Lambda Calculus

Lambda calculus serves as
a formal technique for defining the semantics of functional programming:

1. Define referentially transparent naming mechanism by formally specify parameter passing mechanisms and scoping rules.
2. Show how objects from basic data types can be defined in terms of functions
3. Show how recursion can be implemented so that it does not violate referential transparency
4. Serve as basis for interpreters for functional languages.

Syntax

- Symbols used:
  lower case letters
  $\lambda \quad | \quad (\quad)$

  (Each letter represents a function)

- Two kinds of expressions:
  function application
  function definition

BNF:

$\langle expression \rangle \ := \ \langle identifier \rangle \mid \langle application \rangle \mid \langle function \rangle$

$\langle application \rangle \ := \ (\langle expression \rangle \ \langle expression \rangle)$

$\langle function \rangle := (\lambda \langle identifier \rangle$ "|" $\langle expression \rangle)$

$\langle identifier \rangle \ := \ a \mid b \mid c \ \ldots$

Eg: `f` is an identifier (function)
`(f g)` is application of function `f` to the argument function `g`
$(\lambda\ x \mid (x\ y))$ is a function definition with parameter `x` and body `(x y)`
$(\lambda y \mid \underline{(\lambda x \mid (y\ (y\ x)))}\ )$ is fn defn with parameter `y` and body $(\lambda x \mid (y\ (y\ x)))$

A Model of Computation

- How to "compute" with lambda functions?

Eg1: $(\ (\lambda x \mid (+\ x\ 4))\ 9) \ \mapsto\ (+\ 9\ 4)$

...substitute `9` for `x` in function body: `(+ x 4)`

(Of course, don't have "4", "9" or "+" – see below)

Eg2: $(\ (\lambda f \mid \underline{(f\ x)})\ \overline{s}) \ \mapsto\ (s\ x)$

subst `s` for `f` in `(f x)`

Eg3: $(\ (\lambda f \mid \underline{(f\ x)})\ \overline{(\lambda y \mid y)}\ )$
$\mapsto\ (\ (\lambda y \mid \underline{y})\ \overline{x})$
$\mapsto\ x$

Eg4: $(\ \underline{(\lambda f \mid (f\ (f\ x)))}\ (\lambda y \mid (g\ (g\ (g\ y))))\ )\ )$

$\mapsto (\ (\lambda y \mid (g\ (g\ (g\ y))))\ \underline{((\lambda y \mid (g\ (g\ (g\ y))))\ x)})$

$\mapsto (\ (\lambda y \mid \underline{(g\ (g\ (g\ y)))})\ \overline{(g\ (g\ (g\ x)))}\ )$

$\mapsto (g\ (g\ (g\ (g\ (g\ (g\ x))))))$

Computing in Lambda Calculus

- Computation in λ-Calculus $\equiv$
  reducing expression to simpler form
- Simplification results only from
  rewriting function applications that contain function definitions.
- Can NOT reduce: `f`, `(f g)`, $(\lambda f \mid (f\ g))$
- Can reduce $(\ \underline{(\lambda x \mid (w\ x))}\ y)$ as it is a function application whose left function has a function definition

Example

- Want to apply some function TWICE:

  `twice(f x)` $\equiv$ `f( f(x) )`

Eg: `f = car`, means twice(car x) = caar(x)
`f = square`, means twice(square x) = to-the-fourth(x)
. . .

- Let function I be "increment by 1"
  so (I 2) is 3

$$(\underbrace{((\lambda f \mid (\lambda x \mid (f\ (f\ x)))))}_{\textit{function}}\ \underbrace{I}_{\textit{arg}})\ 2)$$

$\downarrow$ subst `f` by `I`. . .

$$(\underbrace{(\lambda x \mid (I\ (I\ x)))}_{\textit{function}}\ \underbrace{2}_{\textit{arg}}\ )$$

$\downarrow$ subst `x` by `2`. . .

`(I (I 2))`

$\downarrow$ apply `I` to `2`. . .

`(I 3)`

$\downarrow$ apply `I` to `3`. . .

