Defunctionalized Interpreters for Call-by-Need Evaluation

Olivier Danvy, University of Aarhus
Kevin Millikin, Google
Johan Munk, Arctic Lake Systems
Ian Zerny, University of Aarhus

Sendai, Japan

FLOPS 2010

Motivation

Formal semantics: why?

- Understanding linguistic features
- Proving programs correct or equivalent
- Proving language properties
- Proving implementations, analyses or transformations correct

Formal semantics: which kind?

- Denotational?
- Operational? Big step? Small step?
- Axiomatic?

Foundations of this work

Semantic artifacts can be inter-derived mechanically,

and

the inter-derivation is worthwhile:

- it can yield simpler semantics, and
- it can yield new semantics.

Here: call by need.

Call-by-need evaluation

- Demand-driven computation
- Memoization of intermediate results

Semantics for call-by-need evaluation

- Store-based:
  results are saved in the global store
- Storeless:
  results are saved in the term itself

Syntactic theories of call by need

- Different opinions — the POPL'95 affair
  - The call-by-need lambda-calculus by Ariola and Felleisen (JFP'97)
  - The call-by-need lambda-calculus by Maraist, Odersky and Wadler (JFP'98)

Syntactic theories of call by need

- Different opinions — the POPL'95 affair
  - The call-by-need lambda-calculus by Ariola and Felleisen (JFP'97)
  - The call-by-need lambda-calculus by Maraist, Odersky and Wadler (JFP'98)
- Appearances can be deceiving:
  The standard reduction is common to both

Syntactic theories of call by need

- Different opinions — the POPL'95 affair
  - The call-by-need lambda-calculus
    by Ariola and Felleisen (JFP'97)
  - The call-by-need lambda-calculus
    by Maraist, Odersky and Wadler (JFP'98)
- Appearances can be deceiving:
  The standard reduction is common to both
- It is our starting point

Outline

1. The call-by-need λ-calculus
2. Deriving an abstract machine
   and a natural semantics

The call-by-name λ_{let}-calculus

$$\begin{array}{rcl} \text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\ \text{Values} \ni V & ::= & \lambda x.T \\ \text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \end{array}$$

The call-by-name λ_{let}-calculus

$$
\begin{aligned}
\text{Terms} \ni T &::= x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\
\text{Values} \ni V &::= \lambda x.T \\
\text{Eval Cont} \ni E &::= [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E
\end{aligned}
$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{name}}$

The call-by-name λ_{let}-calculus

$$\begin{array}{rcl} \text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\ \text{Values} \ni V & ::= & \lambda x.T \\ \text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \end{array}$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in z $z \mapsto_{\text{name}}$

The call-by-name λ_{let}-calculus

$$
\begin{aligned}
\text{Terms} \ni T &::= x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\
\text{Values} \ni V &::= \lambda x.T \\
\text{Eval Cont} \ni E &::= [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E
\end{aligned}
$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $\boxed{z}\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $((\lambda y.y)\ (\lambda x.x))\ z \mapsto_{\text{name}}$

The call-by-name λ_{let}-calculus

$$\begin{array}{rcl} \text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\ \text{Values} \ni V & ::= & \lambda x.T \\ \text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \end{array}$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $z\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $((\lambda y.y)\ (\lambda x.x))\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in (let y be $\lambda x.x$ in y) $z \mapsto_{\text{name}}$

The call-by-name λ_{let}-calculus

$$\begin{array}{rcl} \text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\ \text{Values} \ni V & ::= & \lambda x.T \\ \text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \end{array}$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in z $z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $((\lambda y.y)\ (\lambda x.x))\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in (let y be $\lambda x.x$ in y) $z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in y $z \mapsto_{\text{name}}$

