[\cdot], \sigma, A, \xi) \to (S, \sigma, A, \xi[\ell := \sigma A])$$

Variables:

$$(S, \sigma :: \{\mathsf{rec}\, x \mapsto A\}, x, \xi) \to (S, \sigma :: \{\mathsf{rec}\, x \mapsto A\}, A, \xi)$$

$$(S, \sigma :: \{x \mapsto \ell\}, x, \xi) \to (S, \rho, A, \xi) \qquad \text{if} \quad \xi(\ell) = \rho A$$

WHAT is WRONG ?

Unrestricted recursion is unsafe:

$$(S, \sigma :: \{x \mapsto \ell\}, x, \xi)$$

raises a run-time error if $\xi(\ell) = ?$. This occurs with

$\mu z((\lambda x \lambda y\, y) z)$	should *evaluate to* $\lambda y\, y$
$\mu z((\lambda x \lambda y\, y)(z(\lambda x\, x)))$	*should* not *evaluate*

$\Rightarrow$ syntactic restriction $\mu f \lambda x M$, or more generally $\mu y A$ (no need for the store), or a more subtle static analysis (and extension of the machine).

Typing Safe Recursion

There is a type system [Boudol 01] such that:

- some "interesting" non-standard recursions (for objects and modules) are typable,
- typability is decidable (with inference of a principal type),
- typability guarantees safety: reduction of typable expressions never gets stuck. In particular, recursive variables are only "passed around":

Key Lemma. *If* $\mu x M$ *is typable and* $M \xrightarrow{*} \mathbf{E}[x]$ *then* $\mathbf{E} = \mathbf{E}'[(\lambda y N[\cdot])]$.

The Revised Machine

Transitions: as before, plus, in case $\xi(\ell) = ?$

$$(S :: (\sigma\lambda y M[\cdot]), \rho :: \{x \mapsto \ell\}, x, \xi) \;\rightarrow\; (S, \sigma :: \{y \mapsto \ell\}, M, \xi)$$

Correctness

$\mathcal{E}val$ = the partial evaluation function determined by the machine.

THEOREM. *Let M be a closed expression.*

- *If $\mathcal{E}val(M)$ is defined then $M \xrightarrow{*} \mathcal{E}val(M)$,*
- *If M is typable and $M \xrightarrow{*} A$ then $\mathcal{E}val(M)$ is defined and $\mathcal{E}val(M) = A$.*

$$
\begin{array}{ccccccccccccc}
\mathcal{L}(M) & \rightarrow & \mathcal{M}_1 & \rightarrow & \mathcal{M}_2 & \rightarrow & \ldots & \rightarrow & \mathcal{M}_i & \rightarrow & \ldots & \rightarrow & \mathcal{M}_k \\
 & & & & & & & & & & & & \downarrow \mathcal{E}val \\
M & = & [\![\mathcal{M}_1]\!] & = & [\![\mathcal{M}_2]\!] & & \rightarrow & & [\![\mathcal{M}_i]\!] & & \rightarrow & & A
\end{array}
$$

Conclusion and Future Work

What we did:

- an abstract machine for unrestricted (typable) recursion, and especially objects as recursive records
- a proof of its correctness

What is next:

- a variation based on de Bruijn's indices
- an efficient implementation (type checker already exists)

Types and "safeness":

$$\begin{array}{rcl} \tau,\ \theta \ldots & ::= & * \mid (\theta^{\alpha} \to \tau) \\ \alpha,\ \beta \ldots & ::= & 0 \mid 1 \mid (\alpha \wedge \beta) \end{array}$$

with $* = (*^0 \to *)$

Typing rules:

$$\frac{\Gamma^{\gamma} \vdash M : \tau \quad \delta \leqslant \gamma}{\Gamma^{\delta} \vdash M : \tau} \qquad \frac{}{\Gamma^{1}, x : \tau^{0} \vdash x : \tau} \qquad \frac{\Gamma^{\gamma}, x : \theta^{\alpha} \vdash M : \tau}{\Gamma^{1} \vdash \lambda x M : (\theta^{\alpha} \to \tau)}$$

$$\frac{\Gamma^{\gamma} \vdash M : \theta^{\alpha} \to \tau \quad \Gamma^{\gamma} \vdash N : \theta}{\Gamma^{0_M \wedge \delta} \vdash (MN) : \tau}\,(*) \qquad \frac{\Gamma^{\gamma}, x : \tau^{1} \vdash M : \tau}{\Gamma^{\gamma} \vdash \mu x M : \tau}$$

$(*)$ $\quad \delta(x) = \text{if } N = x \text{ then } \alpha \text{ else } (\alpha_N \wedge \gamma)(x)$