`4`

- Note: "nested function"

  "value" of $\underbrace{((\lambda f \mid (\lambda x \mid (f\ (f\ x))))}_{\textit{function}}\ \underbrace{I}_{\textit{arg}})$

  is function $(\lambda x \mid (I\ (I\ x)))$

Rewriting Rule

- Given lambda expression $((\lambda x \mid E)\ M)$,
  substitute `M` for parameter `x`
  in body `E` of function definition:

$$((\lambda x \mid E)\ M) \quad \mapsto \quad [M\ /\ x]\ E$$

Eg: $(\ \underline{(\lambda x \mid (w\ x))}\ y) \mapsto [y\ /\ x]\ (w\ x) \quad \mapsto \quad (w\ y)$

- In complicated lambda expression:
  keep applying simplifying reductions
  until no more can be made.

Removing ANNOYING Parentheses

- Write `(...((E1 E2) E3) ... En)`
  as `(E1 E2 E3 ... En)`

  w/ understanding that `E1` MUST be applied to `E2`
  before *that* result is applied to `E3`, etc.

Eg: $(\ (\ (\lambda f \mid (f\ f))\ (\lambda y \mid y))\ (\lambda z \mid w))$
$= ((\lambda f \mid (f\ f))\ (\lambda y \mid y)\ (\lambda z \mid w))$

$((\lambda f \mid \underline{(f\ f)})\ \overline{(\lambda y \mid y)}\ (\lambda z \mid w))$
$\mapsto (((\lambda y \mid \underline{y})\ \overline{(\lambda y \mid y)})\ (\lambda z \mid w))$
$\mapsto ((\lambda y \mid \underline{y})\ \overline{(\lambda z \mid w)}) \quad \mapsto \quad (\lambda z \mid w)$

Left to right order is important! If ignored:

$((\lambda f \mid (f\ f))\ (\lambda y \mid \underline{y})\ \overline{(\lambda z \mid w)})$
$\mapsto ((\lambda f \mid \underline{(f\ f)})\ \overline{(\lambda z \mid w)})$
$\mapsto ((\lambda z \mid \underline{w})\ \overline{(\lambda z \mid w)}) \quad \mapsto w$

which is WRONG!

Removing Parentheses (II)

- Drop outer set of parentheses:

Eg: ((λf | (f f)) (λy | y) (λz | w)) =
(λf | (f f)) (λy | y) (λz | w)

- Drop set of parentheses around body of function:

Eg: (λf | (f f)) (λy | y) (λz | w) =
(λf | f f) (λy | y) (λz | w)

(λf | f f) (λy | y) (λz | w)

↦ (λy | y) (λy | y) (λz | w)

↦ (λy | y) (λz | w) ↦ (λz | w)

- Drop parentheses around nested function definitions, and drop extra lambdas:

Eg: ((λx | (λz | (x z))) y)
= (λx | (λz | x z)) y
= (λx z | x z) y

↦ [y / x] (λz | x z) ↦ (λz | y z)

twice Example

(((λf | (λx | (f (f x)))) I) 2)

= (λf x | f (f x)) I 2

≠ (λf x | f f x)) I 2
as "f f x" means "((f f) x)"

≠ (λx f | f (f x))) I 2
as "(λx f | E)" means "(λx (λf | E))"

Problem with Free Variables

- Direct substitution is not always possible.

Eg: (λxy | x y) y ↦ [y / x] (λy | x y) ↦ (λy | y y)

Incorrect, as original expression could have been:

(λxz | x z) y ↦ [y / x] (λz | x z) ↦ (λz | y z)

- Need substitution rule that *changes* names of parameters if they clash with other names.

- Still use:
(λx | E) M ↦ [M / x] E
but notation [M / x] E needs better definition than simple "macro" replacement of x by M.

Free and Bound Variables

Def'n: Names that appear in variable name list (and expression body) of a lambda expression are *bound variables* in lambda expression.

Def'n: Names that appear in expression body of lambda expression but do not appear in the variable list are called *free variables* in lambda expression.

Eg: In (λx | x y), "x" is bound, "y" is free.

Definition of Bound/Free Vars

E1: Variable x occurs free in expression x.
No variable is bound in an expression consisting of a single variable.

E2: Variable x occurs free (bound) in (YZ) *iff* it occurs free (bound) in Y or Z.

