

Command = $\langle \text{Value} | \text{Context} \rangle$

Hans-Dieter Hiep

15th of June, 2018

Example

Consider a total functional programming language.

`swap` :: $A \wedge B \rightarrow B \wedge A$

`swap` $(a, b) = (b, a)$

`paws` :: $A \vee B \rightarrow B \vee A$

`paws` $[a] = [a]$

`paws` $[b] = [b]$

But, can we find a program with the following type?

`toll` :: $(\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$

What does \neg and `toll` even mean?

Background

Simply typed lambda calculus

Let s, t be terms, defined by the grammar:

$$s, t ::= x \mid (\lambda x.t) \mid (st)$$

where x is a variable. Let A, B be formulas:

$$A, B ::= X \mid (A \rightarrow B)$$

where X is a propositional variable.

$$\frac{}{\Gamma, x : A \vdash x : A} \quad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x.t) : A \rightarrow B} \quad \frac{\Gamma \vdash s : A \rightarrow B \quad \Gamma \vdash t : A}{\Gamma \vdash (st) : B}$$

Type system (corresponding to [minimal logic](#)).

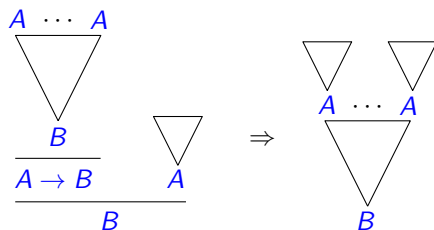
Background

Curry-Howard isomorphism

Simply-typed λ -calculus has β -reduction:

$$(\lambda x.s)t \longrightarrow_{\beta} s[x \mapsto t]$$

corresponding to detour elimination:



Call-by-value evaluation

Let t be a term. We designate a subset of terms

$$v ::= x \mid (\lambda x.t)$$

which we call values. We define **evaluation contexts**:

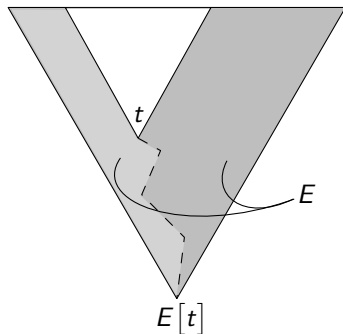
$$E ::= \square \mid (vE) \mid (Et)$$

By allowing β -reduction only in this context, we obtain leftmost-outermost **call-by-value evaluation**:

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

we use \mapsto to denote a rewrite step *at the root only*.

Call-by-value evaluation



Call-by-value evaluation

$$E ::= \square \mid (vE) \mid (Et)$$

Every term is either a value, or it is equal to $E[s]$ for some s .
We say that a term t is reducible if $t = E[(\lambda x.s)v]$

Example

Reduce using call-by-value evaluation:

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

Call-by-value evaluation

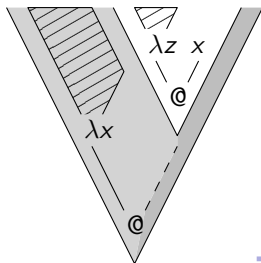
$$E ::= \square \mid (vE) \mid (Et)$$

Every term is either a value, or it is equal to $E[s]$ for some s .
We say that a term t is reducible if $t = E[(\lambda x.s)v]$

Example

Reduce using call-by-value evaluation:

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



Call-by-value evaluation

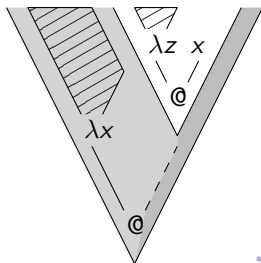
$$E ::= \square \mid (vE) \mid (Et)$$

Every term is either a value, or it is equal to $E[s]$ for some s .
We say that a term t is reducible if $t = E[(\lambda x.s)v]$

Example

Reduce using call-by-value evaluation:

$$(\lambda x.(\lambda y.x)x)((\lambda z.y)x)$$
$$E = (\lambda x.(\lambda y.x)x)\square$$



