Continuations, Processes, and Sharing

Paul Downen, Luke Maurer, Zena M. Ariola,
Daniele Varacca

University of Oregon, Université Paris Diderot

September 8, 2014

The plethora of semantic artifacts

- Many ways to understand programming languages:
  - small-step semantics
  - big-step semantics (natural semantics)
  - abstract machines
  - continuation-passing style transformations
  - ...
- Different tools; different views
  - High-level reasoning
  - Low-level reasoning
  - Proof development

Getting along together

Q But which one to choose?

Getting along together

Q But which one to choose?

A All of them!

Getting along together

Q But which one to choose?

A All of them!

Q But how do we know that they agree?

Getting along together

Q But which one to choose?

A All of them!

Q But how do we know that they agree?

A Systematic inter-derivation; correct by construction (Danvy et al.)

Processes as a semantic tool

- Embedding into processes (π-calculus)
- Computation as communication
- Strong resemblance to continuation-passing

Processes as a semantic tool

- Embedding into processes (π-calculus)
- Computation as communication
- Strong resemblance to continuation-passing

What do processes have to offer?

- Some computations more direct in π- than λ-calculus
  - Concurrency
  - Non-determinism
  - Change over time
- Simple story for memoization
- Reveals techniques in implementations of lazy languages
  - "Black holes" in GHC

From continuations to processes

Example: function composition

Transform the function composition

$$f\ (g\ 1)$$

into a process

Result-named style

Name intermediate results of serious computations:

$$f\ (g\ 1)$$

goes to

$$\textbf{let}\ y = g\ 1\ \textbf{in}\ f\ \ y$$

Continuation-passing style

Rewrite into continuations:

$$\textbf{let}\, y = g\ 1\, \textbf{in}\, f\ \ y$$

goes to

$$g(1, (\lambda y.\, f(y, \textbf{ret})))$$

Value-named style

Name all serious values:

$$g(1, (\lambda y.\, f(y, \textbf{ret})))$$

goes to

$$\textbf{let}\; k = \lambda y.\, f(y, \textbf{ret})\; \textbf{in}\; g(1, k)$$

Environment-based CPS

Rewrite into explicit environment:

$$\textbf{let}\ k = \lambda y.\, f(y, \textbf{ret})\ \textbf{in}\ g(1, k)$$

goes to

$$\nu k.\, k := \lambda y.\, f(y, \textbf{ret})\ \textbf{in}\ g(1, k)$$

Process encoding

Rewrite into processes:

$$\nu k.\, k := \lambda y.\, f(y, \mathbf{ret})\ \mathbf{in}\ g(1, k)$$

goes to

$$\boldsymbol{\nu} k\, (!k(y).\, \overline{f}\langle y, \mathbf{ret}\rangle \mid \overline{g}\langle 1, k\rangle)$$

Uniform CPS transform (CBN and CBV)

$$\mathcal{C}[\![x]\!] \triangleq \lambda k.\, x\ k$$

$$\mathcal{C}[\![\lambda x.\, M]\!] \triangleq \lambda k.\, k\ (\lambda(x, k').\, \mathcal{C}[\![M]\!] k')$$

Call-by-name

$$\mathcal{C}[\![MN]\!] \triangleq \lambda k.\, \mathcal{C}[\![M]\!](\lambda v.\, v\ (\lambda k'.\, \mathcal{C}[\![N]\!] k', k))$$

Call-by-value

$$\mathcal{C}[\![MN]\!] \triangleq \lambda k.\, \mathcal{C}[\![M]\!](\lambda v.\, \mathcal{C}[\![N]\!]\ (\lambda w.\, v\ (\lambda k'.\, k' w, k)))$$