E3: All occurrences of V are bound in (λV | Y)
If variable x $\neq$ V, then
x occurs free (or bound) in (λV | Y) *iff* it occurs free (or bound) in Y.

E4: Variable x occurs free (bound) in (Y) iff it occurs free (or bound) in Y.

Note: Variable can be both free and bound in an expression

Eg: y is bound and free in the lambda expression
$((\lambda \underbrace{y}_{1} \mid \underbrace{y}_{2}) \underbrace{y}_{3})$

Instance 1, 2 are bound (*E2, E3*);
while Instance 3 is free (*E2, E1*)

Substitution Rules

$\boxed{\texttt{[M / x] E} \mapsto \texttt{E'}}$ where

1. E' = M when E is the variable x
   (λx |$\underline{x}$) m $\mapsto$ [m/x] x $\mapsto^1$ m

2. E' = E when E is a variable $\neq$ x
   (λx |$\underline{y}$) m $\mapsto$ [m/x] y $\mapsto^2$ y

3. E' = ([M / x] Y) ([M / x] Z) when E = (Y Z)
   (λx |$\underline{(x\ y)x}$) m $\mapsto$ [m/x] ((xy)x)
   $\mapsto^3$ ([m/x](xy) [m/x]x) $\mapsto$ ((my)m)

   E' = ([M / x] Y) when E = (Y)

4. E' = E when E is the function (λx | Y)
   (λx |$\underline{(\lambda x \mid x\ N)}$) m $\mapsto$ [m/x] (λx | x N)
   $\mapsto^4$ (λx | x N)

   (Note don't subst x*!)*

5. E' = (λy | [M / x] Y) when E = (λy | Y)
   and $\nexists$ free occurrences of y in M
   (λx |$\underline{(\lambda y \mid x\ y)}$) m $\mapsto$ [m/x] (λy | x y)
   $\mapsto^5$ (λy | [m/x] (x y)) $\mapsto$ (λy | (m y))

6. E' = (λz | [M / x] ([z / y] Y)) when E = (λy | Y) and $\exists$ free occurrence of y in M
   (λx |$\underline{(\lambda y \mid x\ y)}$) (a y) $\mapsto$ [(a $\underbrace{y}_{free}$) /x] (λy | x $\underbrace{y}_{bound}$)
   $\mapsto^6$ (λz | [(a y)/x] [z/y] (x y))
   $\mapsto$ (λz | [(a y)/x] (x z)) $\mapsto$ (λz | ((a y) z)
)

Eg: (λxy | x y) y $\mapsto$ [y / x] (λy | x y)
$\mapsto^6$ λz | [y / x] ([z / y] x y)
$\mapsto^3$ (λz | [y / x] (([z / y] x) ([z / y] y)))
$\mapsto^2$ (λz | [y / x] ((x) ([z / y] y)))
$\mapsto^1$ (λz | [y / x] (x z))
$\mapsto^3$ (λz | ([y / x] x) ([y / x] z))
$\mapsto^1$ (λz | (y) ([y / x] z)) $\mapsto^2$ (λz | y z)

Another Example

(λx | (λy | $\underline{(\lambda x \mid x\ (xy))}$ $\underline{(\lambda z \mid +\ x\ z)}$) (+ x 4)) 2
do inner subst first: (+ x 4) *for* y
(λx | [(+ x 4)/ y] $\underline{(\lambda x \mid x\ (xy))}$ $\underline{(\lambda z \mid +\ x\ z)}$) 2
rename bound x
(λx | [(+ x 4)/ y] $\underline{(\lambda w \mid w\ (wy))}$ (λz | + x z)) 2
(λx | $\underline{(\lambda w \mid w\ (w(+\ x\ 4)))}$ (λz | + x z)) 2
[2/x] ((λw | w (w(+ $\underline{x}$ 4))) (λz | + $\underline{x}$ z))
((λw | w (w(+ 2 4))) (λz | + 2 z))
((λw | w (w 6)) (λz | + 2 z))
[(λz | + 2 z)/w] (w (w 6))
(λz | + 2 z) ((λz | + 2 z) 6)
(λz | + 2 z) [6/z] (+ 2 z)
(λz | + 2 z) (+ 2 6)
(λz | + 2 z) 8
[8/z] (+ 2 z)
(+ 2 8)
10

Note: Sometimes, several possible substitutions
- [(+ x 4)/y]
- [2/x]

Which one should we do first?
Does this order matter?