Call-by-value evaluation

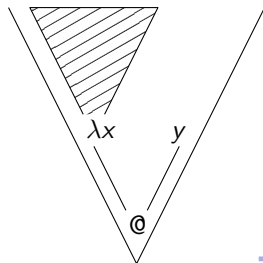
$$E ::= \square \mid (vE) \mid (Et)$$

Every term is either a value, or it is equal to $E[s]$ for some s .
We say that a term t is reducible if $t = E[(\lambda x.s)v]$

Example

Reduce using call-by-value evaluation:

$$\begin{aligned} & (\lambda x.(\lambda y.x)x)((\lambda z.y)x) \\ E = & (\lambda x.(\lambda y.x)x)\square \\ & (\lambda x.(\lambda y.x)x)y \end{aligned}$$



Call-by-value evaluation

$$E ::= \square \mid (vE) \mid (Et)$$

Every term is either a value, or it is equal to $E[s]$ for some s .
We say that a term t is reducible if $t = E[(\lambda x.s)v]$

Example

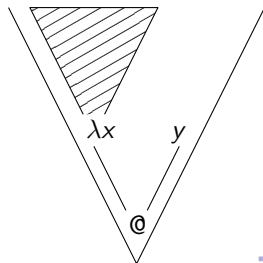
Reduce using call-by-value evaluation:

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

$$E = (\lambda x.(\lambda y.x)x)\square$$

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

$$E = \square$$



Call-by-value evaluation

$$E ::= \square \mid (vE) \mid (Et)$$

Every term is either a value, or it is equal to $E[s]$ for some s .
We say that a term t is reducible if $t = E[(\lambda x.s)v]$

Example

Reduce using call-by-value evaluation:

$$\begin{aligned} & (\lambda x.(\lambda y.x)x)((\lambda z.y)x) \\ E = & (\lambda x.(\lambda y.x)x)\square \\ & (\lambda x.(\lambda y.x)x)y \\ E = & \square \\ & (\lambda z.y)y \\ & y \end{aligned}$$

$\lambda\mathcal{C}$ -calculus

- ▶ first formulated in Felleisen's Ph.D. thesis (1987)
- ▶ corresponds to classical propositional logic (Griffin, 1990)

Let s, t be terms, and E be an evaluation context:

$$\begin{aligned} s, t &::= x \mid (\lambda x. t) \mid (st) \mid (\mathcal{C}t) \\ E &::= \square \mid (vE) \mid (Et) \end{aligned}$$

We now have call-by-value evaluation:

$$\begin{aligned} E[(\lambda x. s)v] &\mapsto E[s[x \mapsto v]] \\ E[\mathcal{C}t] &\mapsto t(\lambda x. \mathcal{A}E[x]) \end{aligned}$$

where $(\mathcal{A}t) := (\mathcal{C}\lambda y. t)$ for fresh y not free in t .

$$E[\mathcal{A}t] \mapsto t$$

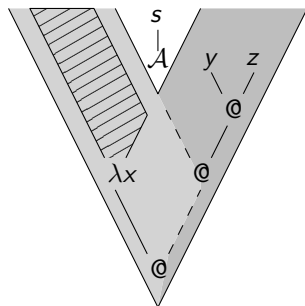
$\lambda\mathcal{C}$ -calculus: intuition

Abort

Example

Reduce using call-by-value evaluation:

