Operational Semantics

Defining an Operational Semantics

- Granurality
- Small-step semantics, big-step semantics
- Order of evaluation: call-by-name, call-by-value, call-by-need, perpetual strategy
- Abstract machines

Big-step Semantics

Consider a very elemental calculator handling expressions belonging to the following grammar $e ::= n \mid x \mid e + e$.
Each rule completely evaluates an expression under a substitution to a "value".

$$\frac{}{\langle n, \sigma \rangle \Downarrow n} \qquad \frac{}{\langle x, \sigma \rangle \Downarrow \sigma(x)}$$

$$\frac{\langle a_1, \sigma \rangle \Downarrow n_1 \qquad \langle a_2, \sigma \rangle \Downarrow n_2 \qquad n \text{ is "}n_1 \text{ plus } n_2\text{"}}{\langle a_1 + a_2, \sigma \rangle \Downarrow n}$$

Properties

- Abstract
- Allows to avoid details
- No specification of evaluation order (e.g. $(1 + 3) + (5 - 3)$)
- No specification of control of errors
- No specification of interleaving

Small-step Semantics

Evaluation of an expression under a substitution is given by a sequence of state changes which terminates when the state cannot be reduced further.

$$\frac{\langle a_1, \sigma\rangle \rightsquigarrow \langle a_1', \sigma'\rangle}{\langle a_1 + a_2, \sigma\rangle \rightsquigarrow \langle a_1' + a_2, \sigma'\rangle} \qquad \frac{\langle a_2, \sigma\rangle \rightsquigarrow \langle a_2', \sigma'\rangle}{\langle n_1 + a_2, \sigma\rangle \rightsquigarrow \langle n_1 + a_2', \sigma'\rangle}$$

$$\frac{}{\langle x, \sigma\rangle \rightsquigarrow \langle \sigma(x), \sigma\rangle} \qquad \frac{n \text{ is } \text{"}n_1 \text{ plus } n_2\text{"}}{\langle n_1 + n_2, \sigma\rangle \rightsquigarrow \langle n, \sigma\rangle}$$

Properties

- Less abstract
- Specification of order of evaluation
- Control of errors: $\dfrac{n_2 \neq 0}{n_1 / n_2 \rightsquigarrow n}$, where n is "n_1 divided by n_2".
- Interleaving: $\dfrac{\langle c_1, \sigma \rangle \rightsquigarrow \langle c_1', \sigma' \rangle}{\langle c_1 \| c_2, \sigma \rangle \rightsquigarrow \langle c_1' \| c_2, \sigma' \rangle}$

From Small-step to Multi-step Semantics

The multi-step semantics is given by the relation $t \rightsquigarrow^* t'$ which is the reflexive and transitive closure of $t \rightsquigarrow t'$.

(P1) $t \rightsquigarrow^* t$ for every t

(P2) $t \rightsquigarrow t'$ implies $t \rightsquigarrow^* t'$

(P3) $t \rightsquigarrow^* t'$ and $t' \rightsquigarrow^* t''$ implies $t \rightsquigarrow^* t''$

Properties of the small and big step semantics

- The relation $\rightsquigarrow$ is deterministic.
- The relation $\Downarrow$ is deterministic.
- $t \Downarrow v$ iff $t \rightsquigarrow^* v$, where v is a ”value”.

Big-step versus small-step semantics

- In small-step semantics evaluation stops at errors. In big-step semantics errors occur deeply inside derivation trees.
- The order of evaluation is **explicit** in small-step semantics but **implicit** in big-step semantics.
- Big-step semantics is more abstract, but less precise.
- Small-step semantics allows to make difference between non-termination and ”getting stuck”.

A functional langage

$$
\begin{array}{lllll}
t, u ::= & x & (variable) & | \\
& \mathtt{c} & (constant) & | \\
& \langle t, u \rangle & (pair) & | \\
& t\, u & (application) & | \\
& \lambda x.t & (abstraction) & | \\
& \mathtt{let}\ x = t\ \mathtt{in}\ u & (let) &
\end{array}
$$

Some constant function symbols : `fst`, `snd`, `ifthenelse`, $+$, $*$ ...
Some constants : `true`, `false`, 0, 1, 2, 3, ...
Thus e.g. `if` t `then` u `else` v can be defined as `ifthenelse`$\langle t, \langle u, v \rangle \rangle$.