. . . later . . .

Conversion Rules for simplifying λ-expr's

α: "variable renaming"

$\boxed{\text{if } \nexists \text{ free occurrences of } y \text{ in } X \text{ then } \quad \lambda x \mid X \;\mapsto^{\alpha}\; \lambda y \mid [y / x]\ X}$

. . . can change name of formal parameter
(but not to name of local / global variable)

Do this before substitution to avoid subst rule #6

Eg: $(\lambda x \mid x\ z\ x) \mapsto_{\alpha} (\lambda y \mid y\ z\ y)$
NOT: $(\lambda x \mid +\ x\ y) \not\mapsto_{\alpha} (\lambda y \mid +\ y\ y)$

β: "function application"

$\boxed{(\lambda x \mid M)\ N \;\mapsto^{\beta}\; [N / x]\ M}$

$\equiv$ standard substitution rule
just substitute arguments for parameters

Eg: $(\lambda x \mid x\ z)\ y \mapsto_{\beta} [y/x]\ (x\ z) \mapsto (y\ z)$

η: "optimization"

$\boxed{\text{if } \nexists \text{ free occurrences of } x \text{ in } M \quad (\lambda x \mid M\ x) \;\mapsto_{\eta}\; M}$

Removes "indirection" in function applications

Eg: $(\lambda x \mid (\lambda y\ z \mid z\ y)\ x) \mapsto_{\eta} (\lambda y\ z \mid z\ y)$

Comments on Conversion Rules

Eg: $(\lambda x \mid (\lambda y \mid x\ y\ ))\ y \mapsto [y / x]\ (\lambda y \mid x\ y\ )$
$\mapsto^{6} \lambda z \mid [y / x]\ (\ [z / y]\ x\ y)$
$\mapsto \lambda z \mid [y / x]\ (x\ z) \mapsto \lambda z \mid y\ z$

becomes

$(\lambda x \mid \underline{(\lambda y \mid x\ y\ )})\ y \mapsto_{\alpha} (\lambda x \mid \underline{(\lambda z \mid x\ z\ )})\ y$
$\mapsto [y / x]\ (\lambda z \mid x\ z\ ) \mapsto^{5} (\lambda z \mid y\ z)$

Eg: $\underline{(\lambda x \mid \lambda yz \mid z\ y)\ x)}\ p\ (\lambda x \mid x) \mapsto_{\eta}$
$\underline{(\lambda yz \mid z\ y)}\ p\ (\lambda x \mid x) \mapsto_{\eta}$
$(\lambda z \mid z\ p)\ (\lambda x \mid x) \mapsto_{\beta} (\lambda x \mid x)\ p \mapsto_{\beta} p$

- Can often use η-rule used in place of β-rule:
  $\underline{(\lambda x \mid M\ x)}\ N \mapsto_{\eta} M\ N$
  $\underline{(\lambda x \mid M\ x)}\ N \mapsto_{\beta} [N / x]\ (M\ x) = M\ N$
- The β- and η-rules used to reduce complexity of expression.

Comparison

- Each of
  6 substitution rules
  3 conversion rules
  is "legal"

  Can use any when "evaluating" an expression
- β corresponds to all substitutions
  . . . , to all 6 substitution rules
- Can chain them together

Reduction, Redex

- β and η conversions called "reductions"
  written "$M \mapsto_{a} N$" where "a" is β, η.
- "$M \mapsto N$" means
  "M is reducible to N
  by a series of (β, η) reductions,
  and possibly some α conversions.
- An expression that can be reduced is called a *redex*.

  "$(\lambda x \mid M)\ N$" is a β-redex
  "$\lambda x \mid M\ x$" is η-redex.
- An expression containing no redexes is in *normal form*

Note: α is NOT reduction

- Computation $\equiv$
  reducing expression to normal form

Order of Reductions

- Cannot always reduce expression to normal form:

Eg: $(\lambda x \mid x\ x)\ (\lambda x \mid x\ x) \ \mapsto\ (\lambda x \mid x\ x)\ (\lambda x \mid x\ x)$
$\mapsto\ (\lambda x \mid x\ x)\ (\lambda x \mid x\ x) \ \mapsto\ \ldots$

- An expression may contain > 1 redex;
  order of reduction is not fixed.

