# Recursion in the Call-by-Value Lambda-Calculus

Gérard Boudol and Pascal Zimmer

INRIA Sophia Antipolis

# Motivation

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

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

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

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 $\lambda$-calculus
- the abstract machine
- adding recursion
- what is wrong?
- solution: a new typing
- the revised machine

# The Call-By-Value $\lambda$-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:

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

Control stack $\sim$ evaluation context

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

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

# The Machine: Transitions

## Application:

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

## Variables:

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

# Recursion

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

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

Extended configurations:

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

Store: mapping from locations to values

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

New stack frames and environments:

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

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

Recursion:

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

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

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

Variables:

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

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

# 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

| | |
|---|---|
| $\mu z.(\lambda xy.y)z$ | should *evaluate to* $\lambda y\, y$ |
| $\mu z.(\lambda xy.y)(z(\lambda x\, x))$ | *should* not *evaluate* |

$\Rightarrow$ syntactic restriction $\mu f \lambda x M$, or more generally $\mu y V$ (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 $\xi(\ell) = ?$

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

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

THEOREM (CORRECTNESS). *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 $M \xrightarrow{*} V$ then $\mathcal{E}val(M)$ is defined and $\mathcal{E}val(M) = V$.*

## 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:

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

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

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