A **program** is a closed expression belonging to the previous grammar.

Call-by-value lambda-calculus (big-step semantics)

(**CBV Normal Forms**) $V, W ::= \mathtt{c} \mid \langle V, V \rangle \mid \lambda x.t$

Meaningless expressions such as $(\langle 1, 1 \rangle\ 3)$ or $(\mathtt{true}\ 3)$ are not considered as CBV Normal Forms.

$$\frac{V \text{ is a CBV Normal Form}}{V \Downarrow_v V} \qquad \frac{t_1 \Downarrow_v V_1 \quad t_2 \Downarrow_v V_2}{\langle t_1, t_2 \rangle \Downarrow_v \langle V_1, V_2 \rangle} \qquad \frac{u \Downarrow_v V \quad r\{x/V\} \Downarrow_v W}{\mathtt{let}\ x = u\ \mathtt{in}\ r \Downarrow_v W}$$

$$\frac{t \Downarrow_v \lambda x.r \quad u \Downarrow_v W \quad r\{x/W\} \Downarrow_v V}{t\,u \Downarrow_v V}$$

$$\frac{t \Downarrow_v \mathtt{fst} \quad u \Downarrow_v \langle V_1, V_2 \rangle}{t\,u \Downarrow_v V_1} \qquad \frac{t \Downarrow_v \mathtt{snd} \quad u \Downarrow_v \langle V_1, V_2 \rangle}{t\,u \Downarrow_v V_2}$$

$$\frac{t \Downarrow_v \mathtt{ifthenelse} \quad u \Downarrow_v \langle \mathtt{true}, \langle V_1, V_2 \rangle \rangle}{tu \Downarrow_v V_1} \qquad \frac{t \Downarrow_v \mathtt{ifthenelse} \quad u \Downarrow_v \langle \mathtt{false}, \langle V_1, V_2 \rangle \rangle}{tu \Downarrow_v V_2}$$

Particular case: closed pure lambda-terms

(**CBV Normal Forms**) $V ::= \lambda x.t$

$$\frac{}{V \Downarrow_v V} \qquad \frac{t \Downarrow_v \lambda x.r \qquad u \Downarrow_v W \qquad r\{x/W\} \Downarrow_v V}{t\,u \Downarrow_v V}$$

An example

$t_0 = \lambda f.\lambda x.\langle x, f\ x\rangle$ and $t_1 = \lambda y.y$.

$$\frac{t_0\ t_1 \Downarrow_v \lambda x.\langle x, t_1\ x\rangle \qquad 1 \Downarrow_v 1 \qquad \langle 1, t_1\ 1\rangle \Downarrow_v \langle 1, 1\rangle}{t_0\ t_1\ 1 \Downarrow_v \langle 1, 1\rangle}$$

$$\frac{t_0\ \Downarrow_v\ \lambda f.\lambda x.\langle x, f\ x\rangle \qquad t_1\ \Downarrow_v\ \lambda y.y \qquad \lambda x.\langle x, f\ x\rangle\{f/t_1\} \Downarrow_v \lambda x.\langle x, t_1\ x\rangle}{t_0\ t_1 \Downarrow_v \lambda x.\langle x, t_1\ x\rangle}$$

$$\frac{1 \Downarrow_v 1 \qquad \dfrac{t_1\ \Downarrow_v \lambda y.y \qquad 1 \Downarrow_v 1 \qquad y\{y/1\} \Downarrow_v 1}{t_1\ 1 \Downarrow_v 1}}{\langle 1, t_1\ 1\rangle \Downarrow_v \langle 1, 1\rangle}$$

