Effective Equality

Overcoming Obstacles with Beta and Eta
Or: How I Learned to Stop Worrying and Love Control

Paul Downen

Shonan 203: Effect Handlers & General Purpose Languages
Thursday, September 28, 2023 — Programmer-Facing Aspects

What's So Hard About Equality and Effects

Now the order matters…

Now the order matters...

(α) Want $\lambda x.M = \lambda y.M\{y/x\}$. Ok.

What's So Hard About Equality and Effects

Now the order matters...

(α) Want $\lambda x.M = \lambda y.M\{y/x\}$. Ok.

(β) Want $(\lambda x.M)\,N = M\{N/x\}$, but

$(\lambda x.M)\,(\texttt{print}\ 5) \neq M\{\texttt{print}\ 5/x\}$, drat

$(\lambda x.M)\,(\texttt{loop\_forever}) \neq M\{\texttt{loop\_forever}/x\}$

in CBV, anyway

What's So Hard About Equality and Effects

Now the order matters...

(α) Want $\lambda x.M = \lambda y.M\{y/x\}$. Ok.

(β) Want $(\lambda x.M)\ N = M\{N/x\}$, but

$(\lambda x.M)\ (\texttt{print}\ 5) \neq M\{\texttt{print}\ 5/x\}$, drat

$(\lambda x.M)\ (\texttt{loop\_forever}) \neq M\{\texttt{loop\_forever}/x\}$

in CBV, anyway

(η) Want $\lambda x.\ M\ x = M$ (when $x \notin FV(M)$), but

$\lambda x.(\texttt{raise}\ \texttt{"oops"})\ x \neq \texttt{raise}\ \texttt{"oops"}$, oops

$\lambda x.\texttt{loop\_forever}\ x \neq \texttt{loop\_forever}$,

in CBV, anwyay

What Does a Variable Stand For?

Choose your own adventure!

Pick what expressions might replace a variable (V)

"Values" (V) are safe to copy/delete freely (i.e., by substitution)

I don't care how you pick these

…but be consistent

Substitution $\{V/x\}$ is only defined for these "values"

What Does a Variable Stand For?

Choose your own adventure!

Pick what expressions might replace a variable (V)

“Values” (V) are safe to copy/delete freely (i.e., by substitution)

I don’t care how you pick these

…but be consistent

Substitution $\{V/x\}$ is only defined for these “values”

Substitution of “values” V performed by **let**s

$$(\beta\,\mathbf{let}) \qquad (\mathbf{let}\,x = V\,\mathbf{in}\,M) = M\{V/x\}$$

$$(\eta\,\mathbf{let}) \qquad (\mathbf{let}\,x = M\,\mathbf{in}\,x) = M \qquad \text{if } x \notin FV(M)$$

lets not associated with specific type, but might decide using types

Universally safe β and η

Via the substitution principle, from **let**

For functions,

$$(\beta\lambda) \qquad (\lambda x.M)\,N = (\mathbf{let}\,x = N\,\mathbf{in}\,M)$$

$$(\eta\lambda) \qquad \lambda x.\,y\,x = y : A \to B \qquad (\text{if } x \neq y)$$

Universally safe β and η

Via the substitution principle, from let

For functions,

$$(\beta\lambda) \qquad (\lambda x.M)\, N = (\mathbf{let}\, x = N\, \mathbf{in}\, M)$$

$$(\eta\lambda) \qquad \lambda x.\, y\, x = y : A \to B \qquad (\text{if } x \neq y)$$

For case-analysis on data,

$$(\beta\,\mathbf{case}) \qquad \begin{array}{l} \mathbf{case}\ \mathsf{K}(M \ldots)\ \mathbf{of} \\ \quad \mathsf{K}(x \ldots) \to N \\ \quad \ldots \end{array} = \begin{array}{l} \mathbf{let}\, x = M \ldots \\ \mathbf{in}\, N \end{array}$$

$$(\eta\,\mathbf{case}) \qquad \begin{array}{l} \mathbf{case}\, M\, \mathbf{of} \\ \quad \mathsf{K}_1(x_1 \ldots) \to \mathsf{K}_1(x_1 \ldots) \\ \quad \ldots \\ \quad \mathsf{K}_n(x_n \ldots) \to \mathsf{K}_n(x_n \ldots) \end{array} = M : \bigoplus_{i=1}^{n} K_i(A_i \ldots)$$

