Defunctionalized Interpreters for Call-by-Need Programming Languages

— a functional pearl with hygiene —

Olivier Danvy (Aarhus University)

Kevin Millikin (Google)

Johan Munk (Arctic Lake Systems)

IFIP WG 2.8 12 June 2009

The contributions

- A hygienic standard call-by-need reduction for the λ-calculus.
- The notion of explicit evaluation contexts.
- Towards an abstract machine and a natural semantics for call by need through refocusing, refunctionalization, etc.

The starting point

The standard call-by-need reduction of

- Ariola and Felleisen, 1997
  JFP 7(3):265-301

- Maraist, Odersky and Wadler, 1998
  JFP 8(3):275-317

The starting point

The standard call-by-need reduction of

- Ariola and Felleisen, 1997
  JFP 7(3):265-301

- Maraist, Odersky and Wadler, 1998
  JFP 8(3):275-317

The goal: to extract a computational content.

Syntax

$$T ::= x \;\mid\; \lambda x.T \;\mid\; T\;T \;\mid\; \text{let } x \text{ be } T \text{ in } T$$

$$A ::= \lambda x.T \;\mid\; \text{let } x \text{ be } T \text{ in } A$$

$$E ::= [\;] \;\mid\; E\;T \;\mid$$

$$\text{let } x \text{ be } T \text{ in } E \;\mid$$

$$\text{let } x \text{ be } E \text{ in } E[x]$$

Axioms

$$
\begin{array}{rcl}
(\lambda x.T)\ T_1 & \longrightarrow & \text{let } x \text{ be } T_1 \text{ in } T \\
\text{let } x \text{ be } \lambda x.T \text{ in } E[x] & \longrightarrow & \text{let } x \text{ be } \lambda x.T \text{ in } E[\lambda x.T] \\
(\text{let } x \text{ be } T_1 \text{ in } A)\ T_2 & \longrightarrow & \text{let } x \text{ be } T_1 \text{ in } A\ T_2 \\
\text{let } x_2 \text{ be let } x_1 \text{ be } T & \longrightarrow & \text{let } x_1 \text{ be } T \\
\text{in } A & & \text{in let } x_2 \text{ be } A \\
\text{in } E[x_2] & & \text{in } E[x_2]
\end{array}
$$

In practice

...too hard to test!

La même chose, with integers

Syntax:

$$T ::= \ulcorner n \urcorner \mid \mathsf{succ}\ T \mid x \mid \ldots$$

$$A ::= \ulcorner n \urcorner \mid \lambda x.T \mid \text{let } x \text{ be } T \text{ in } A$$

$$E ::= [\,] \mid \mathsf{succ}\ E \mid E\ T \mid \ldots$$

La même chose, with integers

Three extra axioms:

$$\begin{array}{rcl} succ\,\ulcorner n \urcorner & \to & \ulcorner n' \urcorner \\ & & \text{where } n' = n + 1 \\ \text{let } x \text{ be } \ulcorner n \urcorner \text{ in } E[x] & \to & \text{let } x \text{ be } \ulcorner n \urcorner \text{ in } E[\ulcorner n \urcorner] \\ succ\,(\text{let } x \text{ be } T \text{ in } A) & \to & \text{let } x \text{ be } T \text{ in } succ\,A \end{array}$$

Some exegesis

1. The potential redexes

2. Barendregt's variable convention

3. The evaluation contexts

1. The potential redexes

A helpful grammar:

$$R ::= succ\ A \ \mid\ A\ T \ \mid\ \text{let } x \text{ be } A \text{ in } E[x]$$

where

$$A ::= \ulcorner n \urcorner \ \mid\ \lambda x.T \ \mid\ \text{let } x \text{ be } T \text{ in } A$$

2. Barendregt's variable convention (1/3)

It is assumed, e.g., in

$$(\text{let } x \text{ be } T_1 \text{ in } A)\ T_2 \ \longrightarrow\ \text{let } x \text{ be } T_1 \text{ in } A\ T_2$$

$$\text{let } x \text{ be } \lambda x.T \text{ in } E[x] \ \longrightarrow\ \text{let } x \text{ be } \lambda x.T \text{ in } E[\lambda x.T]$$

2. Barendregt's variable convention (2/3)

One axiom, however, yields terms
that do not satisfy the convention:

$$\text{let } x \text{ be } \lambda x.T \text{ in } E[x] \longrightarrow \text{let } x \text{ be } \lambda x.T \text{ in } E[\lambda x.T]$$

2. Barendregt's variable convention (3/3)

Simple fix:

$$\text{let } x \text{ be } \lambda x.T \longrightarrow \text{let } x \text{ be } \lambda x.T$$
$$\text{in } E[x] \qquad\qquad\qquad \text{in } E[\lambda x'.T']$$

$$\text{where } \lambda x'.T' = \mathit{freshen\_up}(\lambda x.T)$$

3. The evaluation contexts

The grammar of contexts is unusual

because

it includes identifiers within (delimited) contexts.