Call-by-value lambda calculus (small-step semantics)

$$\frac{t \rightsquigarrow_v t'}{\langle t, u\rangle \rightsquigarrow_v \langle t', u\rangle} \qquad \frac{u \rightsquigarrow_v u'}{\langle V, u\rangle \rightsquigarrow_v \langle V, u'\rangle}$$

$$\frac{u \rightsquigarrow_v u'}{\texttt{let}\ x = u\ \texttt{in}\ r \rightsquigarrow_v \texttt{let}\ x = u'\ \texttt{in}\ r} \qquad \frac{}{\texttt{let}\ x = V\ \texttt{in}\ r \rightsquigarrow_v r\{x/V\}}$$

$$\frac{t \rightsquigarrow_v t'}{t\,u \rightsquigarrow_v t'\,u} \qquad \frac{u \rightsquigarrow_v u'}{V\,u \rightsquigarrow_v V\,u'}$$

$$\frac{}{(\lambda x.t)\,V \rightsquigarrow_v t\{x \backslash V\}}$$

$$\frac{}{\texttt{fst}\ \langle V_1, V_2\rangle \rightsquigarrow_v V_1} \qquad \frac{}{\texttt{snd}\ \langle V_1, V_2\rangle \rightsquigarrow_v V_2}$$

$$\frac{}{\texttt{ifthenelse}\langle \texttt{true}, \langle V_1, V_2\rangle\rangle \rightsquigarrow_v V_1} \qquad \frac{}{\texttt{ifthenelse}\langle \texttt{false}, \langle V_1, V_2\rangle\rangle \rightsquigarrow_v V_2}$$

The same example

$t_0 = \lambda f.\lambda x.\langle x, f\ x\rangle$ and $t_1 = \lambda y.y$.

$$
\begin{array}{ll}
t_0\ t_1\ 1 & \leadsto_v \\
(\lambda x.\langle x, t_1\ x\rangle)\ 1 & \leadsto_v \\
\langle 1, t_1\ 1\rangle & \leadsto_v \\
\langle 1, 1\rangle &
\end{array}
$$

Call-by-name lambda-calculus (big-step semantics)

(**CBN Normal Forms**) $P ::= \mathtt{c} \mid \langle t, u \rangle \mid \lambda x.t$

$$\frac{P \text{ is a CBN normal-form}}{P \Downarrow_n P} \qquad \frac{r\{x/u\} \Downarrow_n P}{\mathtt{let}\ x = u\ \mathtt{in}\ r \Downarrow_n P}$$

$$\frac{t \Downarrow_n \lambda x.r \qquad r\{x/u\} \Downarrow_n P}{t\,u \Downarrow_n P}$$

$$\frac{t \Downarrow_n \mathtt{fst} \qquad u \Downarrow_n \langle t_1, t_2 \rangle \qquad t_1 \Downarrow_n P}{tu \Downarrow_n P} \qquad \frac{t \Downarrow_n \mathtt{snd} \qquad u \Downarrow_n \langle t_1, t_2 \rangle \qquad t_2 \Downarrow_n P}{tu \Downarrow_n P}$$

$$\frac{t \Downarrow_n \mathtt{ifthenelse} \qquad u \Downarrow_n \langle u_1, u_2 \rangle \qquad u_1 \Downarrow_n \mathtt{true} \qquad u_2 \Downarrow_n \langle m_1, m_2 \rangle \qquad m_1 \Downarrow_n P}{tu \Downarrow_n P}$$

$$\frac{t \Downarrow_n \mathtt{ifthenelse} \qquad u \Downarrow_n \langle u_1, u_2 \rangle \qquad u_1 \Downarrow_n \mathtt{false} \qquad u_2 \Downarrow_n \langle m_1, m_2 \rangle \qquad m_2 \Downarrow_n P}{tu \Downarrow_n P}$$

Particular case: closed pure lambda-terms

(**CBN Normal Forms**) $P ::= \lambda x.t$

$$\frac{}{P \Downarrow_n P} \qquad \frac{t \Downarrow_n \lambda x.r \qquad r\{x/u\} \Downarrow_n P}{t\,u \Downarrow_n P}$$

