Command = $\langle$Value|Context$\rangle$

Hans-Dieter Hiep

15th of June, 2018

Example

Consider a total functional programming language.

$$\begin{aligned}
&\texttt{swap} :: A \wedge B \to B \wedge A \\
&\texttt{swap}\ (a, b) = (b, a)
\end{aligned}$$

$$\begin{aligned}
&\texttt{paws} :: A \vee B \to B \vee A \\
&\texttt{paws}\ \lfloor a \rfloor = \lceil a \rceil \\
&\texttt{paws}\ \lceil b \rceil = \lfloor b \rfloor
\end{aligned}$$

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

$$\texttt{toll} :: (\neg A \to \neg B) \to (B \to A)$$

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:

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

where X is a propositional variable.

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

Type system (corresponding to minimal logic).

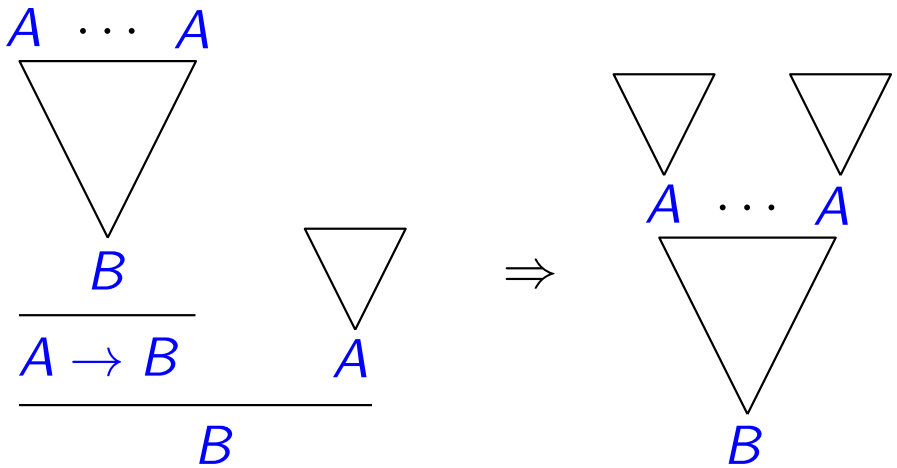
Background

Curry-Howard isomorphism

Simply-typed λ-calculus has β-reduction:

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

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**:

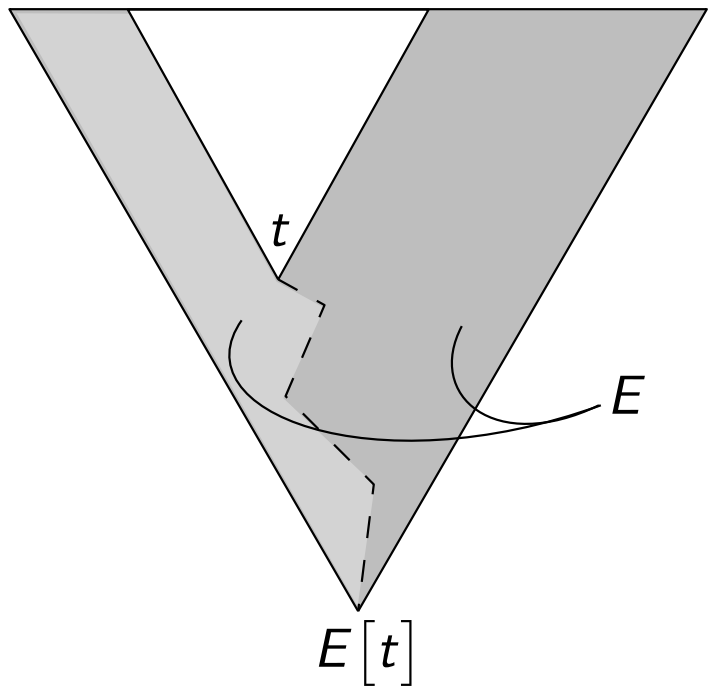
$$E ::= \Box \mid (vE) \mid (Et)$$

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

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

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

Call-by-value evaluation



Call-by-value evaluation

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

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

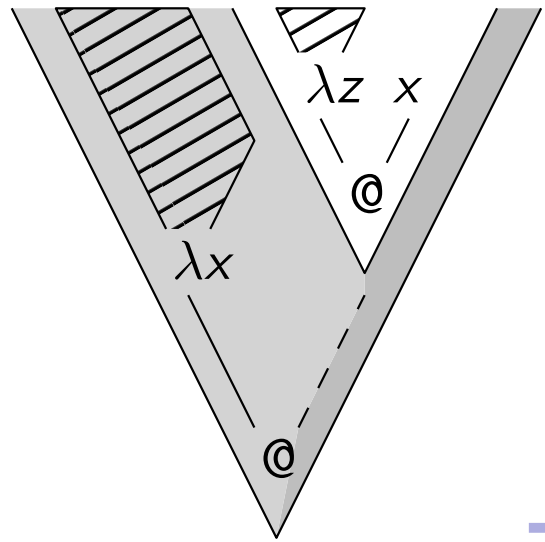
$$E ::= \Box \mid (vE) \mid (Et)$$

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

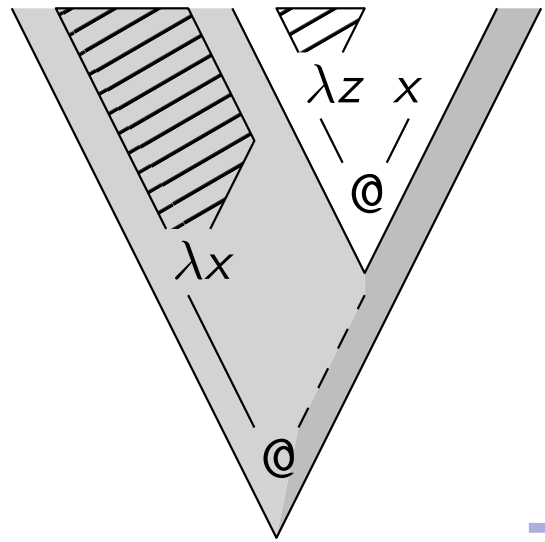
$$E ::= \Box \mid (vE) \mid (Et)$$

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:

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



Call-by-value evaluation

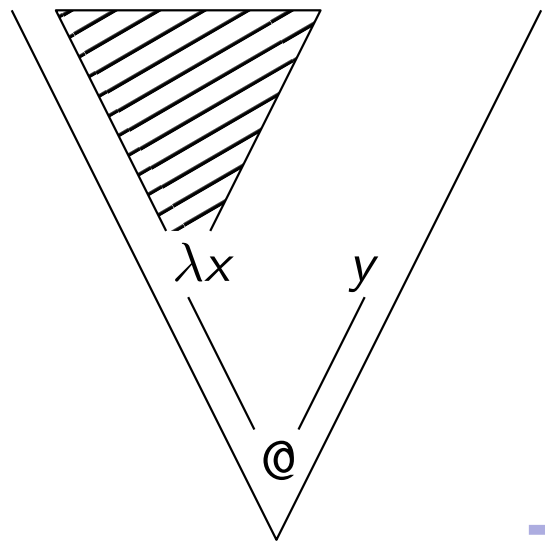
$$E ::= \Box \mid (vE) \mid (Et)$$

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:

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



Call-by-value evaluation

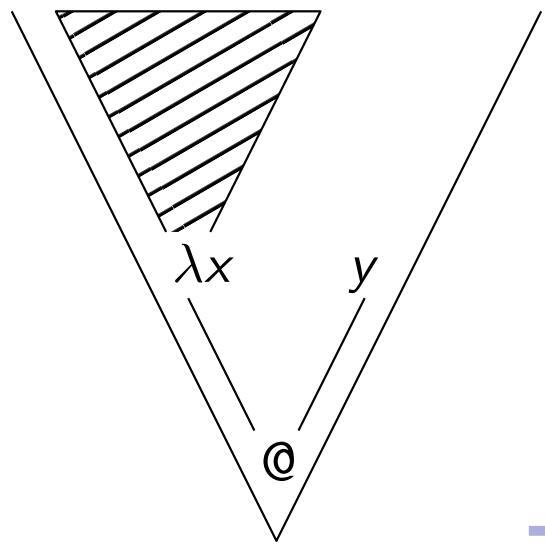
$$E ::= \Box \mid (vE) \mid (Et)$$

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:

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



Call-by-value evaluation

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

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:

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

$\lambda\mathcal{C}$-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:

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

We now have call-by-value evaluation:

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

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

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

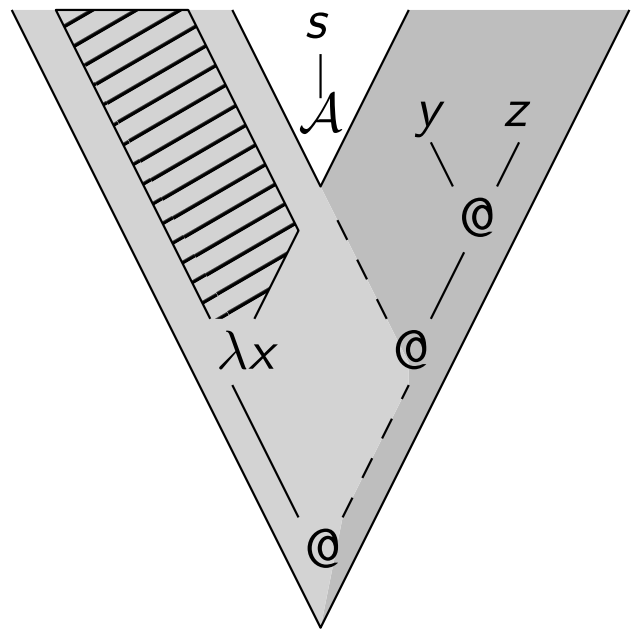
$\lambda\mathcal{C}$-calculus: intuition

$\mathcal{A}$bort

Example

Reduce using call-by-value evaluation:

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



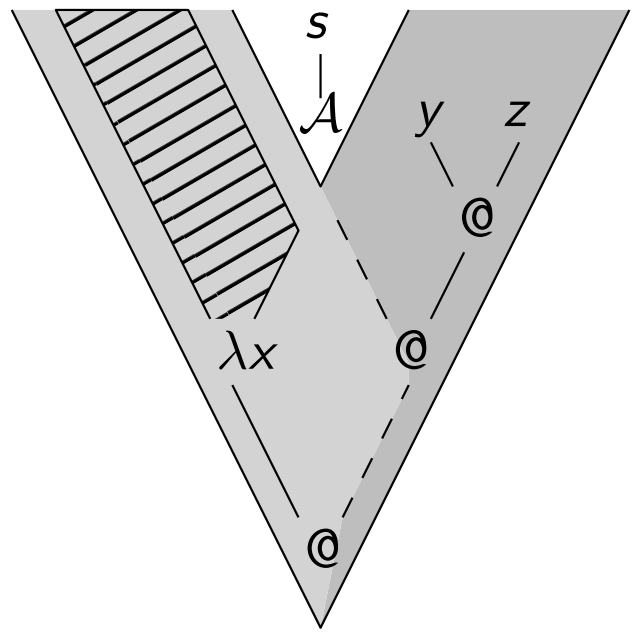
$\lambda\mathcal{C}$-calculus: intuition

$\mathcal{A}$bort

Example

Reduce using call-by-value evaluation:

$$\begin{aligned} & (\lambda x.x)((\mathcal{A}s)(yz)) \\ E = & (\lambda x.x)(\Box(yz)) \end{aligned}$$



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

$\mathcal{A}$bort

Example

Reduce using call-by-value evaluation:

$$\begin{aligned} & (\lambda x.x)((\mathcal{A}s)(yz)) \\ E = & (\lambda x.x)(\Box(yz)) \\ & s \end{aligned}$$

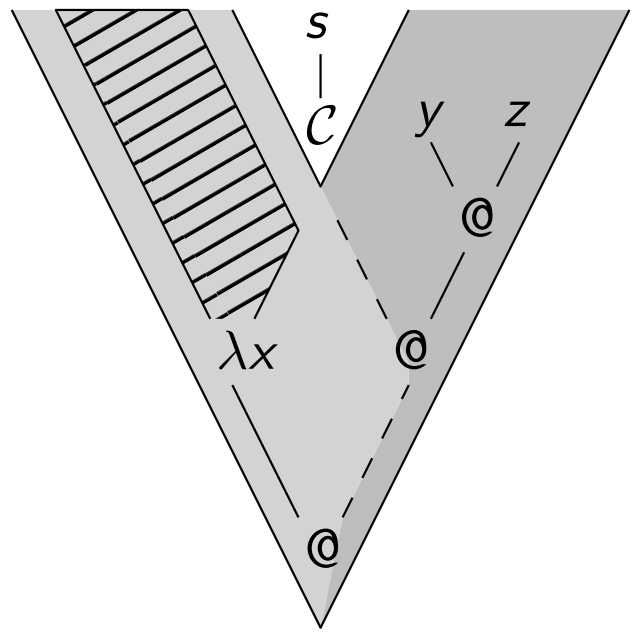
$\lambda\mathcal{C}$-calculus: intuition