Attack of the Commuting Conversions

$$\textbf{let}\ y = (\textbf{let}\ x = M\ \textbf{in}\ N)\ \textbf{in}\ P = \textbf{let}\ x = M\ \textbf{in}\ (\textbf{let}\ y = N\ \textbf{in}\ P)$$

$$(\textbf{let}\ x = M\ \textbf{in}\ N)\ P = \textbf{let}\ x = M\ \textbf{in}\ (N\ P)$$

$$\begin{array}{l}\textbf{case}\ (\textbf{let}\ x = M\ \textbf{in}\ N)\ \textbf{of}\\ \quad p_1 \to P_1 \ldots\\ \quad \ldots\\ \quad p_n \to P_n \ldots\end{array} = \begin{array}{l}\textbf{let}\ x = M\\ \textbf{in}\ \begin{pmatrix}\textbf{case}\ N\ \textbf{of}\\ \quad p_1 \to P_1 \ldots\\ \quad \ldots\\ \quad p_n \to P_n \ldots\end{pmatrix}\end{array}$$

$$\begin{array}{l}\textbf{let}\ x = \begin{pmatrix}\textbf{case}\ M\ \textbf{of}\\ \quad p_1 \to N\\ \quad \ldots\\ \quad p_n \to N_n\end{pmatrix}\\ \textbf{in}\ P\end{array} = \begin{array}{l}\textbf{case}\ M\ \textbf{of}\\ \quad p_1 \to (\textbf{let}\ x = N_1\ \textbf{in}\ P)\\ \quad \ldots\\ \quad p_n \to (\textbf{let}\ x = N_n\ \textbf{in}\ P)\end{array} \qquad (???)$$

$$\ldots$$

Big oof.....

Evaluation Contexts on the Move

Sometimes evaluation contexts need to move around

Even if you don't have control effects

Tell me which contexts are strict on their input

(Don't worry, I won't be offended)

And let's write the code to move them to where they're needed

Evaluation Contexts on the Move

What Does a Co-Variable Stand For?

Sometimes evaluation contexts need to move around

Even if you don't have control effects

Tell me which contexts are strict on their input

(Don't worry, I won't be offended)

And let's write the code to move them to where they're needed

Strucural substitution of "evaluation contexts" E performed by μ

$$(\beta\mu) \quad \langle E[\mu\beta.c] \| \alpha\rangle = c\,\{\langle E[N]\|\alpha\rangle / \langle N \| \beta\rangle\}$$

$$(\eta\mu) \quad \mu\alpha.\langle M \| \alpha\rangle = M \qquad (\text{if } \alpha \notin FV(M))$$

The One commmunting conversion to Rule Them All

$$(\mu\,\mathbf{let}) \qquad \langle \mathbf{let}\, x = M\, \mathbf{in}\, N \| \alpha \rangle = \langle M \| \mathbf{let}\, x\, \mathbf{in}\, \langle N \| \alpha \rangle \rangle$$

$$(\mu\,\mathbf{case}) \qquad \left\langle \begin{array}{l} \mathbf{case}\, M\, \mathbf{of} \\ \quad p_1 \to N_1 \\ \quad \ldots \\ \quad p_n \to N_n \end{array} \middle\| \alpha \right\rangle = \left\langle M \middle\| \begin{array}{r} \mathbf{case}\, p_1 \to \langle N_1 \| \alpha \rangle \\ \ldots \\ p_n \to \langle N_n \| \alpha \rangle \end{array} \right\rangle$$

Done!!!