- When > 1 "reduction" can be made,
  Which one should be done first?
  Does it matter?
  (or always produce same answer?)

Eg: $((\lambda f \mid f\ x)\ ((\lambda z \mid \underline{z})\ \overline{y})) \ \mapsto\ ((\lambda f \mid \underline{f\ x})\ \overline{y}) \ \mapsto\ (y\ x)$

$((\lambda f \mid \underline{f\ x})\ \overline{((\lambda z \mid z)\ y)}) \ \mapsto\ ((\lambda z \mid \underline{z})\ \overline{y}\ x) \ \mapsto\ (y\ x)$

Normal vs Applicative Order

- Normal Order Reduction
  (≡ leftmost order)
  Take leftmost variable first

- Applicative Order Reduction
  (≡ innermost order)
  Substitute INNERMOST variable first

Eg: f(x) = x+x   g(x) = x+1
Normal Order:
f(g(2)) = g(2) + g(2) = 3 + g(2) = 3 + 3 = 6

Applicative Order:
f(g(2)) = f(3) = 3+3 = 6

Q: Always same answers?
What about other orderings?

Example#1

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

Normal:
$\mapsto\ (\lambda \underline{x} \mid ((\lambda y\ x \mid x)\ z))\ \overline{(\lambda x \mid x\ y)}$
$\mapsto\ [(\lambda x \mid x\ y)\ /\ x]\ ((\lambda y\ x \mid x)\ z)$
$\mapsto\ (\underline{[(\lambda x \mid x\ y)\ /\ x]\ (\lambda y\ x \mid x)}\ \underline{[(\lambda x \mid x\ y)\ /\ x]\ z}\ )$
$\mapsto\ (\lambda y \mid (\underline{[(\lambda x \mid x\ y)\ /\ x]\ (\lambda x \mid x)})\ z\ )$
$\mapsto\ (\lambda y\ x \mid x)\ z$
$\mapsto\ (\lambda x \mid x)$

Applicative:
$\mapsto\ (\lambda x \mid ((\lambda \underline{y}\ x \mid x)\ \overline{z}))\ (\lambda x \mid x\ y)$
$\mapsto\ (\lambda \underline{x} \mid (\lambda x \mid x))\ \overline{(\lambda x \mid x\ y)}$
$\mapsto\ [(\lambda x \mid x\ y)\ /\ x]\ (\lambda x \mid x)$
$\mapsto\ (\lambda x \mid x)$

Example #2

$(\lambda x \mid (\lambda y \mid x))\ ((\lambda x \mid y)\ x)$

Normal:
$(\lambda \underline{x} \mid (\lambda y \mid x))\ \overline{((\lambda x \mid y)\ x)}$
$\mapsto\ [((\lambda x \mid y)\ x)\ /\ x]\ (\lambda y \mid x)$
$\mapsto_\alpha\ [((\lambda x \mid y)\ x)\ /\ x]\ (\lambda z \mid x)$
$\mapsto\ (\lambda z \mid ((\lambda x \mid y)\ x)\ )$
$\mapsto\ (\lambda z \mid [x\ /\ x]\ y) \ \mapsto\ (\lambda z \mid y)$

Applicative:
$(\lambda x \mid (\lambda y \mid x))\ ((\lambda \underline{x} \mid y)\ \overline{x})$
$\mapsto\ (\lambda x \mid (\lambda y \mid x))\ y$
$\mapsto\ [y\ /\ x]\ (\lambda z \mid x) \ \mapsto\ (\lambda z \mid y)$

Example #3

$(\lambda x\ y \mid y)\ ((\lambda x \mid x\ x)\ (\lambda x \mid x\ x))\ a$

Normal:

$(\lambda \underline{x}\ y \mid y)\ \overline{((\lambda x \mid x\ x)\ (\lambda x \mid x\ x))}\ a$

