Control Controls Extensional Execution

Philip Johnson-Freyd   Paul Downen
Zena M. Ariola

University of Oregon

HOPE'15, August 30, 2015

Some Goals of Language Design

The language designer's dilemma

- Many goals for a programming language
  - Easy to reason about
  - Modularity and compositionality
  - Efficient implementations with simple specifications
- But sometimes good things don't go together
  - λ-calculus with η and *weak-head normal forms* is inconsistent
- The problem is accentuated by side-effects (like divergence and control effects)

Have our cake and eat it too

- But control reveals a way out of the dilemma
- Reprioritize our choices for what we *really* want

Benefits of extensionality (η)

Extensionality is essential for *observational properties* about programs

Monad law for State in Haskell:

```
m >>= return
= -- inline (>>=) and return
\s -> let (x, s') = m s in (x, s')
= -- lazy pattern-match
\s -> (fst (m s), snd (m s))
= -- eta law for tuples
\s -> m s
= -- eta law for functions
m
```

Benefits of laziness (call-by-name evaluation)

Laziness enables *infinite data structures* and decoupling *producers* from *consumers*

John Hugh's minimax algorithm in Haskell:

```
gameTree :: Position -> Tree Position
estimate :: Position -> Score
mapTree  :: (a -> b) -> Tree a -> Tree b
prune    :: Int -> Tree a -> Tree a
maximize, minimize :: Tree Score -> Score

evaluate :: Int -> Position -> Score
evaluate n = maximize . mapTree estimate
           . prune n . gameTree
```

Benefits of evaluating closed terms (WHNFs)

Evaluating closed terms means the substitution

$$(\lambda x.v)\ v' \to v[v'/x]$$

does not require renaming

Results are *weak-head normal forms:*

$$\begin{aligned} \mathit{WHNF} ::= {} & \lambda x.v \\ & \mid x\ v_1\ \ldots\ v_n \end{aligned}$$

$\lambda x.(\lambda y.y)\ x$ is done, but $(\lambda y.y)\ x$ is a redex

The trilemma

Observing WHNFs is inconsistent with the untyped λ-calculus

$$(\lambda x.\Omega\ x) = \Omega \qquad (\eta)$$

Ω loops forever while $(\lambda x.\Omega\ x)$ is done

Ways out: give up one of the above

η, call-by-name, and WHNFs, pick at most two:

- Punt (only observe programs of certain types)
- Give up on η ("fast and loose reasoning")
- Give up on call-by-name (call-by-value)
- Give up on WHNFs (head normal forms)?

Head normal forms

Computing head normal forms:

$$\begin{aligned} \mathit{HNF} ::= {} & \lambda x.\mathit{HNF} \\ & \mid x \; v_1 \; \ldots \; v_n \end{aligned}$$

Seems to require *evaluating open terms* (inside λs)

$\lambda x.(\lambda y.y)\ x$ is not done, must reduce $(\lambda y.y)\ x$

Abstract Machines, Control, and Call Stacks

Krivine Machine

$$v \in \mathit{Term} ::= x \mid \lambda x.v \mid v\ v$$
$$E \in \mathit{CoTerm} ::= \mathsf{tp} \mid v \cdot E$$
$$c \in \mathit{Command} ::= \langle v | E \rangle$$

$$\langle v\ v' | E \rangle \rightsquigarrow \langle v | v' \cdot E \rangle$$
$$\langle \lambda x.v | v' \cdot E \rangle \rightsquigarrow \langle v[v'/x] | E \rangle$$

$$\langle \lambda x.v | \mathsf{tp} \rangle \not\rightsquigarrow \qquad \langle x | E \rangle \not\rightsquigarrow$$

Labeling the context

- We give names (x, y, z) to terms
- Why not give names (α, β, γ) to co-terms?

$$
\begin{aligned}
v \in \mathit{Term} &::= x \mid \lambda x.v \mid v\ v \mid \mu\alpha.c \\
E \in \mathit{CoTerm} &::= \mathrm{tp} \mid v \cdot E \mid \alpha
\end{aligned}
$$

$$
\begin{aligned}
\langle v\ v' | E \rangle &\rightsquigarrow \langle v | v' \cdot E \rangle \\
\langle \lambda x.v | v' \cdot E \rangle &\rightsquigarrow \langle v[v'/x] | E \rangle \\
\langle \mu\alpha.c | E \rangle &\rightsquigarrow c[E/\alpha]
\end{aligned}
$$

Pattern-matching on the context

$$\begin{aligned}
\langle \mu\alpha.c | v' \cdot E \rangle &\rightsquigarrow c[(v' \cdot E)/\alpha] \\
\langle \mu(x \cdot \alpha).c | v' \cdot E \rangle &\rightsquigarrow c[v'/x, E/\alpha] \\
\langle \lambda x.v | v' \cdot E \rangle &\rightsquigarrow \langle v[v'/x] | E \rangle
\end{aligned}$$

$$\begin{aligned}
\mu(x \cdot \alpha).c &= \lambda x.\mu\alpha.c \\
\lambda x.v &= \mu(x \cdot \alpha).\langle v | \alpha \rangle
\end{aligned}$$