Call-by-name lambda calculus (small-step semantics)

$$\frac{}{\texttt{let}\ x = t\ \texttt{in}\ r \rightsquigarrow_n r\{x/t\}}$$

$$\frac{t \rightsquigarrow_n t'}{t\,u \rightsquigarrow_n t'\,u} \qquad \frac{}{(\lambda x.t)\,u \rightsquigarrow_n t\{x \backslash u\}}$$

$$\frac{t \rightsquigarrow_n t'}{\texttt{fst}\ t \rightsquigarrow_n \texttt{fst}\ t'} \qquad \frac{}{\texttt{fst}\ \langle t, u\rangle \rightsquigarrow_n t}$$

$$\frac{t \rightsquigarrow_n t'}{\texttt{snd}\ t \rightsquigarrow_n \texttt{snd}\ t'} \qquad \frac{}{\texttt{snd}\ \langle t, u\rangle \rightsquigarrow_n u}$$

$$\frac{t \rightsquigarrow_n t'}{\texttt{ifthenelse}\ t \rightsquigarrow_n \texttt{ifthenelse}\ t'} \qquad \frac{t \rightsquigarrow_n t'}{\texttt{ifthenelse}\ \langle t, u\rangle \rightsquigarrow_n \texttt{ifthenelse}\ \langle t', u\rangle}$$

$$\frac{t \in \{\texttt{true}, \texttt{false}\} \qquad u \rightsquigarrow_n u'}{\texttt{ifthenelse}\ \langle t, u\rangle \rightsquigarrow_n \texttt{ifthenelse}\ \langle t, u'\rangle}$$

$$\frac{t \rightsquigarrow_n t'}{\texttt{ifthenelse}\ \langle \texttt{true}, \langle t, u\rangle\rangle \rightsquigarrow_n \texttt{ifthenelse}\ \langle \texttt{true}, \langle t', u\rangle\rangle}$$

$$\frac{u \rightsquigarrow_n u'}{\texttt{ifthenelse}\ \langle \texttt{false}, \langle t, u\rangle\rangle \rightsquigarrow_n \texttt{ifthenelse}\ \langle \texttt{false}, \langle t, u'\rangle\rangle}$$

Coherence of results

- If $t \Downarrow_v u$, then u is a CBV normal-form.
- If $t \Downarrow_n u$, then u is a CBN normal-form.

Deterministic properties

- If $t \Downarrow_v V$ and $t \Downarrow_v V'$, then $V = V'$.
- If $t \Downarrow_n P$ and $t \Downarrow_n P'$, then $P = P'$.
- If $t \rightsquigarrow_v u$ and $t \rightsquigarrow_v u'$, then $u = u'$.
- If $t \rightsquigarrow_n u$ and $t \rightsquigarrow_n u'$, then $u = u'$.

Relating big and small-steps semantics

- If $t \Downarrow_v V$, then $t \rightsquigarrow_v^* V$.
- If $t \Downarrow_n P$, then $t \rightsquigarrow_n^* P$.
- If $t \rightsquigarrow_v^* u$ and u is a CBV normal-form, then $t \Downarrow_v u$.
- If $t \rightsquigarrow_n^* u$ and u is a CBN normal-form, then $t \Downarrow_n u$.

Call-by-Need

Lazy Evaluation and Call-by-Need

Lazy evaluation **(Wadsworth'71)**

- Based on demand-driven computation
- Implements memoization (the first demand-driven function call transforms the argument into a value)
- May manipulate potentially infinite data
- The best of call-by-name and the best of call-by-value
- Modeled by call-by-need calculi (Ariola-Felleisen).

Call-by-need different from call-by-name

Call-by-need is different from call-by-name: first evaluates arguments (like call-by-value).

$$\texttt{Twice}\ (4+3) \to_{\texttt{cbname}}\ (4+3)+(4+3) \to_{\texttt{cbname}}\ 7+(4+3) \to_{\texttt{cbname}}\ 7+7 \to_{\texttt{cbname}}\ 14$$
$$\texttt{Twice}\ (4+3) \to_{\texttt{cbneed}}\ \texttt{Twice}\ 7 \to_{\texttt{cbneed}}\ 7+7 \to_{\texttt{cbneed}}\ 14$$

where $\texttt{Twice} = \lambda x.x + x$.