The Full Axiomatization

For a generic functional language with any effects, any eval strategy

Pick your own definition of V and E

$$
\begin{aligned}
&(\beta\,\mathbf{let}) & (\mathbf{let}\,x = V\,\mathbf{in}\,M) &= M\{V/x\} \\
&(\beta\lambda) & (\lambda x.M)\;N &= (\mathbf{let}\,x = N\,\mathbf{in}\,M) \\
&(\beta\,\mathbf{case}) & \mathbf{case}\;\mathsf{K}(M\ldots)\;\mathbf{of}\;\mathsf{K}(x\ldots) \to N\ldots &= (\mathbf{let}\,x = M\ldots\,\mathbf{in}\,N) \\
&(\beta\mu) & \langle E[\mu\beta.c] \| \alpha\rangle &= c\,\{\langle E[N]\|\alpha\rangle/\langle N\|\beta\rangle\}
\end{aligned}
$$

$$
\begin{aligned}
&(\eta\,\mathbf{let}) & (\mathbf{let}\,x = M\,\mathbf{in}\,x) &= M \\
&(\eta\lambda) & \lambda x.y\;x &= y : A \to B \\
&(\eta\,\mathbf{case}) & \mathbf{case}\,M\,\mathbf{of}\;\mathsf{K}(x\ldots) \to \mathsf{K}(x\ldots)\ldots &= M : \bigoplus \mathsf{K}(A\ldots) \\
&(\eta\mu) & \mu\alpha.\langle M\|\alpha\rangle &= M
\end{aligned}
$$

$$
\begin{aligned}
&(\mu\,\mathbf{let}) & \langle \mathbf{let}\,x = M\,\mathbf{in}\,N \| \alpha\rangle &= \langle M \| \mathbf{let}\,x\,\mathbf{in}\,\langle N\|\alpha\rangle\rangle \\
&(\mu\,\mathbf{case}) & \langle \mathbf{case}\,M\,\mathbf{of}\,p \to N\ldots \| \alpha\rangle &= \langle M \| \mathbf{case}\,p \to \langle N\|\alpha\rangle\ldots\rangle
\end{aligned}
$$

Sound (for any effects) and Complete[1] (up to effect-specific laws)

[1] With respect to classical/intuitionistic sequent calculus and CPS for CBV, CBN

But! But!!! What about. . . ?!

$\eta_v \Longleftarrow \beta\,\textbf{let}\,, \eta\lambda, \eta\,\textbf{let}$

ANF, naming, $\Longleftarrow \eta\mu, \mu\,\textbf{let}\,\beta\mu, \eta\mu, \eta\,\textbf{let}$

$\beta_\Omega \Longleftarrow \beta\lambda, \eta\mu, \mu\,\textbf{let}\,\beta\mu, \eta\mu, \eta\,\textbf{let}$

commuting conversions $\Longleftarrow \eta\mu, \beta\mu, \mu\,\textbf{let}\,/\mu\,\textbf{case}$

inversion $\Longleftarrow \beta\,\textbf{let}$, commuting conversion, $\eta\,\textbf{case}\,, \beta\,\textbf{let}$

$\lambda\mu$-style bubbling capture $\Longleftarrow \eta\mu, \beta\mu$

Particularly Polite Effects

And extra privileges for upstanding citizens

$$(\text{commute})\qquad \begin{array}{l}\textbf{let}\ x = M\ \textbf{in}\\ \textbf{let}\ y = N\ \textbf{in}\ P\end{array} = \begin{array}{l}\textbf{let}\ y = N\ \textbf{in}\\ \textbf{let}\ x = M\ \textbf{in}\ P\end{array}$$

$$(\text{delete})\qquad (\textbf{let}\ \_ = M\ \textbf{in}\ N) = N$$

$$(\text{copy})\qquad \begin{array}{l}\textbf{let}\ x = M\ \textbf{in}\\ \textbf{let}\ y = M\ \textbf{in}\ P\end{array} = (\textbf{let}\ x = M\ \textbf{in}\ P\{x/y\})$$