Digression: tuples in programming languages

Two ways to break down tuples:

- Matching/destructuring bind: $\textbf{let}\,(x, y) = v \textbf{ in } v'$
- Projection: $\mathrm{fst}(v)$, $\mathrm{snd}(v)$

Both ways are equivalent:

$$\mathrm{fst}(v) = \textbf{let}\,(x, y) = v \textbf{ in } x$$
$$\mathrm{snd}(v) = \textbf{let}\,(x, y) = v \textbf{ in } y$$

$$\textbf{let}\,(x, y) = v \textbf{ in } v' = v'[\mathrm{fst}(v)/x, \mathrm{snd}(v)/y]$$

Call stack as structures

- $\mu\alpha.c$ dual to $\textbf{let}\, x = \square\, \textbf{in}\, v$
- $\mu(x \cdot \alpha).c$ dual to $\textbf{let}\,(x, y) = \square\, \textbf{in}\, v$
- Recall

$$\textbf{let}\,(x, y) = v\, \textbf{in}\, v' = v'[\mathrm{fst}(v)/x, \mathrm{snd}(v)/y]$$

- So…

$$\langle \mu(x \cdot \alpha).c | E \rangle = c[\mathrm{car}(E)/x, \mathrm{cdr}(E)/\alpha]$$

Functions as co-data

- Call stacks ($v \cdot E$) are constructed
- Functions are their destructors
- Alternatively, projections out of call stacks
- Negative functions from sequent calculus
  - Herbelin (2005), Munch-Maccagnoni (2013), Downen and Ariola (2014)
- Restores confluence to λ-calculi with control
  - Nakazawa and Nagai (2014)

Implementing Head Reduction

A head reduction abstract machine (with control)

$$\begin{aligned} v \in \mathit{Term} &::= x \mid v\ v \mid \mu\alpha.c \mid \mu(x\cdot\alpha).c \mid \mathrm{car}(S) \\ E \in \mathit{CoTerm} &::= S \mid \alpha \mid v \cdot E \\ S \in \mathit{StuckCoTerm} &::= \mathrm{tp} \mid \mathrm{cdr}(S) \\ c \in \mathit{Command} &::= \langle v \| E \rangle \end{aligned}$$

$$\begin{aligned} \langle v\ v' \| E \rangle &\rightsquigarrow \langle v \| v' \cdot E \rangle \\ \langle \mu\alpha.c \| E \rangle &\rightsquigarrow c[E/\alpha] \\ \langle \mu(x\cdot\alpha).c \| v \cdot E \rangle &\rightsquigarrow c[v/x, E/\alpha] \\ \langle \mu(x\cdot\alpha).c \| S \rangle &\rightsquigarrow c[\mathrm{car}(S)/x, \mathrm{cdr}(S)/\alpha] \end{aligned}$$

$$\langle \mathrm{car}(S) \| E \rangle \not\rightsquigarrow \qquad\qquad \langle x \| E \rangle \not\rightsquigarrow$$

Back to λ-calculus

$$
\begin{aligned}
v \in \mathit{Term} &::= x \mid \lambda x.v \mid v\ v \mid \mathrm{car}(S) \\
E \in \mathit{CoTerm} &::= S \mid v \cdot E \\
S \in \mathit{StuckCoTerm} &::= \mathrm{tp} \mid \mathrm{cdr}(S) \\
c \in \mathit{Command} &::= \langle v | E \rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle v\ v' | E \rangle &\rightsquigarrow \langle v | v' \cdot E \rangle \\
\langle \lambda x.v | v' \cdot E \rangle &\rightsquigarrow \langle v[v'/x] | E \rangle \\
\langle \lambda x.v | S \rangle &\rightsquigarrow \langle v[\mathrm{car}(S)/x] | \mathrm{cdr}(S) \rangle
\end{aligned}
$$

$$\langle \mathrm{car}(S) | E \rangle \not\rightsquigarrow \qquad\qquad \langle x | E \rangle \not\rightsquigarrow$$