The call-by-name λ_{let}-calculus

$$
\begin{array}{rcl}
\text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \textsf{let}\ x\ \textsf{be}\ T\ \textsf{in}\ T \\
\text{Values} \ni V & ::= & \lambda x.T \\
\text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \textsf{let}\ x\ \textsf{be}\ T\ \textsf{in}\ E
\end{array}
$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{name}}$
$\textsf{let}\ z\ \textsf{be}\ (\lambda y.y)\ (\lambda x.x)\ \textsf{in}\ z\ z \mapsto_{\text{name}}$
$\textsf{let}\ z\ \textsf{be}\ (\lambda y.y)\ (\lambda x.x)\ \textsf{in}\ ((\lambda y.y)\ (\lambda x.x))\ z \mapsto_{\text{name}}$
$\textsf{let}\ z\ \textsf{be}\ (\lambda y.y)\ (\lambda x.x)\ \textsf{in}\ (\textsf{let}\ y\ \textsf{be}\ \lambda x.x\ \textsf{in}\ y)\ z \mapsto_{\text{name}}$
$\textsf{let}\ z\ \textsf{be}\ (\lambda y.y)\ (\lambda x.x)\ \textsf{in}\ \textsf{let}\ y\ \textsf{be}\ \lambda x.x\ \textsf{in}\ y\ z \mapsto_{\text{name}}$
$\textsf{let}\ z\ \textsf{be}\ (\lambda y.y)\ (\lambda x.x)\ \textsf{in}\ \textsf{let}\ y\ \textsf{be}\ \lambda x.x\ \textsf{in}\ (\lambda x.x)\ z \mapsto_{\text{name}}$

The call-by-name λ_{let}-calculus

$$
\begin{array}{rcl}
\text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\
\text{Values} \ni V & ::= & \lambda x.T \\
\text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E
\end{array}
$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $\boxed{z}\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $((\lambda y.y)\ (\lambda x.x))\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in (let y be $\lambda x.x$ in y) $z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in $\boxed{y}\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in $(\lambda x.x)\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in let x be z in $\boxed{x} \mapsto_{\text{name}}$

The call-by-name λ_{let}-calculus

$$
\begin{array}{rcl}
\text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\
\text{Values} \ni V & ::= & \lambda x.T \\
\text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E
\end{array}
$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $\boxed{z}\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $((\lambda y.y)\ (\lambda x.x))\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in (let y be $\lambda x.x$ in y) $z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in $\boxed{y}\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in $(\lambda x.x)\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in let x be z in $\boxed{x} \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in let x be z in $\boxed{z} \mapsto_{\text{name}}$

The call-by-name λ_{let}-calculus

$$
\begin{array}{rcl}
\text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\
\text{Values} \ni V & ::= & \lambda x.T \\
\text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E
\end{array}
$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $\boxed{z}\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $((\lambda y.y)\ (\lambda x.x))\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in (let y be $\lambda x.x$ in y) $z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in $\boxed{y}\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in $(\lambda x.x)\ z \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in let x be z in $\boxed{x} \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in let x be z in $\boxed{z} \mapsto_{\text{name}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in let y be $\lambda x.x$ in let x be z in $(\lambda y.y)\ (\lambda x.x)$
...

The call-by-need λ_{let}-calculus

$$
\begin{array}{rcl}
\text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\
\text{Values} \ni V & ::= & \lambda x.T \\
\text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \mid \\
& & \text{let } x \text{ be } E \text{ in } E[x]
\end{array}
$$

The call-by-need λ_{let}-calculus

$$
\begin{array}{rcl}
\text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\
\text{Values} \ni V & ::= & \lambda x.T \\
\text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \mid \\
 & & \text{let } x \text{ be } E \text{ in } E[x]
\end{array}
$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{need}}$

The call-by-need λ_{let}-calculus

$$\begin{array}{rcl} \text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\ \text{Values} \ni V & ::= & \lambda x.T \\ \text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \mid \\ & & \text{let } x \text{ be } E \text{ in } E[x] \end{array}$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{need}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in z $z \mapsto_{\text{need}}$

The call-by-need λ_{let}-calculus

$$
\begin{aligned}
\text{Terms} \ni T &::= x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\
\text{Values} \ni V &::= \lambda x.T \\
\text{Eval Cont} \ni E &::= [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \mid \\
&\qquad \text{let } x \text{ be } E \text{ in } E[x]
\end{aligned}
$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{need}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $\boxed{z}\ z \mapsto_{\text{need}}$
let z be (let y be $\lambda x.x$ in $\boxed{y}$) in $z\ z \mapsto_{\text{need}}$