$(\lambda x.x)((\mathcal{A}s)(yz))$



$\lambda\mathcal{C}$ -calculus: intuition

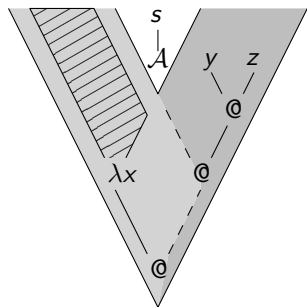
\mathcal{A} abort

Example

Reduce using call-by-value evaluation:

$$(\lambda x.x)((\mathcal{A}s)(yz))$$

$$E = (\lambda x.x)(\square(yz))$$



$\lambda\mathcal{C}$ -calculus: intuition

\mathcal{A} bort

Example

Reduce using call-by-value evaluation:

$$(\lambda x.x)((\mathcal{A}s)(yz))$$
$$E = (\lambda x.x)(\square(yz))$$

s

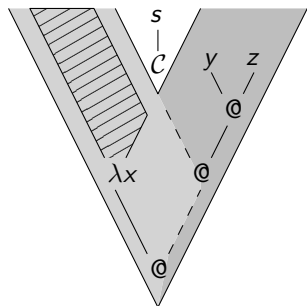
$\lambda\mathcal{C}$ -calculus: intuition

Control

Example

Reduce using call-by-value evaluation:

$(\lambda x.x)((\mathcal{C}s)(yz))$



$\lambda\mathcal{C}$ -calculus: intuition

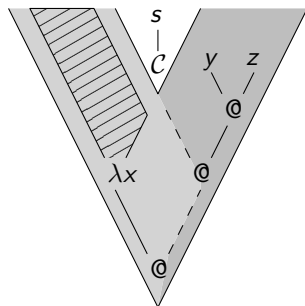
Control

Example

Reduce using call-by-value evaluation:

$$(\lambda x.x)((\mathcal{C}s)(yz))$$

$$E = (\lambda x.x)(\square(yz))$$



$\lambda\mathcal{C}$ -calculus: intuition

Control

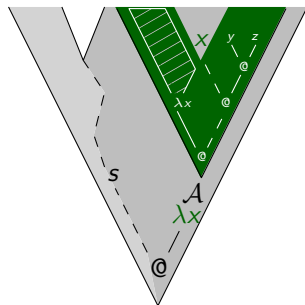
Example

Reduce using call-by-value evaluation:

$$(\lambda x.x)((\mathcal{C}s)(yz))$$

$$E = (\lambda x.x)(\square(yz))$$

$$s(\lambda x.\mathcal{A}E[x])$$



$\lambda\mathcal{C}$ -calculus: intuition

Control

Example

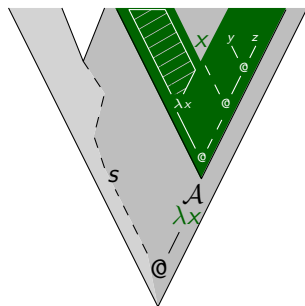
Reduce using call-by-value evaluation:

$$(\lambda x.x)((\mathcal{C}s)(yz))$$

$$E = (\lambda x.x)(\square(yz))$$

$$s(\lambda x.\mathcal{A}E[x])$$

s is a function taking a **continuation**



$\lambda\mathcal{C}$ -calculus: intuition

Control

Example

Reduce using call-by-value evaluation:

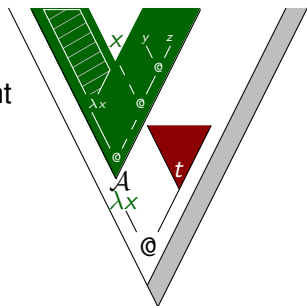
$$(\lambda x.x)((\mathcal{C}s)(yz))$$

$$E = (\lambda x.x)(\square(yz))$$

$$s(\lambda x.\mathcal{A}E[x])$$

s is a function taking a **continuation**
calling continues the original environment

$$\dots(\lambda x.\mathcal{A}E[x])t\dots$$



$\lambda\mathcal{C}$ -calculus: intuition

Control

Example

Reduce using call-by-value evaluation:

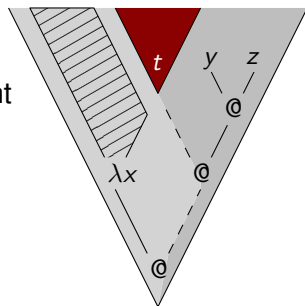
$(\lambda x.x)((\mathcal{C}s)(yz))$

$E = (\lambda x.x)(\square(yz))$
 $s(\lambda x.\mathcal{A}E[x])$

s is a function taking a **continuation**
calling continues the original environment