Coalescing projections: de Bruijn indexes

$$\begin{aligned}\mathrm{drop}^n(\mathrm{tp}) &= \mathrm{cdr}(\overset{n}{\ldots}\mathrm{cdr}(\mathrm{tp}))\\ \mathrm{pick}^n(\mathrm{tp}) &= \mathrm{car}(\mathrm{drop}^n(\mathrm{tp}))\end{aligned}$$

$$\begin{aligned}v \in \mathit{Term} &::= x \mid \lambda x.v \mid v\ v \mid \mathrm{pick}^n(\mathrm{tp})\\ E \in \mathit{CoTerm} &::= \mathrm{drop}^n(\mathrm{tp}) \mid v \cdot E\\ c \in \mathit{Command} &::= \langle v \| E\rangle\end{aligned}$$

$$\begin{aligned}\langle v\ v' \| E\rangle &\rightsquigarrow \langle v \| v' \cdot E\rangle\\ \langle \lambda x.v \| v' \cdot E\rangle &\rightsquigarrow \langle v[v'/x] \| E\rangle\\ \langle \lambda x.v \| \mathrm{drop}^n(\mathrm{tp})\rangle &\rightsquigarrow \langle v[\mathrm{pick}^n(\mathrm{tp})/x] \| \mathrm{drop}^{n+1}(\mathrm{tp})\rangle\end{aligned}$$

$$\langle \mathrm{pick}^n(\mathrm{tp}) \| E\rangle \not\rightsquigarrow \qquad\qquad \langle x \| E\rangle \not\rightsquigarrow$$

Closed head reduction

$$\begin{aligned}\langle \lambda x.(\lambda y.y)\ x | \mathsf{tp}\rangle &\rightsquigarrow \langle (\lambda y.y)\ \ \mathsf{car}(\mathsf{tp}) | \mathsf{cdr}(\mathsf{tp})\rangle \\ &\rightsquigarrow \langle \lambda y.y | \mathsf{car}(\mathsf{tp}) \cdot \mathsf{cdr}(\mathsf{tp})\rangle \\ &\rightsquigarrow \langle \mathsf{car}(\mathsf{tp}) | \mathsf{cdr}(\mathsf{tp})\rangle \\ &\leftsquigarrow \langle \lambda x.x | \mathsf{tp}\rangle\end{aligned}$$

$$\begin{aligned}\langle \lambda x.\Omega\ x | \mathsf{tp}\rangle &\rightsquigarrow \langle \Omega\ \ \mathsf{car}(\mathsf{tp}) | \mathsf{cdr}(\mathsf{tp})\rangle \\ &\rightsquigarrow \langle \Omega | \mathsf{car}(\mathsf{tp}) \cdot \mathsf{cdr}(\mathsf{tp})\rangle \\ &\rightsquigarrow \ldots\end{aligned}$$

Restoring extensionality by concretizing the context

- Reconcile extensionality and call-by-name by computing HNFs instead of WHNFs
  - Correspond to existing semantics for head reduction (Barendregt; Sestoft, 2002)
- Still maintain closed execution of HNFs
  - Descend into top-level λs by projecting out of top-level context
- Coalesced stack operations as de Bruijn indexes

Lessons learned

- Studying control can help design even pure languages
- We can have our cake and eat it too
- Don’t let the means destroy the ends

Appendix

Small-step operational semantics for (weak-)head reduction

$$
\begin{aligned}
v \in \mathit{Term} &::= x \mid \lambda x.v \mid v\ v \\
E \in \mathit{EvalCxt} &::= \square \mid E\ v \\
H \in \mathit{HeadCxt} &::= E \mid \lambda x.H
\end{aligned}
$$

Head reduction:

$$H[(\lambda x.v)\ v'] \mapsto H[v[v'/x]]$$

Weak-head reduction:

$$E[(\lambda x.v)\ v'] \mapsto E[v[v'/x]]$$

Big-step operational semantics for weak-head reduction

$$\frac{}{\lambda x.v \Downarrow_{wh} \lambda x.v} \qquad \frac{}{x \Downarrow_{wh} x}$$

$$\frac{v_1 \Downarrow_{wh} \lambda x.v_1' \quad v_1'[v_2/x] \Downarrow_{wh} v'}{v_1\ v_2 \Downarrow_{wh} v'}$$

$$\frac{v_1 \Downarrow_{wh} v_1' \quad v_1' \neq \lambda x.v_1''}{v_1\ v_2 \Downarrow_{wh} v_1'\ v_2}$$

Big-step operational semantics for head reduction

$$\frac{v \Downarrow_{wh} \lambda x.v' \quad v' \Downarrow_h v''}{v \Downarrow_h \lambda x.v''}$$

$$\frac{v \Downarrow_{wh} v' \quad v' \neq \lambda x.v''}{v \Downarrow_h v'}$$

References I

H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, Amsterdam, 1984.

P. Downen and Z. M. Ariola. The duality of construction. In *ESOP*, pages 249–269, 2014.

H. Herbelin. C'est maintenant qu'on calcule : Au cœur de la dualité. In *Habilitation à diriger les reserches*, 2005.

G. Munch-Maccagnoni. *Syntax and Models of a non-Associative Composition of Programs and Proofs*. PhD thesis, Univ. Paris Diderot, 2013.

K. Nakazawa and T. Nagai. Reduction system for extensional lambda-mu calculus. In *RTA-TLCA*, pages 349–363, 2014.

P. Sestoft. Demonstrating lambda calculus reduction. In *The Essence of Computation*, pages 420–435. 2002.