Call-by-need different from call-by-value

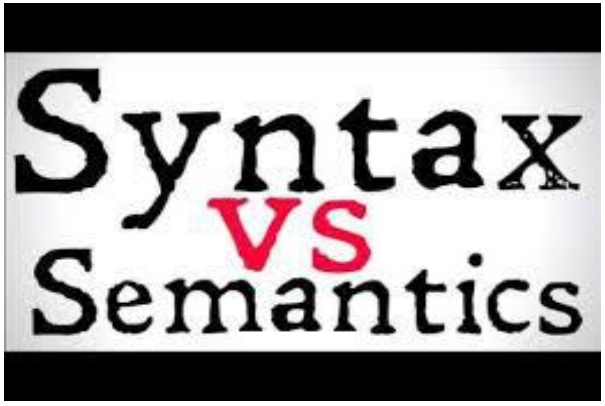
Call-by-need is different from call-by-value: values are only consumed when required

$$(\lambda x.8)(4+3) \rightarrow_{\texttt{cbvalue}} (\lambda x.8)7 \rightarrow_{\texttt{cbvalue}} 8$$
$$(\lambda x.8)(4+3) \rightarrow_{\texttt{cbneed}} 8$$

In particular

$$(\lambda x.8)\Omega \not\rightarrow_{\texttt{cbvalue}}$$
$$(\lambda x.8)\Omega \rightarrow_{\texttt{cbneed}} 8$$

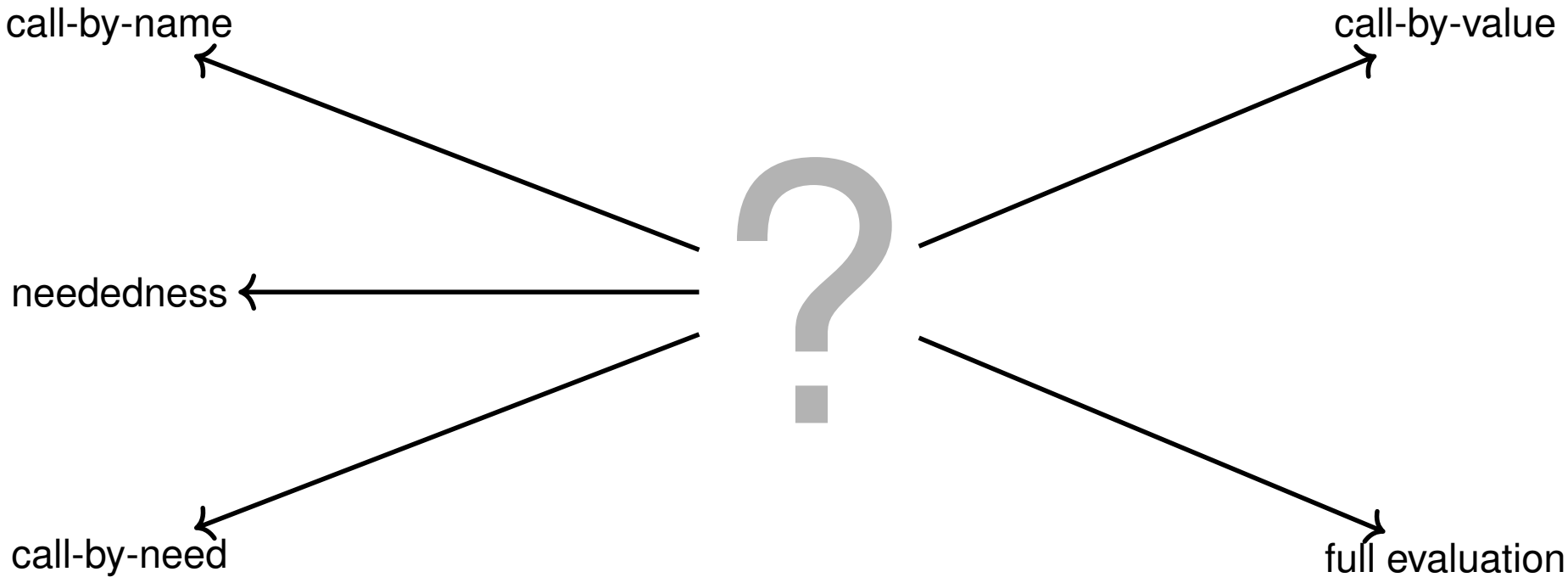
(Syntactical) call-by-need different from (semantical) neededness



Syntax: Call-by-need Evaluation strategy defined syntactically, using a notion of need context and let-constructors

Semantics: Neededness Evaluation strategy defined semantically, using the residual theory of λ-calculus.