Uniform CPS to Uniform π-encoding

$$\begin{aligned}
\mathcal{C}[\![x]\!]k &= x\ k \\
\mathcal{N} \circ \mathcal{C}[\![x]\!]k &= x\ k \\
\mathcal{P} \circ \mathcal{N} \circ \mathcal{C}[\![x]\!]k &= \overline{x}\langle k \rangle
\end{aligned}$$

$$\begin{aligned}
\mathcal{C}[\![\lambda x.\, M]\!]k &= k\ (\lambda(x, k').\, \mathcal{C}[\![M]\!]k') \\
\mathcal{N} \circ \mathcal{C}[\![\lambda x.\, M]\!]k &= \nu f.\, f := \lambda(x, k').\, \mathcal{N} \circ \mathcal{C}[\![M]\!]k' \textbf{ in } k\ f \\
\mathcal{P} \circ \mathcal{N} \circ \mathcal{C}[\![\lambda x.\, M]\!]k &= \boldsymbol{\nu} f\ (!f(x, k').\, \mathcal{P} \circ \mathcal{N} \circ \mathcal{C}[\![M]\!]k' \mid \overline{k}\langle f \rangle)
\end{aligned}$$

$$\ldots$$

Interlude: Of variables and values

A mismatch

Soundness: steps in source are steps in target

$$(\lambda x.\, \lambda y.\, y)z = \lambda y.\, y$$

This is invalid by CBV transform of application:

$$\begin{aligned} \mathcal{C}[\![(\lambda x.\, \lambda y.\, y)z]\!] &= \lambda k.\, z\ (\lambda w.\, \mathcal{C}[\![\lambda y.\, y]\!]k) \\ &\neq \mathcal{C}[\![\lambda y.\, y]\!] \end{aligned}$$

Variables are not values

$$\mathcal{C}[\![x]\!] \triangleq \lambda k.\, x\ k$$

In CBN we need to execute a computation

In CBV we need to lookup or fetch the value

	Plotkin CBV	**Restricted CBV**
Values:	$V ::= x \mid \lambda x.\, M$	$V ::= \lambda x.\, M$
Evaluation Contexts:	$E ::= [\,] \mid EM \mid VE$	$E ::= [\,] \mid EM \mid VE$

$$(\lambda x.\, M)V = M\{V/x\} \qquad (\beta_v)$$

Correctness bisimulation

Criteria for correctness

A transformation should preserve observable results of a program:

- Termination: the program reaches an answer
- Divergence: the program loops forever
- Getting stuck: the program cannot proceed

How the proof should go

- $\mathcal{T}[\![\cdot]\!]$ preserves immediate results
  - If M *is* an answer then $\mathcal{T}[\![M]\!]$ *is* an answer
  - $\ldots$
- $\mathcal{T}[\![\cdot]\!]$ preserves reduction

$$\begin{array}{ccc} M & \longmapsto & N \\ \rightsquigarrow & & \rightsquigarrow \\ \mathcal{T}[\![M]\!] & \mapsto\!\!\!\twoheadrightarrow & \mathcal{T}[\![N]\!] \end{array}$$

How the proof actually goes

- $\mathcal{T}[\![\cdot]\!]$ preserves immediate results
  - If M *is* an answer then $\mathcal{T}[\![M]\!]$ *reaches* an answer
  - ...
- $\mathcal{T}[\![\cdot]\!]$ preserves reduction

$$\begin{tikzcd}
M \arrow[r, mapsto] \arrow[d, rightsquigarrow] & N \arrow[d, rightsquigarrow] \\
\mathcal{T}[\![M]\!] \arrow[r, mapsto, twoheadrightarrow] & \mathcal{T}[\![N]\!]
\end{tikzcd}$$

How the proof actually goes

- $\mathcal{T}\llbracket\cdot\rrbracket$ preserves immediate results
  - If M *is* an answer then $\mathcal{T}\llbracket M\rrbracket$ *reaches* an answer
  - ...
- $\mathcal{T}\llbracket\cdot\rrbracket$ preserves reduction?

$$\begin{tikzcd}
M \arrow[rr, mapsto] \arrow[d, rightsquigarrow] & & N \arrow[d, rightsquigarrow] & \\
\mathcal{T}\llbracket M\rrbracket \arrow[rrr, bend right, mapsto, two heads] & & \mathcal{T}\llbracket N\rrbracket \arrow[r, mapsto, two heads] & P
\end{tikzcd}$$

