CMPUT 325 : Lambda Calculus Basics

Dr. B. Price and Dr. R. Greiner

13th October 2004

Lambda Calculus

- Lambda calculus serves as a formal technique for defining the semantics of functional programming:
  - Defines referentially transparent naming mechanism by formally specifying parameter passing mechanisms and scoping rules
  - Represents basic data types in terms of functions
  - Implements recursion witihout violating referential transparency
  - Provides a model for interpreters for functional languages

Computation as Rewriting

- Computation transforms existing values into resulting values
- In λ-Calculus
  - Represent initial values as λ-Calculus expressions
  - Transform λ-Calculus expressions into new expressions
  - Interpret resulting λ-Calculus expression to get result
- Transformations $\equiv$ rewriting expressions according to rules
- If rule ρ transforms λ-Calculus expression E_1 to E_2 write

$$E_1 \quad \stackrel{\rho}{\rightarrow} \quad E_2$$

Preservation of Semantics

- Every transformed expression preserves semantics of expression
  - Represent: 2+0 as λ-calculus expression E_1
  - Transform $E_1 \stackrel{\rho}{\rightarrow} E_2$
    - Eg, transform "2+0" into "2", or transform "4*(2+0)" into "4*2"
  - Now E_2 must still represent 2+0 (perhaps "2")

Syntax

- Lambda Calculus expressions use
  - lower case letters: { a, b, c, d, ... }
  - four symbols: λ | ( )
- Each letter represents a function
- Three kinds of expressions
  - function constant
  - function definition
  - function application

Examples of Lambda Calculus Expressions

f	a *function* identifer
(f g)	an application of *function* f to g
(λx| (x y))	definition of *function* with parameter x and body (x y) Body is an application!
(λy | (λx| (y (y x))))	definition of *function* with parameter y and body (λx | (y (y x)))

Formal BNF Grammar

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

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

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

$\langle$identifier$\rangle$:= a | b | c | $\cdots$

The λ Definition

- Function definitions have the form: ($\lambda\langle$identifier$\rangle$ | $\langle$expression$\rangle$)
- λ is followed by a *single* identifier,
  called a *formal parameter* or *variable*
- When the λ is applied to an argument **E**,
  the *formal parameter* will bind to **E**.
  Below the $\langle$identifier$\rangle$x binds to value y

  ((λx | $\langle$body$\rangle$) y)
- Appearances of the identifier in the body of the λ are called *instances*
  - Every instance refers to the same expression — the one λ was called on. In the example below, each instance of x refers to y.

    $(\quad (\lambda x \mid (\underbrace{x}_{\text{instance}} \quad \underbrace{x}_{\text{instance}})) \quad y \quad)$

Notational Conveniences

- Where order of operations is clear, can drop brackets
- Can use spacing arbitrarily to aid readability
  - In function definition

    $(\lambda x|\ \texttt{(x y)}) \equiv (\lambda x|\ \texttt{x y}) \equiv (\lambda x|\ \texttt{xy})$
  - In function application

    (f g) $\equiv$ f g $\equiv$ fg

Associativity of λ-calculus operators

- *Associative* operators like integer addition can be composed in any order

  (1+2)+3 = 1 + (2+3)
- *Non-associative* operators like subtraction *cannot* be composed in any order

  (5-3)-2 $\neq$ 5-(3-2)
- λ-application is not associative
  (λC must be able to represent non-associative functions!)
- By convention, λ-application is *left-associative*... terms group *from the left*

  f g h $\equiv$ ((f g) h) $\neq$ (f (g h))

More Left-associativity Examples

f g h $\stackrel{?}{\equiv}$ (f g) h YES

(λa | (a (a b))) $\stackrel{?}{\equiv}$ (λa | a a b) NO

(λz | (a (λy| b))) $\stackrel{?}{\equiv}$ (λz | a (λy| b)) YES

a b (c d) $\stackrel{?}{\equiv}$ a b c d NO

(a b) c d $\stackrel{?}{\equiv}$ a b c d YES

Free and Bound Variables

- An instance of variable *v* is *bound* in expression *E* when it is:
  - a formal parameter of a λ
  - it is enclosed by a λ with parameter *v* within the expression *E*

$$(\underbrace{\lambda x}_{\text{bound}} \mid z)$$

$$(\underbrace{\lambda x}_{\text{bound}} \mid \underbrace{x}_{\text{bound}})$$

$$(\underbrace{\lambda x}_{\text{bound}} \mid y\ \underbrace{x}_{\text{bound}}\ z)$$

$$(\underbrace{\lambda x}_{\text{bound}} \mid (\lambda \underbrace{y}_{\text{bound}} \mid \underbrace{x}_{\text{bound}}\ \underbrace{y}_{\text{bound}}))$$

$$(\underbrace{\lambda x}_{\text{bound}} \mid (\lambda \underbrace{y}_{\text{bound}} \mid \underbrace{x}_{\text{bound}}\ \underbrace{y}_{\text{bound}}))$$