The call-by-need λ_{let}-calculus

$$\begin{array}{rrcl} \text{Terms} \ni & T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\ \text{Values} \ni & V & ::= & \lambda x.T \\ \text{Eval Cont} \ni & E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \mid \\ & & & \text{let } x \text{ be } E \text{ in } E[x] \end{array}$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{need}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in z $z \mapsto_{\text{need}}$
let z be (let y be $\lambda x.x$ in y) in $z\ z \mapsto_{\text{need}}$
let z be (let y be $\lambda x.x$ in $\lambda x.x$) in $z\ z \mapsto_{\text{need}}$

The call-by-need λ_{let}-calculus

$$\begin{array}{rrcl}
\text{Terms} \ni & T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\
\text{Values} \ni & V & ::= & \lambda x.T \\
\text{Eval Cont} \ni & E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \mid \\
 & & & \text{let } x \text{ be } E \text{ in } E[x]
\end{array}$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{need}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $\boxed{z}\ z \mapsto_{\text{need}}$
let z be (let y be $\lambda x.x$ in $\boxed{y}$) in $z\ z \mapsto_{\text{need}}$
let z be (let y be $\lambda x.x$ in $\lambda x.x$) in $z\ z \mapsto_{\text{need}}$
let y be $\lambda x.x$ in let z be $\lambda x.x$ in $\boxed{z}\ z \mapsto_{\text{need}}$

The call-by-need λ_{let}-calculus

$$\begin{array}{rcl}
\text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\
\text{Values} \ni V & ::= & \lambda x.T \\
\text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \mid \\
 & & \text{let } x \text{ be } E \text{ in } E[x]
\end{array}$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{need}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $z\ z \mapsto_{\text{need}}$
let z be (let y be $\lambda x.x$ in y) in $z\ z \mapsto_{\text{need}}$
let z be (let y be $\lambda x.x$ in $\lambda x.x$) in $z\ z \mapsto_{\text{need}}$
let y be $\lambda x.x$ in let z be $\lambda x.x$ in $z\ z \mapsto_{\text{need}}$
let y be $\lambda x.x$ in let z be $\lambda x.x$ in $(\lambda x.x)\ z \mapsto_{\text{need}}$

The call-by-need λ_{let}-calculus

$$
\begin{array}{rcl}
\text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\
\text{Values} \ni V & ::= & \lambda x.T \\
\text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \mid \\
 & & \text{let } x \text{ be } E \text{ in } E[x]
\end{array}
$$

$(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{need}}$
let z be $(\lambda y.y)\ (\lambda x.x)$ in $\boxed{z}\ z \mapsto_{\text{need}}$
let z be (let y be $\lambda x.x$ in $\boxed{y}$) in $z\ z \mapsto_{\text{need}}$
let z be (let y be $\lambda x.x$ in $\lambda x.x$) in $z\ z \mapsto_{\text{need}}$
let y be $\lambda x.x$ in let z be $\lambda x.x$ in $\boxed{z}\ z \mapsto_{\text{need}}$
let y be $\lambda x.x$ in let z be $\lambda x.x$ in $(\lambda x.x)\ z \mapsto_{\text{need}}$
let y be $\lambda x.x$ in let z be $\lambda x.x$ in let x be $\boxed{z}$ in $x \mapsto_{\text{need}}$

The call-by-need λ_{let}-calculus

$$
\begin{array}{rcl}
\text{Terms} \ni T & ::= & x \mid \lambda x.T \mid T\ T \mid \text{let } x \text{ be } T \text{ in } T \\
\text{Values} \ni V & ::= & \lambda x.T \\
\text{Eval Cont} \ni E & ::= & [\,] \mid E\ T \mid \text{let } x \text{ be } T \text{ in } E \mid \\
 & & \text{let } x \text{ be } E \text{ in } E[x]
\end{array}
$$