How the proof actually goes

- $\mathcal{T}[\![\cdot]\!]$ preserves immediate results
  - If M *is* an answer then $\mathcal{T}[\![M]\!]$ *reaches* an answer
  - ...
- $\mathcal{T}[\![\cdot]\!]$ preserves reduction???

$$
\begin{tikzcd}
M \arrow[r, mapsto] \arrow[d, rightsquigarrow] & N \arrow[d, rightsquigarrow] \\
\mathcal{T}[\![M]\!] \arrow[dr, "" ', |->>] & \mathcal{T}[\![N]\!] \arrow[d, phantom, "?"] \\
 & Q
\end{tikzcd}
$$

Out-of-synch computations: administration

Source:

$$(\lambda x.\, x)(\lambda y.\, y) \longmapsto \lambda y.\, y$$

Target:

$$\mathcal{C}[\![(\lambda x.\, x)(\lambda y.\, y)]\!]\,\mathbf{ret} \longmapsto\!\!\!\!\rightarrow \mathbf{ret}\ (\lambda(y, k).\, (\lambda k'.\, y\ \ k')\ \ k)$$

$$\mathcal{C}[\![\lambda y.\, y]\!]\,\mathbf{ret} \longmapsto \mathbf{ret}\ (\lambda(y, k).\, (\lambda k'.\, y\ \ k')\ \ k)$$

Out-of-synch computations: aliasing

Source:

$$(\lambda f.\, g(f,f))(\lambda x.\, x) \longmapsto g((\lambda x.\, x),(\lambda x.\, x))$$

Target:

$$\begin{aligned}
&\mathcal{N}[\![(\lambda f.\, g(f,f))(\lambda x.\, x)]\!]\\
&= \nu i.\, i := \lambda x.\, x \;\mathbf{in}\; (\lambda f.\, g(f,f))\ i\\
&\longmapsto \nu i.\, i := \lambda x.\, x \;\mathbf{in}\; g(i,i)
\end{aligned}$$

$$\begin{aligned}
&\mathcal{N}[\![g((\lambda x.\, x),(\lambda x.\, x))]\!]\\
&= \nu i.\, i := \lambda x.\, x \;\mathbf{in}\; \nu j.\, j := \lambda x.\, x \;\mathbf{in}\; g(i,j)
\end{aligned}$$

Reasoning up to out-of-synch administration

Define an administrative free transform (Danvy and Nielsen TCS 2003)

$$
\begin{tikzcd}
M \arrow[r, mapsto] \arrow[d, squiggly] & N \arrow[d, squiggly] \\
\mathcal{T}[\![M]\!] \arrow[d, two heads, "ad"] & \mathcal{T}[\![N]\!] \arrow[d, two heads, "ad"] \\
P \arrow[r, mapsto, two heads] & Q
\end{tikzcd}
$$

Reasoning up to out-of-synch administration

Reason up to bisimulation

$$
\begin{array}{ccc}
M & \longmapsto & N \\
\sim & & \sim \\
P & \longmapsto\!\!\!\!\rightarrow & Q
\end{array}
$$

$$M \sim P \text{ iff } \mathcal{T}[\![M]\!]\,\mathbf{ret} \twoheadrightarrow_{ad} P$$

Reasoning up to out-of-synch aliasing

Reason up to bisimulation

$$\begin{array}{ccc} M & \longmapsto & N \\ \sim & & \sim \\ P & \longmapsto & Q \end{array}$$

$$M \sim P \text{ iff } M \equiv \mathcal{N}^{-1}\langle\!\langle P \rangle\!\rangle$$

Bisimulation technique

Start out similar

$$\begin{array}{c} M \\ \sim \\ \mathcal{T}[\![M]\!] \end{array}$$

Keep being similar

$$\begin{array}{ccc} M & \longmapsto & N \\ \sim & & \sim \\ P & \mapsto\!\!\!\twoheadrightarrow & Q \end{array} \qquad \begin{array}{ccc} M & \mapsto\!\!\!\twoheadrightarrow & N \\ \sim & & \sim \\ P & \longmapsto & Q \end{array}$$