$\mapsto [((\lambda x \mid x\ x)\ (\lambda x \mid x\ x))\ /\ x\ ]\ (\lambda y \mid y)\ a$

$\mapsto (\lambda y \mid [((\lambda x \mid x\ x)\ (\lambda x \mid x\ x))\ /\ x\ ]\ y)\ a$

$\mapsto a$

Applicative:

$(\lambda x\ y \mid y)\ ((\lambda \underline{x} \mid x\ x)\ \overline{(\lambda x \mid x\ x)})\ a$

$\mapsto (\lambda x\ y \mid y)\ ([(\lambda x \mid x\ x)\ /\ x]\ (x\ x))\ a$

$\mapsto (\lambda x\ y \mid y)\ ((\lambda x \mid x\ x)\ (\lambda x \mid x\ x))\ a$

$\mapsto \ldots$

Uniqueness Theorem

- *Church Rosser Theorem I (CR-I)*

  . . . guarantees uniqueness . . .

  If "A $\mapsto$ B" and "A $\mapsto$ C" then
  $\exists$ expression D s.t. B $\mapsto$ D and C $\mapsto$ D.

$\Rightarrow$ cannot get non-equivalent expressions by choosing different reductions.

- *If* B is normal form,
  then no more simplifications possible
  so D $\equiv$ B

  CR-I says: can get from C to B.

- How to find such a good reduction?

Existence Theorem

Def'n: If A and B are redexes in expression M and first λ in A is *left of* first λ in B, then "A *is left of* B"

Def'n: If A, a redex in M, is left of all other redexes, then "A *is leftmost redex of* M"

Def'n: Successive reduction of leftmost redex of expression called
"*normal order reduction*".

- *Church Rosser Theorem II – CR-II*:

  . . . guarantees existence . . .

  if A $\mapsto$ B and B is in normal form
  then A $\mapsto$ B by normal order reduction.

- *The 2 Church Rosser theorems provide existence and uniqueness of normal forms!*

Parameter Passing

- Common parameter passing mechanisms:
  #1 call by value
  #2 call by value-result (copy)
  #3 call by reference
  #4 call by name

- Exclude $\left\{ \begin{array}{c} \textit{call by value-result} \\ \textit{call by reference} \end{array} \right\}$
  as they work by side-effect.

- *Call by value* $\equiv$ applicative order reduction
  arguments to function are reduced *before* they are passed

- *Call by name* $\equiv$ normal order reduction

Comparison

- In *CbValue*,
  each parameter is reduced only once:

Eg: $(\lambda x \mid x + x)\ ((\lambda y \mid y + 1)\ 6)$
$\mapsto (\lambda x \mid x + x)\ (6 + 1)$
$\mapsto (\lambda x \mid x + x)\ 7 \quad \mapsto 7 + 7 \quad \mapsto 14$

- In *CbName*, parameter may be reduced more than once:

Eg: $(\lambda x \mid x + x)\ ((\lambda y \mid y + 1)\ 6)$
$\mapsto ((\lambda y \mid y + 1)\ 6) + ((\lambda y \mid y + 1)\ 6)$
$\mapsto (6 + 1) + ((\lambda y \mid y + 1)\ 6)$
$\mapsto 7 + ((\lambda y \mid y + 1)\ 6) \quad \mapsto 7 + (6 + 1)$
$\mapsto 7 + 7 \quad \mapsto 14$

$\Rightarrow$ *CbValue* appears more efficient!

- but. . .

CbName Succeeds More Often

- $\exists$ expressions that
  can be reduced using *CbName*, but
  cannot be reduced using *CbValue*!

Eg: Using *CbValue*:

$(\lambda z\ y \mid 3 + y)\ ((\lambda x \mid \underline{x\ x})\ \overline{(\lambda x \mid x\ x)})\ 6 \quad \mapsto$
$(\lambda z\ y \mid 3 + y)\ ((\lambda x \mid \underline{x\ x})\ (\lambda x \mid x\ x))\ 6 \quad \mapsto$
. . .

Reduction does not terminate as reduction of first argument is itself, which is still a redex. . .