$\dots(\lambda x.\mathcal{A}E[x])t\dots$
 $(\lambda x.x)(t \quad (yz))$

with argument in place of \mathcal{C}



$\lambda\mathcal{C}$ -calculus: expressivity

$s, t ::= x \mid (\lambda x.t) \mid (st)$ (minimal)

$s, t ::= x \mid (\lambda x.t) \mid (st) \mid (\mathcal{A}t)$ (intuitionistic)

$s, t ::= x \mid (\lambda x.t) \mid (st) \mid (\mathcal{C}t)$ (classical)

We have that (Ariola&Herbelin, 2003):

- ▶ \mathcal{A}, \mathcal{C} are not definable in minimal
- ▶ \mathcal{C} is not definable in intuitionistic
- ▶ \mathcal{A} is definable in classical

$\lambda\mathcal{C}$ -calculus: types

Let A, B be formulas:

$$A, B ::= X \mid \perp \mid (A \rightarrow B)$$

where X is a propositional variable. Let $\neg A$ abbreviate $(A \rightarrow \perp)$.

$$\frac{}{\Gamma, x : A \vdash x : A} \quad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x.t) : A \rightarrow B} \quad \frac{\Gamma \vdash s : A \rightarrow B \quad \Gamma \vdash t : A}{\Gamma \vdash (st) : B}$$

$$\frac{\Gamma \vdash t : \perp}{\Gamma \vdash (\mathcal{A}t) : A} \quad \frac{\Gamma \vdash t : \neg\neg A}{\Gamma \vdash (\mathcal{C}t) : A}$$

$\lambda\mathcal{C}$ -calculus: example

Modus tollens

$\lambda x.\lambda y.\mathcal{C}\lambda z.(xz)y : (\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$

$\lambda\mathcal{C}$ -calculus: example

Modus tollens

$\lambda x.\lambda y.\mathcal{C}\lambda z.(xz)y : (\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$

$$\frac{\frac{\frac{(\neg A \rightarrow \neg B)^x \quad (\neg A)^z}{\neg B} \text{ @} \quad B^y}{\perp} \text{ @}}{\frac{\frac{\frac{\frac{\frac{\frac{\perp}{\neg\neg A} \lambda z}{A} \mathcal{C}}{B \rightarrow A} \lambda y}}{(\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)} \lambda x}}$$

$\lambda\mathcal{C}$ -calculus: example

Modus tollens

$\lambda x.\lambda y.\mathcal{C}\lambda z.(xz)y : (\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$

$(\lambda xy.\mathcal{C}\lambda z.(xz)y)fb \mapsto$

$(\lambda y.\mathcal{C}\lambda z.(fz)y)b \mapsto$

$\mathcal{C}\lambda z.(fz)b \mapsto$

$(\lambda z.(fz)b)(\lambda x.Ax) \mapsto$

$(f(\lambda x.Ax))b$

$\lambda\mathcal{C}$ -calculus: example

Modus tollens

$\lambda x.\lambda y.\mathcal{C}\lambda z.(xz)y : (\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$

$(\lambda xy.\mathcal{C}\lambda z.(xz)y)fb \mapsto$

$(\lambda y.\mathcal{C}\lambda z.(fz)y)b \mapsto$

$\mathcal{C}\lambda z.(fz)b \mapsto$

$(\lambda z.(fz)b)(\lambda x.Ax) \mapsto$

$(f(\lambda x.Ax))b$

Note that $(\lambda xy.\mathcal{C}\lambda z.(xz)y)fb : A$.

We expect $f : \neg A \rightarrow B \rightarrow \perp$ and $b : B$.

Hence, $(\lambda x.Ax) : \neg A$ and thus $x : A$.