Particularly Polite Effects

And extra privileges for upstanding citizens

$$\text{(commute)} \qquad \begin{array}{l}\textbf{let}\, x = M\, \textbf{in} \\ \textbf{let}\, y = N\, \textbf{in}\, P\end{array} = \begin{array}{l}\textbf{let}\, y = N\, \textbf{in} \\ \textbf{let}\, x = M\, \textbf{in}\, P\end{array}$$

$$\text{(delete)} \qquad (\textbf{let}\, \_ = M\, \textbf{in}\, N) = N$$

$$\text{(copy)} \qquad \begin{array}{l}\textbf{let}\, x = M\, \textbf{in} \\ \textbf{let}\, y = M\, \textbf{in}\, P\end{array} = (\textbf{let}\, x = M\, \textbf{in}\, P\,\{x/y\})$$

Of course, these hold for any effect with CBN **let**

Equational Reasoning about Abstract Machines

And deriving definitional "compilation"

Already have

$$(\mu\,\mathbf{let}) \qquad \langle \mathbf{let}\, x = M\, \mathbf{in}\, N \| \alpha \rangle = \langle M \| \mathbf{let}\, x\, \mathbf{in}\, \langle N \| \alpha \rangle \rangle$$

$$(\mu\,\mathbf{case}) \qquad \left\langle \begin{array}{l} \mathbf{case}\, M\, \mathbf{of} \\ \quad p_1 \rightarrow N_1 \\ \quad \dots \\ \quad p_n \rightarrow N_n \end{array} \middle\| \alpha \right\rangle = \left\langle M \middle\| \begin{array}{l} \mathbf{case}\, p_1 \rightarrow \langle N_1 \| \alpha \rangle \\ \qquad \dots \\ \qquad p_n \rightarrow \langle N_n \| \alpha \rangle \end{array} \right\rangle$$

Just need to push other evaluation frames on the stack

$$(\mu\lambda) \langle M\ N \| \alpha \rangle = \langle M \| N \cdot \alpha \rangle$$

Property
Given any source term N, there is a machine command c, such that $\langle N \| \alpha \rangle = c$ via β, η, μ

Inductive and Coinductive Principles

Restoring duality of equational reasoning

Induction is reasoning on the structure of values

- An enhancement of plain **case**-based inversion
- Add an inductive hypothesis for every recursive sub-tree

Coinduction is reasoning on the structure of contexts

- An enhancement of plain η-expansion of objects
- Add a co-inductive hypothesis for every recursive sub-tree

Beware! Undisciplined (co)induction leads to unsafe η "laws"

- Induction is always OK in CBV
- Otherwise, only induct over an x in contexts strict on x
- Co-induction is always OK in CBN
- Otherwise, only co-induct on a context α when given a value (i.e., productivity)

Maximizing η: The Best Strategy

There's a conflict, balancing substitution ($\beta\,\mathbf{let}$, $\beta\mu$)

$\eta\lambda$ is strongest when V is biggest (CBN)

$\eta\,\mathbf{case}$ is strongest when E is biggest (CBV)

Why not both? Polarity

Use the type of M to decide if it is a substitutable V

Use context's type to decide if it's a substitutable E

If **let** s make you squeamish, use an unambiguous **case**

$\mathbf{case}\,M\,\mathbf{of}\ \text{return}\,x \to N$ (ordinary **data** $\text{Id}\,A = \text{return}\,A$)

$=$

$\mathbf{do}\,x \leftarrow M; N$ (monadic)

$=$

$M\,\mathbf{to}\,x.N$ (*CBPV*)

Warmup: Modeling The State Monad

PL Theorists Hate Him! One Weird Trick To Control Your State

Equations for state with (dynamic) allocation

$$
\begin{aligned}
\textbf{alloc}\ V\ \textbf{in}\ E[\mathrm{get}()] &= \textbf{alloc}\ V\ \textbf{in}\ E[V] \\
\textbf{alloc}\ V\ \textbf{in}\ E[\mathrm{put}(V')] &= \textbf{alloc}\ V'\ \textbf{in}\ E[()] \\
\textbf{alloc}\ V\ \textbf{in}\ \mathrm{return}\ V' &= (V, V')
\end{aligned}
$$