End up similar

$$\begin{array}{l} M \downarrow \\ \sim \\ P \mapsto\!\!\!\twoheadrightarrow Q \downarrow \end{array} \qquad \begin{array}{l} M \mapsto\!\!\!\twoheadrightarrow N \downarrow \\ \sim \\ Q \downarrow \end{array}$$

One direction suffices

The forward direction sufficient if (Leroy):

- The source language is deterministic;
- No infinite loop in source terminates in target

Dichotomy of source reductions (Danvy and Zerny, PPDP'13) guarantees point 2:

- Proper reduction *must* cause work in target
- Administrative reduction *must* terminate

Sharing

Call-by-need evaluation

$$\textbf{let}\, x = 1 + 2 \textbf{ in } x * x$$

Call-by-need evaluation

$$\textbf{let}\, x = 1 + 2\ \textbf{in}\ x * x$$

Call-by-need evaluation

$$\textbf{let}\, x = 1 + 2 \,\textbf{in}\, x * x$$

Call-by-need evaluation

$$\begin{aligned} &\textbf{let}\, x = 1 + 2 \ \textbf{in}\ x * x \\ \longmapsto\ &\textbf{let}\, x = 3\ \textbf{in}\ x * x \end{aligned}$$

Call-by-need evaluation

$$\begin{aligned} & \textbf{let}\, x = 1 + 2 \textbf{ in } x * x \\ \longmapsto\ & \textbf{let}\, x = 3 \textbf{ in } x * x \\ \longmapsto\ & \textbf{let}\, x = 3 \textbf{ in } 3 * x \end{aligned}$$

Call-by-need evaluation

$$
\begin{aligned}
&\mathbf{let}\, x = 1 + 2 \,\mathbf{in}\, x * x \\
&\longmapsto \mathbf{let}\, x = 3 \,\mathbf{in}\, x * x \\
&\longmapsto \mathbf{let}\, x = 3 \,\mathbf{in}\, 3 * x
\end{aligned}
$$

Call-by-need evaluation

$$
\begin{aligned}
&\textbf{let}\, x = 1 + 2 \textbf{ in } x * x \\
\longmapsto\ &\textbf{let}\, x = 3 \textbf{ in } x * x \\
\longmapsto\ &\textbf{let}\, x = 3 \textbf{ in } 3 * x
\end{aligned}
$$

Call-by-need evaluation

$$
\begin{aligned}
&\textbf{let}\, x = 1 + 2 \;\textbf{in}\; x * x \\
&\longmapsto \textbf{let}\, x = 3 \;\textbf{in}\; x * x \\
&\longmapsto \textbf{let}\, x = 3 \;\textbf{in}\; 3 * x \\
&\longmapsto \textbf{let}\, x = 3 \;\textbf{in}\; 3 * 3
\end{aligned}
$$

Call-by-need evaluation

$$
\begin{aligned}
&\mathbf{let}\, x = 1 + 2 \,\mathbf{in}\, x * x \\
\longmapsto\ &\mathbf{let}\, x = 3 \,\mathbf{in}\, x * x \\
\longmapsto\ &\mathbf{let}\, x = 3 \,\mathbf{in}\, 3 * x \\
\longmapsto\ &\mathbf{let}\, x = 3 \,\mathbf{in}\, 3 * 3
\end{aligned}
$$

Call-by-need evaluation

$$
\begin{aligned}
&\textbf{let}\, x = 1 + 2 \textbf{ in } x * x \\
&\longmapsto \textbf{let}\, x = 3 \textbf{ in } x * x \\
&\longmapsto \textbf{let}\, x = 3 \textbf{ in } 3 * x \\
&\longmapsto \textbf{let}\, x = 3 \textbf{ in } 3 * 3 \\
&\longmapsto \textbf{let}\, x = 3 \textbf{ in } 9
\end{aligned}
$$