$\mathcal{C}$ontrol

Example

Reduce using call-by-value evaluation:

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



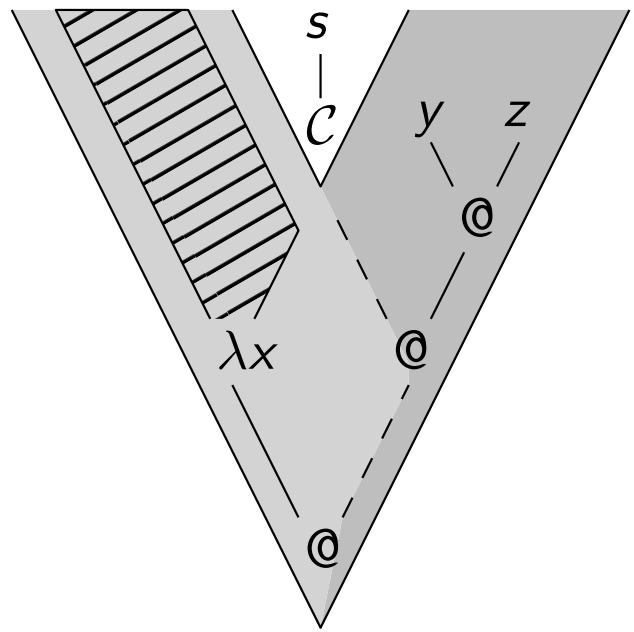
$\lambda\mathcal{C}$-calculus: intuition

$\mathcal{C}$ontrol

Example

Reduce using call-by-value evaluation:

$$\begin{aligned} & (\lambda x.x)((\mathcal{C}s)(yz)) \\ E = & (\lambda x.x)(\Box(yz)) \end{aligned}$$



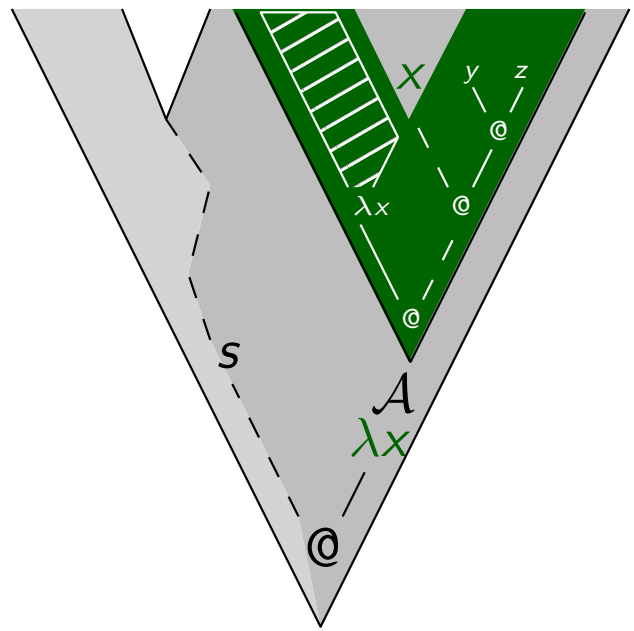
$\lambda\mathcal{C}$-calculus: intuition

$\mathcal{C}$ontrol

Example

Reduce using call-by-value evaluation:

$$\begin{aligned} & (\lambda x.x)((\mathcal{C}s)(yz)) \\ E = & (\lambda x.x)(\Box(yz)) \\ & s(\lambda x.\mathcal{A}E[x]) \end{aligned}$$



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

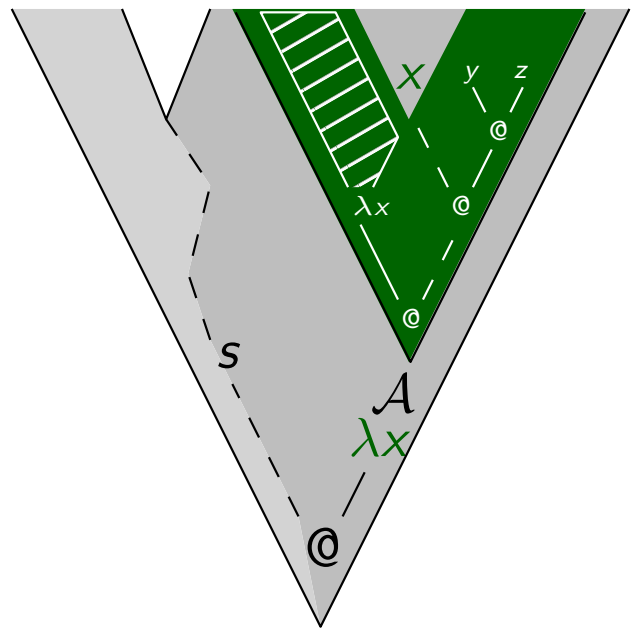
$\mathcal{C}$ontrol

Example

Reduce using call-by-value evaluation:

$$\begin{aligned} & (\lambda x.x)((\mathcal{C}s)(yz)) \\ E = & (\lambda x.x)(\Box(yz)) \\ & s(\lambda x.\mathcal{A}E[x]) \end{aligned}$$

s is a function taking a continuation



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

$\mathcal{C}$ontrol

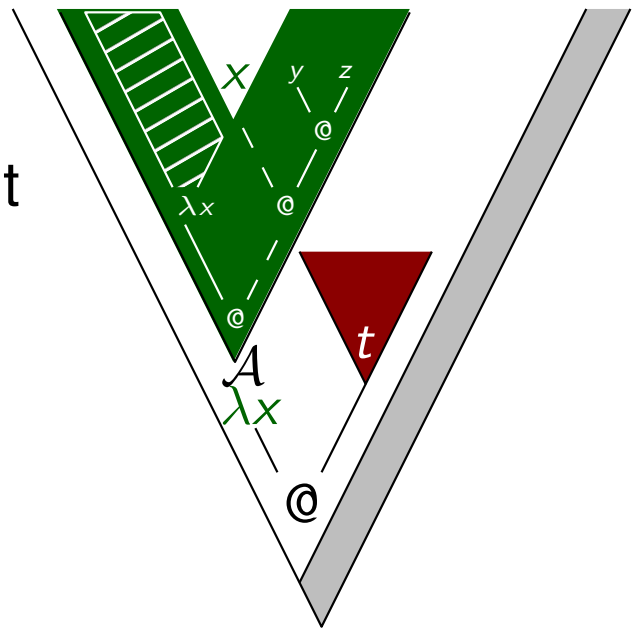
Example

Reduce using call-by-value evaluation:

$$\begin{aligned} & (\lambda x.x)((\mathcal{C}s)(yz)) \\ E = & (\lambda x.x)(\Box(yz)) \\ & s(\lambda x.\mathcal{A}E[x]) \end{aligned}$$

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

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



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

$\mathcal{C}$ontrol

Example

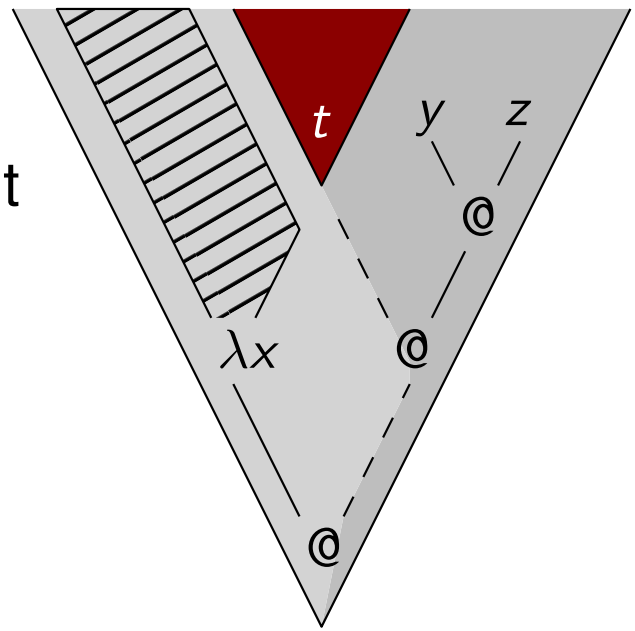
Reduce using call-by-value evaluation:

$$
\begin{array}{ll}
 & (\lambda x.x)((\mathcal{C}s)(yz)) \\
E = & (\lambda x.x)(\Box(yz)) \\
 & s(\lambda x.\mathcal{A}E[x])
\end{array}
$$

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

$$
\begin{array}{ll}
 & \ldots(\lambda x.\mathcal{A}E[x])t\ldots \\
 & (\lambda x.x)(t \quad (yz))
\end{array}
$$

with argument in place of $\mathcal{C}$



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

$$s,t ::= x \mid (\lambda x.t) \mid (st) \qquad \text{(minimal)}$$
$$s,t ::= x \mid (\lambda x.t) \mid (st) \mid (\mathcal{A}t) \qquad \text{(intuitionistic)}$$
$$s,t ::= x \mid (\lambda x.t) \mid (st) \mid (\mathcal{C}t) \qquad \text{(classical)}$$

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:

$$A, B ::= X \mid \bot \mid (A \to B)$$

where X is a propositional variable. Let $\neg A$ abbreviate $(A \to \bot)$.