Warmup: Modeling The State Monad

PL Theorists Hate Him! One Weird Trick To Control Your State

Equations for state with (dynamic) allocation

$$\begin{aligned} \textbf{alloc}\, V\, \textbf{in}\, E[\text{get}()] &= \textbf{alloc}\, V\, \textbf{in}\, E[V] \\ \textbf{alloc}\, V\, \textbf{in}\, E[\text{put}(V')] &= \textbf{alloc}\, V'\, \textbf{in}\, E[()] \\ \textbf{alloc}\, V\, \textbf{in}\ \text{return}\ V' &= (V, V') \end{aligned}$$

Representing the state monad using $\Lambda\mu$ (thanks Filinski)

$$\begin{aligned} \textbf{alloc}\, M\, \textbf{in}\, N &= \langle N \| \textbf{case}\ \text{return}\, x\, \textbf{in}\, \lambda s.(x, s)\rangle\ M \\ \text{get}() &= \mu\alpha.\lambda s.\langle s \| \alpha\rangle\ s \\ \text{put}(s) &= \mu\alpha.\lambda\_.\langle () \| \alpha\rangle\ s \end{aligned}$$

Warmup 2: Modeling The State Handler

Move the interpretation out of the operation

Monadic state represented via reflection/reification

$$\textbf{alloc}\, M\, \textbf{in}\, N = \langle N \| \textbf{case}\ \text{return}\, x\, \textbf{in}\, \lambda s.(x, s)\rangle\ M$$

$$\text{get}() = \mu\alpha.\lambda s.\langle s \| \alpha\rangle\ s$$

$$\text{put}(s) = \mu\alpha.\lambda\_.\langle () \| \alpha\rangle\ s$$

Representing the state handler using $\Lambda\mu$

$$\textit{handleState}\ (\text{get}() \cdot \alpha)\ s = \textit{handleState}\ \langle s \| \alpha\rangle\ s$$

$$\textit{handleState}\ (\text{put}(s') \cdot \alpha)\ s = \textit{handleState}\ \langle () \| \alpha\rangle\ s'$$

$$\textit{handleState}\ \ \text{return}(x)\ s = (x, s)$$

$$\textbf{alloc}\, M\, \textbf{in}\, N = \textit{handleState}\ \langle N \| \textbf{case}\ \text{return}(x) \to \text{return}(x)\rangle\ M$$

$$\textbf{do}\ \text{get}() = \mu\alpha.(\text{get}() \cdot \alpha)$$

$$\textbf{do}\ \text{put}(s) = \mu\alpha.(\text{put}(s) \cdot \alpha)$$

Modeling a Theory For Effect Handlers

Just a sketch, use at your own risk

Shallow

$$\textbf{handle}\, M\, \textbf{with}\, H = H\,\langle M \| \textbf{case}\ \mathsf{return}\, x \to \mathsf{return}\, x\rangle$$

$$\textbf{do}\ \mathsf{Op}\, x = \mu\alpha.(\mathsf{Op}\, x \cdot \alpha)$$

Deep

$$\textbf{handle}\, M\, \textbf{with}\, H = \langle\langle M \| \textbf{case}\ \mathsf{return}\, x \to \mathsf{return}\, x\rangle \| H\rangle$$

$$\textbf{do}\ \mathsf{Op}\, x = \mu\alpha.\mu\beta.\langle \mathsf{Op}\, x \cdot \textbf{let}\, y\, \textbf{in}\ \langle\langle y \| \alpha\rangle \| \beta\rangle \| \beta\rangle$$

Limited resumption

$$\textbf{handle}\, M\, \textbf{with}\, H = \langle\langle M \| \textbf{case}\ \mathsf{return}\, x \to \mathsf{return}\, x\rangle \| H\rangle$$

$$\textbf{do}\ \mathsf{Op}\, x = \mu\alpha.\mu\beta.\,\textbf{case}\ \langle \mathsf{Op}\, x \| \beta\rangle\ \textbf{of}$$

$$\mathsf{resume}\, y \to \langle\langle y \| \alpha\rangle \| \beta\rangle$$

$$\mathsf{return}\, z \to z$$

Handling Many Things

There's a big design space

Lexical handlers

Stacks of commands, indexed by μ

Multi-handlers

One handler with multiple sub-commands

Dynamic handlers

Multiple promtps, of course

But what about my η, though ¯_(ツ)_/¯