Free and Bound Variables

- A variable that is not bound is *free*

$$\underbrace{\mathtt{y}}_{\mathtt{free}}$$

$$(\ \underbrace{\lambda \mathtt{x}}_{\mathtt{bound}} \mid \underbrace{\mathtt{y}}_{\mathtt{free}}\ )$$

$$(\ \underbrace{\mathtt{y}}_{\mathtt{free}}\ (\ \underbrace{\lambda \mathtt{y}}_{\mathtt{bound}} \mid \underbrace{\mathtt{y}}_{\mathtt{bound}}\ )\ )$$

$$(\ \underbrace{\lambda \mathtt{x}}_{\mathtt{bound}} \mid (\mid \underbrace{\mathtt{q}}_{\mathtt{free}} \underbrace{\mathtt{y}}_{\mathtt{free}}\ )\ )$$

- Bound and free instances of same variable within an expression:

$$(\ \underbrace{\lambda \mathtt{x}}_{\mathtt{bound}} \mid \underbrace{\mathtt{x}}_{\mathtt{bound}} \underbrace{\mathtt{y}}_{\mathtt{free}}\ (\ \underbrace{\lambda \mathtt{y}}_{\mathtt{bound}} \mid \underbrace{\mathtt{x}}_{\mathtt{bound}} \underbrace{\mathtt{y}}_{\mathtt{bound}} \underbrace{\mathtt{z}}_{\mathtt{free}}\ )\ \underbrace{\mathtt{y}}_{\mathtt{free}}\ )$$

More on Variables in λ-calculus

- Free variables in an expression can be later bound in an enclosing expression

$$(\ \underbrace{\lambda \mathtt{x}}_{\mathtt{bound}} \mid \underbrace{\mathtt{x}}_{\mathtt{bound}} \underbrace{\mathtt{y}}_{\mathtt{free}}\ )$$

$$(\ \underbrace{\lambda \mathtt{y}}_{\mathtt{bound}}\ (\ \underbrace{\lambda \mathtt{x}}_{\mathtt{bound}} \mid \underbrace{\mathtt{x}}_{\mathtt{bound}} \underbrace{\mathtt{y}}_{\mathtt{bound}}\ ))$$

- Variables in λ-calculus derive their meaning from the argument the enclosing λ is applied to
  - They cannot be "assigned" a new "value"

More Convenience: Collapsing Enclosing λ's

- The scope of a λ is the expression to which its bindings apply

`(λx | (λy|y) x)) ((λw | w) v )`

- Scope of **outer** λ includes (λy|y) x
- If the scopes of nested λ's coincide, the arguments can be coalesced into a multi-argument λ

`( λx | (λy | x y)) ≡ (λxy | xy)`

- Just notational convenience!!

λ-calculus Computation

- λ-calculus computation is ...
  Reducing complex expression to "simpler" form
  - `( (λz | (z y)) w ) →(w y)`
    [Every occurance of `z` →`w`]
  - `( (λz | (z y)) (λw | w) ) →( (λw | w) y) )`
    [Every occurance of `z` →`(λw | w)`]
    `( (λw | w) y) ) → y`
    [Every occurance of `w` →`y`]
- Need to
  "Replace every occurance of `z` in `(z y)` with `(λw | w)`"
  "Replace every occurance of ⟨identifier⟩ in ⟨expression⟩ with ⟨expression'⟩"
-

Substitution

- λ-calculus rules uses a special substitution
- Generally: write substitution of x for y in expression ⟨E⟩as: $[x/y]$ ⟨E⟩

  [x/y] (z y) $\rightarrow$ (z x)
  [(a b)/y] (λz |(z y)) $\rightarrow$ (λz (z (a b)))

- In λ-calculus, only *free* variables are replaced:

  [x/y] (z (λy|z y)) $\rightarrow$ (z (λy|z y))

Legal Substitution

- Legal substitutions do not change meaning of an expression
  - Legal: Substitute x for y