But....


call-by-name
call-by-value
?
neededness
call-by-need
full evaluation


OBSERVATIONS

Observational equivalence, formally

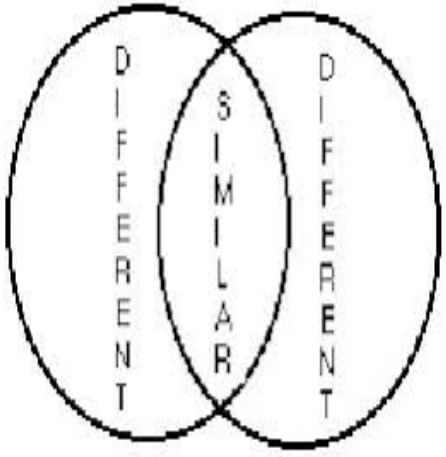
Let $\mathcal{R}$ be a reduction relation with an associated notion of result (e.g. value). We write $t \Downarrow_{\mathcal{R}}$ if the term t converges to some result.

Two terms t and u are said to be observationally equivalent for $\mathcal{R}$, written $t \cong_{\mathcal{R}} u$, if for **every context** C,

$$C[t] \Downarrow_{\mathcal{R}} \text{ if and only if } C[u] \Downarrow_{\mathcal{R}}.$$

- Terms that cannot be distinguished by any context.
- Often used to compare two different implementations or protocols.
- Difficult to reason because of the universal quantification of contexts.

Call-by-need observationally equivalent to call-by-name



Theorem

Given a program t, the call-by-name interpreter on t stops in a value if and only if the call-by-need interpreter on t stops in an answer.

Said differently,

Theorem

Given t and u we have that $t \cong_{cbname} u$ **if and only if** *$t \cong_{cbneed} u$.*

Call-by-Need

- Syntax uses let constructors/explicit substitutions

$$\texttt{let}\ x = u\ \texttt{in}\ t \text{ is written } t[x\backslash u]$$

Grammar of terms:

$t, u ::=$	x	(variable)	|
	$t\ u$	(application)	|
	$\lambda x.t$	(abstraction)	
	$t[x\backslash u]$	(explicit substitutions)	

- Reduction computes weak head normal forms with let constructors, so that reduction does not take place inside abstractions:
  $t \to t'$ does not imply $\lambda x.t \to \lambda x.t'$.
- Memoization $\Rightarrow$ notion of value (terms of the form $V ::= \lambda x.t$).
- Demand-driven computations $\Rightarrow$ notion of need context.

Explicit binding of arguments

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

$$\rightarrow$$

$$\texttt{let}\ x\ =\ (\lambda y.y)\ \texttt{in}\ id$$

(also written $id[\, x \setminus (\lambda y.y)\,]$)

First Reduction Rule:

$$(\lambda x.t)\,u \mapsto_{\mathtt{dB}} t[\, x \setminus u\,]$$

Explicit binding of arguments again

$$(\lambda x.\, id)\ [y\backslash\Omega]\ (\lambda y.y)$$

$$\rightarrow$$

$$id[\ x\ \backslash\ (\lambda y.y)\ ]\ [y\backslash\Omega]$$

First (Revised) Reduction Rule:

$$(\lambda\ x\ .t)\ [x_1\backslash u_1]\ldots[x_n\backslash u_n]\ \ u\ \mapsto_{\mathtt{dB}}\ t[\ x\ \backslash\ u\ ]\ [x_1\backslash u_1]\ldots[x_n\backslash u_n]$$

Substituting values when needed

$$x[x\backslash\lambda y.y]$$
$$\rightarrow$$
$$(\lambda y.y)[x\backslash\lambda y.y]$$

Second Reduction Rule:

$$\mathtt{N}[\![x]\!][x/V] \mapsto_{\mathtt{lsv}} \mathtt{N}[\![V]\!][x/V]$$

In the example $\mathtt{N} = \Box$, but what is $\mathtt{N}$ in general??