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

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

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

Modus tollens

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

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

Modus tollens

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

$$\dfrac{\dfrac{\dfrac{(\neg A \to \neg B)^x \quad (\neg A)^z}{\neg B}\,@ \qquad B^y}{\bot}\,@}{\dfrac{\dfrac{\neg\neg A}{A}\,\mathcal{C}}{B \to A}\,\lambda y}\,\lambda z}{(\neg A \to \neg B) \to (B \to A)}\,\lambda x$$

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

Modus tollens

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

$$
\begin{aligned}
(\lambda xy.\mathcal{C}\lambda z.(xz)y)fb &\mapsto \\
(\lambda y.\mathcal{C}\lambda z.(fz)y)b &\mapsto \\
\mathcal{C}\lambda z.(fz)b &\mapsto \\
(\lambda z.(fz)b)(\lambda x.\mathcal{A}x) &\mapsto \\
(f(\lambda x.\mathcal{A}x))b)
\end{aligned}
$$

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

Modus tollens

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

$$
\begin{aligned}
(\lambda xy.\mathcal{C}\lambda z.(xz)y)fb &\mapsto \\
(\lambda y.\mathcal{C}\lambda z.(fz)y)b &\mapsto \\
\mathcal{C}\lambda z.(fz)b &\mapsto \\
(\lambda z.(fz)b)(\lambda x.\mathcal{A}x) &\mapsto \\
(f(\lambda x.\mathcal{A}x))b)
\end{aligned}
$$

Note that $(\lambda xy.\mathcal{C}\lambda z.(xz)y)fb : A$.
We expect $f : \neg A \to B \to \bot$ and $b : B$.
Hence, $(\lambda x.\mathcal{A}x) : \neg A$ and thus $x : A$.

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

Modus tollens

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

$$
\begin{aligned}
(\lambda xy.\mathcal{C}\lambda z.(xz)y)fb &\mapsto \\
(\lambda y.\mathcal{C}\lambda z.(fz)y)b &\mapsto \\
\mathcal{C}\lambda z.(fz)b &\mapsto \\
(\lambda z.(fz)b)(\lambda x.\mathcal{A}x) &\mapsto \\
(f(\lambda x.\mathcal{A}x))b)
\end{aligned}
$$

Note that $(\lambda xy.\mathcal{C}\lambda z.(xz)y)fb : A$.
We expect $f : \neg A \to B \to \bot$ and $b : B$.
Hence, $(\lambda x.\mathcal{A}x) : \neg A$ and thus $x : A$.
But $\mathcal{A}x$ requires $x : \bot$! No **type preservation**?

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

The problem lies in the fact that reducing $\mathcal{C}$ forces a conversion from $\bot$ 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 $ot : \bot$.

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

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

$$\dfrac{\dfrac{\dfrac{(\neg A)^x \quad A}{\bot}}{\neg\neg A}\,\lambda x}{A}\,\mathcal{C} \Rightarrow A$$

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

Modus tollens

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

$$\begin{aligned}
\mathcal{C}(\lambda o.o(\lambda xy.\mathcal{C}\lambda z.(xz)y)fb) &\mapsto \\
\mathcal{C}(\lambda o.o(\lambda y.\mathcal{C}\lambda z.(fz)y)b) &\mapsto \\
\mathcal{C}(\lambda o.o\mathcal{C}\lambda z.(fz)b) &\mapsto \\
\mathcal{C}(\lambda o.(\lambda z.(fz)b)(\lambda x.\mathcal{A}(ox))) &\mapsto \\
\mathcal{C}(\lambda o.(f(\lambda x.\mathcal{A}(ox)))b))
\end{aligned}$$

We now verify: top-level has type A. Hence $o : \neg A$.
We expect $f : \neg A \to B \to \bot$ and $b : B$.
We have $(\lambda x.\mathcal{A}(ox)) : \neg A$ since $x : A$ and $ox : \bot$
We use $\mathcal{A}$ to derive $\bot$ (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!