

## λ-CALCULUS SYNTAX

$e ::= v$  variable reference  
|  $\lambda v. e$  lambda expression / abstraction  
|  $(e e)$  procedure calls

## λ-CALCULUS SEMANTICS

$$(\lambda x. E M) \rightarrow E\{M/x\}$$

where we choose a "fresh  $x$ ", by  $\alpha$ -renaming the lambda-expression if necessary.

A "Scheme-like" representation of  $\lambda v. e$

is:

(lambda (v) e)

## NORMAL VS APPLICATIVE ORDER

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

In normal order (call-by-name)

$(\lambda x. y \frac{(\lambda x. (x x) \lambda x. (x x))}{M})$

$\xrightarrow{A} y$

In applicative order (call-by-value)

$(\lambda x. y (\lambda x. \frac{(x x)}{E} \frac{\lambda x. (x x)}{M}))$

$\xrightarrow{B} (\lambda x. y (\lambda x. (x x) \lambda x. (x x)))$

## $\alpha$ -RENAMING

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

$$\xrightarrow{\alpha} (\lambda z. (y z) x)$$

Only bound variables can be renamed. No free variables can be captured (become bound) in the process. For example, we cannot  $\alpha$ -rename  $x$  to  $y$ .

## $\beta$ -REDUCTION

$$(\lambda x. E M) \xrightarrow{\beta} E\{M/x\}$$

May require  $\alpha$ -renaming to prevent capturing free variable occurrences.  
For example:

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

$$\xrightarrow{\alpha} (\lambda x. \lambda z. (x z) (y w))$$

$$\xrightarrow{\beta} \cancel{\lambda x. (\lambda z. (x z) (y w))} \lambda z. ((y w) z)$$

## $\eta$ -CONVERSION

$$\lambda x. (E x) \xrightarrow{\eta} E$$

if  $x$  is not free in  $E$ .

For example:

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

$$\xrightarrow{\alpha} (\lambda x. \lambda z. (x z)) (y w)$$

$$\xrightarrow{\beta} \lambda z. ((y w) z)$$

$$\xrightarrow{\eta} (y w)$$

## COMBINATORS

A  $\lambda$ -calculus expression with no free variables is called a combinator. For example:

Identity  $I = \lambda x. x$

Application  $App = \lambda f. \lambda x. (f x)$

Currying  $Cur = \lambda f. \lambda x. \lambda y. ((f x) y)$

Sequencing  $Seq = \lambda x. \lambda y. (\lambda z. y x)$   
(normal order)

$ASeq = \lambda x. \lambda y. (y x)$   
(applicative order)

where 'y' is a lambda abstraction 'wrapping' the second expression to evaluate.

Loop  $Loop = (\lambda x. (x x)) (\lambda x. (x x))$

## SEQUENCING COMBINATOR

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

(Normal Order)

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

(Applicative Order)

with the 2nd. argument as  
a thunk.

## RECURSION (Y) COMBINATOR

$$n! = \begin{cases} 1, & n=0 \\ n \times (n-1)!, & n>0 \end{cases}$$

$$f: \lambda g. \lambda n. (if \ (= \ n \ 0) \\ \ 1 \\ (\# \ n \ (g \ (- \ n \ 1))))$$

$$f: (\mathbb{Z} \rightarrow \mathbb{Z}) \rightarrow (\mathbb{Z} \rightarrow \mathbb{Z})$$

(f f) ✗ Not well-typed.

$$(f I) \quad f(n) = \begin{cases} 1, & n=0 \\ n \times (n-1), & n>0 \end{cases}$$

$$(f X) = X$$

$$X: (\lambda x. (\lambda g. \lambda n. (\text{if } (= n 0) \\ \frac{1}{(x\ n\ (g\ (-\ n\ 1))}) \\ \lambda y. ((x\ x)\ y)) \\ \lambda x. (\lambda g. \lambda n. (\text{if } (= n 0) \\ (x\ n\ (g\ (-\ n\ 1))) \\ ))$$

$X$  can be defined as  $(Y\ F)$   
 where  $Y$  is the recursion  
 combinator.

$$(F\ X) = \underline{(F\ (Y\ F))} = (Y\ F)$$

Exercise: Prove equation  
 holds, for a  
 given  $f$ .

## RECURSION COMBINATOR ( $Y$ )

$$Y = \text{Rec} = \lambda f. (\lambda x. (f\ (x\ x)) \\ \lambda x. (f\ (x\ x))) \\ \text{(normal order)}$$

$$Y = \text{Rec} = \lambda f. (\lambda x. (f\ \lambda y. ((x\ x)\ y)) \\ \lambda x. (f\ \lambda y. ((x\ x)\ y))) \\ \text{(applicative order)}$$

You get from the normal order to the  
 applicative order recursion combinator by  
 $\lambda$ -expansion ( $\lambda$ -conversion from right  
 to left).

## NATURAL NUMBERS IN $\lambda$ -CALCULUS

$$[0] = \lambda x. x$$

$$[1] = \lambda x. \lambda x. x$$

⋮

$$[n+1] = \lambda x. [n]$$

## BRANCHING (IF) IN $\lambda$ -CALCULUS (NORMAL ORDER)

true

$$\lambda x. \lambda y. x$$

~~$v \rightarrow (v \rightarrow v)$~~

false

$$\lambda x. \lambda y. y$$

~~$v \rightarrow (v \rightarrow v)$~~

if  $\lambda b. \lambda t. \lambda e. (b \ t) e$

~~$(v \rightarrow (v \rightarrow v))$~~

$$(v \rightarrow (v \rightarrow v)) \rightarrow E \rightarrow (E \rightarrow E)$$