Substituting values when needed

Left of an application	Left of a let	Inside a needed let
$(x\ id)[x \backslash (\lambda y.y)]$	$x\,[y \backslash id][x \backslash (\lambda y.y)]$	$z\,[z \backslash x][x \backslash \lambda y.y]$
$(\mathtt{N}'[\![x]\!]t)[x \backslash \mathtt{V}]$	$\mathtt{N}'[\![x]\!][y \backslash t][x \backslash \mathtt{V}]$	$\mathtt{N}'[\![z]\!][z \backslash \mathtt{M}[\![x]\!]][x \backslash \mathtt{V}]$

Second Reduction Rule:

$$\mathtt{N}[\![x]\!][x/V] \mapsto_{\mathtt{lsv}} \mathtt{N}[\![V]\!][x/V]$$

where (need contexts) $\mathtt{M}, \mathtt{N} ::= \Box \mid \mathtt{N}t \mid \mathtt{N}[y \backslash t] \mid \mathtt{N}[\![z]\!][z \backslash \mathtt{M}]$

Sharing while Substituting

$$x\,[\,x\backslash(\lambda y.y)\;[z\backslash id]\,]$$
$$\rightarrow$$
$$x\,[x\backslash\lambda y.y]\;[z\backslash id]$$
$$\rightarrow$$
$$(\lambda y.y)[x\backslash(\lambda y.y)]\;[z\backslash id]$$

Second (Revised) Reduction Rule:

$$\mathtt{N}[\![\,x\,]\!][\,x\,/V\;[x_1\backslash u_1]\ldots[x_n\backslash u_n]\,] \mapsto_{\mathtt{lsv}}$$
$$\mathtt{N}[\![V]\!][\,x\,/V]\;[x_1\backslash u_1]\ldots[x_n\backslash u_n]$$

A Call-by-Need Calculus (Small-Step Semantics)

- Reduction Rules:

(Distant Beta reduction)

$$(\lambda x.t)\ [x_1\backslash u_1]\dots[x_n\backslash u_n]\ u \mapsto_{\mathtt{dB}} t[x \backslash u]\ [x_1\backslash u_1]\dots[x_n\backslash u_n]$$

(Linear substitution of values)

$$\mathtt{N}[\![x]\!][x/V\ [x_1\backslash u_1]\dots[x_n\backslash u_n]\ ] \mapsto_{\mathtt{lsv}} \mathtt{N}[\![V]\!][x/V]\ [x_1\backslash u_1]\dots[x_n\backslash u_n]$$

- Closed by need contexts $\mathtt{N}$

A full example:

$$\begin{array}{ll}
(\lambda x.id\ (x\ id))\ (\lambda y.\lambda z.id\ y) & \to_{\mathtt{dB}} \\
id(x\ id)[x/\lambda y.\lambda z.id\ y] & \to_{\mathtt{dB}} \\
w[w/x\ id][x/\lambda y.\lambda z.id\ y] & \to_{\mathtt{lsv}} \\
w[w/(\lambda y.\lambda z.id\ y)id][x/\lambda y.\lambda z.id\ y] & \to_{\mathtt{dB}} \\
w[w/(\lambda z.id\ y)[y/id]][x/\lambda y.\lambda z.id\ y] & \to_{\mathtt{lsv}} \\
(\lambda z.id\ y)[w/\lambda z.id\ y]\ [y/id]\ [x/\lambda y.\lambda z.id\ y] &
\end{array}$$

Different but equivalent weak head normal forms

Related weak head normal forms for $t = (\lambda x.id\ (x\ id))(\lambda y.\lambda z.id\ y)$

$$t \quad \rightarrow^*_{\texttt{cbname}} \quad \underset{value}{\lambda z.id\ id}$$

$$t \quad \rightarrow^*_{\texttt{cbneed}} \quad \underset{answer}{(\lambda z.id\ y)\,[w/\lambda z.id\ y]\,[y/id]\,[x/\lambda y.\lambda z.id\ y]}$$