$\lambda\mathcal{C}$ -calculus: example

Modus tollens

$\lambda x.\lambda y.\mathcal{C}\lambda z.(xz)y : (\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$

$(\lambda xy.\mathcal{C}\lambda z.(xz)y)fb \mapsto$

$(\lambda y.\mathcal{C}\lambda z.(fz)y)b \mapsto$

$\mathcal{C}\lambda z.(fz)b \mapsto$

$(\lambda z.(fz)b)(\lambda x.Ax) \mapsto$

$(f(\lambda x.Ax))b$

Note that $(\lambda xy.\mathcal{C}\lambda z.(xz)y)fb : A$.

We expect $f : \neg A \rightarrow B \rightarrow \perp$ and $b : B$.

Hence, $(\lambda x.Ax) : \neg A$ and thus $x : A$.

But Ax requires $x : \perp$! No **type preservation**?

$\lambda\mathcal{C}$ -calculus: implicit top-level

The problem lies in the fact that reducing \mathcal{C} forces a conversion from \perp to the top-level type.

To solve this problem, Griffin proposed to consider evaluation of $t : A$ within $\mathcal{C}(\lambda o.ot)$ where $o : \neg A$ and $ot : \perp$.

$$\mathcal{C}(\lambda o.E[(\lambda x.s)v]) \mapsto \mathcal{C}(\lambda o.E[s[x \mapsto v]])$$

$$\mathcal{C}(\lambda o.E[\mathcal{C}t]) \mapsto \mathcal{C}(\lambda o.t(\lambda x.AE[x]))$$

$$\mathcal{C}(\lambda o.ov) \mapsto v$$

Logically, we work under a detour, removing it at last:

$$\frac{\frac{\frac{(\neg A)^x \quad A}{\perp}}{\neg\neg A} \lambda x}{A} \mathcal{C} \Rightarrow A$$

$\lambda\mathcal{C}$ -calculus: example

Modus tollens

$\lambda x.\lambda y.\mathcal{C}\lambda z.(xz)y : (\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$

$\mathcal{C}(\lambda o.o(\lambda xy.\mathcal{C}\lambda z.(xz)y)fb) \mapsto$

$\mathcal{C}(\lambda o.o(\lambda y.\mathcal{C}\lambda z.(fz)y)b) \mapsto$

$\mathcal{C}(\lambda o.o\mathcal{C}\lambda z.(fz)b) \mapsto$

$\mathcal{C}(\lambda o.(\lambda z.(fz)b)(\lambda x.\mathcal{A}(ox))) \mapsto$

$\mathcal{C}(\lambda o.(f(\lambda x.\mathcal{A}(ox)))b)$

We now verify: top-level has type A . Hence $o : \neg A$.

We expect $f : \neg A \rightarrow B \rightarrow \perp$ and $b : B$.

We have $(\lambda x.\mathcal{A}(ox)) : \neg A$ since $x : A$ and $ox : \perp$

We use \mathcal{A} to derive \perp (which is useless but harmless).

From $\lambda\mathcal{C}$ to $\mu\tilde{\mu}$

$\lambda\mathcal{C}$ -calculus (Felleisen*, 1987)

Isomorphism (de Groote, 1994; Ariola&Herbelin, 2003)

$\lambda\mu$ -calculus (Parigot, 1992)

From natural deduction to sequent calculus

$\bar{\lambda}\mu$ -calculus (Herbelin*, 1995)

Duality of call-by-name and call-by-value

$\bar{\lambda}\mu\tilde{\mu}$ -calculus (Curien&Herbelin, 2000)

Generalized type connectives

$\mu\tilde{\mu}$ -calculus (Downen*, 2017)

$\mu\tilde{\mu}$ -calculus

See the report for the rest:

- ▶ Command = $\langle \text{Value} | \text{Context} \rangle$
- ▶ $\mu\tilde{\mu}$ -calculus
- ▶ Focussed sequent calculus
- ▶ Generalized type connectives

on <http://www.hansdieterhiep.nl>

Thank you for your attention!