Using *CbName*:

$(\lambda z\ \underline{y} \mid 3 + y)\ \overline{((\lambda x \mid x\ x)\ (\lambda x \mid x\ x))}\ 6$
$\mapsto (\lambda y \mid \underline{3 + y})\ \overline{6} \quad \mapsto 3 + 6 \quad \mapsto 9$

- By CR-II, the reverse situation
  (reducible by value but not by name)
  cannot occur.

Combine: Call by Need?

- Normal order (*CbName*) is
  sometimes stupid, inefficient

  But has guarantee:

  $\boxed{\text{If } \exists \text{ normal form, normal order will find it.}}$

- Applicative order (*CbValue*) is often more efficient . . . But no guarantee!

- Can combine *CbName* and *CbValue* into
  "call by need"

  parameter evaluated only when needed, and
  its value is remembered "cached" for future use.

$(\lambda x\ z \mid x + x)\ ((\lambda y \mid y + 1)\ 6)\ ((\lambda x \mid x\ x)\ (\lambda x \mid x\ x))$
$\mapsto (\lambda z \mid ((\lambda y \mid y + 1)\ 6) + ((\lambda y \mid y + 1)\ 6))$
$\quad ((\lambda x \mid x\ x)\ (\lambda x \mid x\ x))$
$\mapsto ((\lambda y \mid y + 1)\ 6) + ((\lambda y \mid y + 1)\ 6))$
$\mapsto (6 + 1) + ((\lambda y \mid y + 1)\ 6)$
$\mapsto 7 + ((\lambda y \mid y + 1)\ 6) \quad \mapsto 7 + 7 \quad \mapsto 14$

- $\exists$ effective way to compute "optimal" reduction?

Summary of Reduction Order

- Not every expression has a normal form
  $((\lambda x \mid x\ x)\ (\lambda x \mid x\ x))$

CR-II: *If* expr has normal form, then
Normal Order Reduction (*CbName*) will find it

$+$ Normal Form is unique (up to var names)

But Normal Order Reduction may be *inefficient*
ie, may apply same reduct. to same expr many times

- Applicative Order Reduction (*CbValue*)
  may be more efficient
  but not never find normal form, even if it $\exists$

CR-I: Cannot get "wedged"
No matter what initial sequence of reductions,
can always go on to find normal form, if it $\exists$
but may have to start using normal order. . .

- Size of expression may grow, before simplifying

$\neq$ replacing parentheses. . .

Multiple Arguments

- Syntax: $(\lambda x\ y\ z \mid E)\ A\ B\ C$
  means: $(\lambda x \mid (\lambda y\ (\lambda z \mid E)))\ A\ B\ C$

  Apply each arg, one at a time:

  $([A/x]\ (\lambda y\ (\lambda z \mid E)))\ B\ C$
  If A *has free* y, z, *must rename!*

  $([B/y]\ ([A/x]\ (\lambda z \mid E)))\ C$
  If B *has free* z, *must rename!*

  $([C/z]\ ([B/y]\ ([A/x]\ E)))$

Eg: $(\lambda x\ y \mid x\ y)\ (N\ y)\ m$

$\mapsto^6\ [(Ny)/x]\ [z/y]\ (\lambda y \mid x\ y)\ m$

$\mapsto [(Ny)/x]\ (\lambda z \mid x\ z)\ m$

$\mapsto (\lambda z \mid (Ny)\ z)\ m$

$\mapsto [m\ /\ z]\ (Ny)\ z$

$\mapsto (Ny)\ m$

Simultaneous Substitution

- Note: can substitute x, y AT SAME TIME:

  $(\lambda \underline{x}\ \overline{y} \mid (x\ y))\ \underline{(N\ y)}\ \overline{m} \mapsto (\underline{(N\ y)}\ \overline{m})$

  (Don't need renaming!)

- !True in general!

- Why?
  only potential problem is if...

  $([B/y]\ (\lambda z \mid \ldots z \ldots))\ C$
  If B *has free* z, *must rename!*

  Using $\mapsto^6$, to ...
  $([B/y]\ (\lambda w \mid \ldots w \ldots))\ C$

  But note this "w" is just dummy variable
  to be replaced by C anyway!

- ...check no other substitution rule causes problem...

- Done by Lisp!