$$
\begin{array}{l}
(\lambda z.z\ z)\ ((\lambda y.y)\ (\lambda x.x)) \mapsto_{\text{need}} \\
\text{let } z \text{ be } (\lambda y.y)\ (\lambda x.x) \text{ in } \boxed{z}\ z \mapsto_{\text{need}} \\
\text{let } z \text{ be } (\text{let } y \text{ be } \lambda x.x \text{ in } \boxed{y}) \text{ in } z\ z \mapsto_{\text{need}} \\
\text{let } z \text{ be } (\text{let } y \text{ be } \lambda x.x \text{ in } \lambda x.x) \text{ in } z\ z \mapsto_{\text{need}} \\
\text{let } y \text{ be } \lambda x.x \text{ in let } z \text{ be } \lambda x.x \text{ in } \boxed{z}\ z \mapsto_{\text{need}} \\
\text{let } y \text{ be } \lambda x.x \text{ in let } z \text{ be } \lambda x.x \text{ in } (\lambda x.x)\ z \mapsto_{\text{need}} \\
\text{let } y \text{ be } \lambda x.x \text{ in let } z \text{ be } \lambda x.x \text{ in let } x \text{ be } \boxed{z} \text{ in } x \mapsto_{\text{need}} \\
\text{let } y \text{ be } \lambda x.x \text{ in let } z \text{ be } \lambda x.x \text{ in let } x \text{ be } \lambda x.x \text{ in } \boxed{x} \mapsto_{\text{need}} \\
\text{let } y \text{ be } \lambda x.x \text{ in let } z \text{ be } \lambda x.x \text{ in let } x \text{ be } \lambda x.x \text{ in } \lambda x.x
\end{array}
$$

Accounting for call by need syntactically

On-demand: already present for call by name

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

Memoization: restricting substitution to values

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

and adapting the search
(hence the extra context constructor)

Related semantics for call by need

- A variety of fascinating semantics exist:
  TIM (Fairbairn and Wray, FPCA'87)
  Lazy Krivine Machine (Sestoft, JFP'97)
  Maraist et al., POPL'98
  Garcia et al., POPL'09
  etc.
- How do they relate to
  the call-by-need λ-calculus?

Our thesis

What: We can constructively calculate
the corresponding abstract machine
and other semantic artifacts.

Our thesis

What: We can constructively calculate
the corresponding abstract machine
and other semantic artifacts.

How: We use program transformations
off the shelf.

Our sub-thesis

- Spell out the reduction semantics in detail
- Leads directly to the abstract machine

The reduction semantics

Decomposition
: Searches for the next redex,
  if there is one, and its context

Contraction
: Contracts a redex to a contractum

Recomposition
: Reconstructs a term
  from a contractum and the context

The reduction semantics

Decomposition
: Searches for the next redex,
if there is one, and its context

Contraction
: Contracts a redex to a contractum

Recomposition
: Reconstructs a term
from a contractum and the context

Hygiene
: We maintain hygiene explicitly
(e.g., a variable convention)

The reduction semantics

One-step reduction

Decompose, contract and recompose

Reduction-based evaluation

Iterate one-step reduction

The syntactic correspondence

Reduction semantics

↓ Danvy and Nielsen's refocusing

↓ inlining

↓ Ohori and Sasano's lightweight fusion

↓ transition compression

Abstract machine

The syntactic correspondence

Reduction semantics

↓ Danvy and Nielsen's refocusing

↓ inlining

↓ Ohori and Sasano's lightweight fusion

↓ transition compression

Abstract machine

Result: the simplest storeless abstract machine for call-by-need evaluation (see paper).

The functional correspondence

Kudos for John Reynolds:

Evaluator

↓ closure conversion

↓ CPS transformation

↓ defunctionalization

Abstract machine

The functional correspondence

Kudos for John Reynolds:

Evaluator

↓ closure conversion

↓ CPS transformation

↓ defunctionalization

Abstract machine

Result: the first storeless natural semantics for call-by-need evaluation (see paper).

Our results

- Identify what is common to Ariola et al.: the standard reduction
- Calculate the corresponding abstract machine
- Calculate the corresponding natural semantics
- And also: calculate the corresponding extended direct semantics
- A denotational semantics can be constructed with a control monad or a state monad

Our results

- Identify what is common to Ariola et al.:
  the standard reduction
- Calculate the corresponding abstract machine
- Calculate the corresponding natural semantics
- And also: calculate the corresponding
  extended direct semantics
- A denotational semantics can be constructed
  with a control monad or a state monad

Thank you.