3. The evaluation contexts

The grammar of contexts is unusual

because

it includes identifiers within (delimited) contexts.

- These contexts are constructed <u>outside in</u>.
- All the others are constructed <u>inside out</u>.

Towards explicit evaluation contexts

Analogy with explicit substitutions:

delay the actual substitution.

Here: delay the recomposition, i.e.,

keep E instead of having $\lambda x.E[x]$.

Joint work with Kristoffer Rose

Contexts as lists of frames

$$\begin{aligned} F \quad ::= \; & succ\,\Box \;\; | \\ & \Box\, T \;\; | \\ & \text{let } x \text{ be } \Box \text{ in } C_{oi}[x] \;\; | \\ & \text{let } x \text{ be } T \text{ in } \Box \end{aligned}$$

$$C_{oi} ::= \bullet \;\;|\;\; F \circ C_{oi}$$

$$C_{io} ::= \bullet \;\;|\;\; F \circ C_{io}$$

Recomposition of outside-in contexts

$$\frac{}{\langle \bullet,\ T\rangle_{oi} \uparrow_{rec} T}$$

$$\frac{\langle C_{oi},\ T\rangle_{oi} \uparrow_{rec} T_0}{\langle (\Box\ T_1) \circ C_{oi},\ T\rangle_{oi} \uparrow_{rec} T_0\ T_1}$$

...

Recomposition of inside-out contexts

$$\frac{}{\langle \bullet,\ T \rangle_{io} \uparrow_{rec} T}$$

$$\frac{}{\langle (\Box\ T_1) \circ C_{io},\ T \rangle_{io} \uparrow_{rec} \langle C_{io},\ T\ T_1 \rangle_{io}}$$

...

Decomposition

A convenient format: as a transition system.

Accepting states: $\langle T, C_{io} \rangle_{\mathit{term}}$

$$\langle C_{io}, A \rangle_{\mathit{context}}$$

$$\langle C_{io}, (C_{oi}, x) \rangle_{\mathit{reroot}}$$

Final states: $\langle A \rangle_{\mathit{answer}}$

$$\langle R, C_{io} \rangle_{\mathit{decomposition}}$$

One-step reduction

$$T \mapsto_{let} T' \quad \text{if} \quad \begin{cases} \langle T, \bullet \rangle_{term} \downarrow^{*}_{dec} \langle R, C_{io} \rangle_{decomposition} \\ (R, R') \in \text{...the axioms...} \\ \langle C_{io}, R' \rangle_{io} \uparrow^{*}_{rec} T' \end{cases}$$

Reduction-based evaluation

$$\mathrm{T} \longmapsto^{*}_{\mathit{let}} A$$

Good news

The rest is (essentially) mechanical.

Reference: Defunctionalized Interpreters for Programming Languages, ICFP'08.

The syntactic correspondence

- Refocusing: from reduction semantics to small-step abstract machine
- Lightweight fusion: from small-step abstract machine to big-step abstract machine
- Transition compression: from big-step abstract machine to big-step abstract machine

The functional correspondence

- Refunctionalization: from abstract machine to continuation-passing interpreter
- Back to direct style: from continuation-passing interpreter to first-order natural semantics
- Refunctionalization: from first-order natural semantics to higher-order natural semantics

Main results

- A readable, hygienic abstract machine.
- A readable, hygienic natural semantics.

Orthogonal issues

- Adding a garbage-collection rule
- Introducing a heap
- Introducing a store

Variants

Ensuring hygiene.

Latest news

More aggressive transition compression (using a global invariant) makes outside-in contexts unnecessary.

Good news for Simon's head:

a continuation-free account of lazy evaluation.

Work in progress.

Conclusion

- The standard call-by-need reduction of the lambda-calculus, plus hygiene, can be uniformly mirrored into an abstract machine and a natural semantics that make sense.

- Further transition compression leads to a continuation-free account of call by need.

Thank you.