Call-by-need and stateful CPS

Okasaki call-by-need CPS using assignment

$$
\begin{aligned}
\mathcal{C}[\![x]\!] &\triangleq \lambda k.\, x\ k \\
\mathcal{C}[\![\lambda x.\, M]\!] &\triangleq \lambda k.\, k\ (\lambda(x, k').\, \mathcal{C}[\![M]\!]k') \\
\mathcal{C}[\![MN]\!] &\triangleq \lambda k.\, \mathcal{C}[\![M]\!](\lambda v.\, \nu x. \\
&\qquad\qquad x := \mathrm{memo}_x(N)\ \mathbf{in}\ v(x, k))
\end{aligned}
$$

$$
\mathrm{memo}_x(N) \triangleq \lambda k.\, \mathcal{C}[\![N]\!](\lambda w.\, x := (\lambda k'.\, k'\ w)\ \mathbf{in}\ k\ w)
$$

On liveness of variables

- First evaluate M, with N assigned to x

$$\mathbf{let}\, x = N\, \mathbf{in}\, M$$

- When x is forced, evaluate N and x is no longer in scope

$$\mathbf{let}\, x = N\, \mathbf{in}\, E[x]$$

- When N becomes V, continue in body with V assigned to x

$$\mathbf{let}\, x = V\, \mathbf{in}\, E[V]$$

Constructive update

Initial binding is ephemeral, disappears on lookup

$$\begin{array}{l} \nu f.\, f :=_1 \mathsf{memo}_f(N) \;\mathbf{in}\; f\ k \\ \longmapsto \nu f.\ \mathsf{memo}_f(N) k \end{array}$$

Updated binding is permanent, always available and can never be changed

$$\begin{array}{l} \nu f.\, f := V \;\mathbf{in}\; f\ k \\ \longmapsto \nu f.\, f := V \;\mathbf{in}\; V\ k \end{array}$$

Call-by-need and constructive update CPS

Thunking protocol strictly enforced (thunks only evaluated once):

$$
\begin{aligned}
\mathcal{C}[\![x]\!] &\triangleq \lambda k.\, x\ k \\
\mathcal{C}[\![\lambda x.\, M]\!] &\triangleq \lambda k.\, k\ (\lambda(x, k').\, \mathcal{C}[\![M]\!]k') \\
\mathcal{C}[\![MN]\!] &\triangleq \lambda k.\, \mathcal{C}[\![M]\!](\lambda v.\, \nu x. \\
&\qquad\qquad x :=_1 \mathrm{memo}_x(N)\ \mathbf{in}\ v(x, k))
\end{aligned}
$$

$$
\mathrm{memo}_x(N) \triangleq \lambda k.\, \mathcal{C}[\![N]\!](\lambda w.\, x := (\lambda k'.\, k'\ w)\ \mathbf{in}\ k\ w)
$$

Constructive update in the π-calculus

Permanent assignment: replicated server

$$\mathcal{P}[\![x := \lambda y.\, M \textbf{ in } N]\!] = !x(y).\, \mathcal{P}[\![M]\!] \mid \mathcal{P}[\![N]\!]$$

Ephemeral assignment: unreplicated server

$$\mathcal{P}[\![x :=_1 \lambda y.\, M \textbf{ in } N]\!] = x(y).\, \mathcal{P}[\![M]\!] \mid \mathcal{P}[\![N]\!]$$

Processes can now responsively *change* their behavior

Constructive update in machines

- Suspended computations removed on retrieval (Sestoft)
- Thunks become dead when forced
  - "Black holes" in GHC
- Practical implementation techniques reflected by theory

Conclusions

- Program transformation from CPS to processes
- Unite CBN and CBV with CBNeed
- Constructive update model of memoization
- CPS λ-calculus for change over time

Conclusions

- Program transformation from CPS to processes
- Unite CBN and CBV with CBNeed
- Constructive update model of memoization
- CPS λ-calculus for change over time
- Reminder: Decisions have consequences; live with them

Questions?