    [x/y] (λz | yz) $\rightarrow$ (λz | xz)

- *Illegal substitutions* introduce bindings not present in original expressions

    [z/y] (λz | yz) $\not\rightarrow$(λz|zz)
  - Illegal because variable named y in (λz | yz) was free but now, as z, is bound

    [(λx|xz) / y] (λz|yz) $\not\rightarrow$ (λz|(λx|xz)z)
  - Illegal because z was free in (λx|xz) but now is bound

β (Beta Rule): Function Application

- A *function application* $((\lambda x \mid \langle E \rangle)\ \langle F \rangle)$ has function $(\lambda x \mid \langle E \rangle)$ and argument $\langle F \rangle$
- β-rule: apply $(\lambda x \mid \langle E \rangle)$ to $\langle F \rangle$

  $\equiv$

  substitute $\langle F \rangle$ for every *free* occurence of x in body $\langle E \rangle$

  $$\texttt{eval[(}\lambda\texttt{x | }\langle \texttt{E} \rangle\texttt{) }\langle \texttt{F} \rangle\texttt{)]}$$
  $$\equiv [\langle \texttt{F} \rangle/\texttt{x}]_\lambda\ \texttt{E} \quad \textit{if}\ [\langle \texttt{F} \rangle/\texttt{x}]_\lambda \texttt{ is legal}$$
- β defines a relationship between manipulation of symbols and a computation

Substitution Legality and the β-rule

- β-rule starts with application: $(\ (\lambda x \mid \langle E \rangle)\ \langle F \rangle\ )$
- Substitution is *illegal* only if
  $\exists$ free occurences of variables in $\langle F \rangle$ that would become bound in $\langle E \rangle$
- Later, a way to fix things when a substitution would be illegal

β example: constant argument

(λf | (f x)) s

$\xrightarrow{\beta}$ [s/f] (f x)

free variables in s that would get bound?

No, go ahead and substitute

≡(s x)

Can we do more? No - normal form

β example: λargument

((λf | (f x)) (λy | y))

$\xrightarrow{\beta}$ [(λy | y)/f] (f x)

Free vars in (λy | y) get bound? No.

≡ ((λy| y) x)

((λy| y) x)

$\xrightarrow{\beta}$ [x / y] y

Free vars in x get bound? No.

≡ x

β example: constant substitutions

((λf | (f (f x))) s)

$\xrightarrow{\beta}$ [s /f] (f (f x))

Free vars in s get bound?No.

$\equiv$(s (s x))

Can we do more? No - in normal form

β example: λ substitutions

((λf | (f (f x))) (λy | y))

$\xrightarrow{\beta}$*[(λy | y) / f] (f (f x))*

Free vars in (f (f x)) get bound? No.

$\equiv$((λy| y) ((λy | y) x))

Are we done? No

((λy| y) ((λy | y) x))

$\xrightarrow{\beta}$((λy| y) [x / y] y)

((λy| y) x)

Now are we done? No

((λy| y) x) $\xrightarrow{\beta}$ [x / y] y

$\rightarrow$x

β example: complex multiple substitution

```
( (λf | (f (f x)))  (λy | (g (g (g y))) ) )
≡(λf | (f (f x)))  (λy | (g (g (g y))) )
β→ [(λy | (g (g (g y))) )/f] (f (f x))
Free vars in (λy | (g (g (g y))) ) get bound? No
≡( (λy | (g (g (g y)))) ( (λy | (g (g (g y)))) x)))

(   (λy | (g (g (g y))))  ((λy | (g (g (g y)))) x)))
β→( (λy | (g (g (g y))))  [x/y] (g (g (g y))))  ))
≡ ( (λy | (g (g (g y))))   (g (g (g x))) )

( (λy | (g (g (g y)))) (g (g (g x))) )
β→[(g (g (g x))) / y ] (g (g (g y)))
→(g (g (g (g (g (g x))) )))
```


A formal definition of β-substitution

- Let ⟨E⟩, ⟨F⟩ and ⟨G⟩ be λ-calculus expressions; x and y be distinct λ-calculus identifiers (constants)

```
[⟨E⟩ / x] x → ⟨E⟩
[⟨E⟩ / x] y → y
[⟨E⟩ / x] (⟨F⟩ ⟨G⟩) → ( [⟨E⟩ / x] ⟨F⟩    [⟨E⟩ / x] ⟨G⟩  )
[⟨E⟩ / x] (λx | ⟨F⟩) → (λx | ⟨F⟩)
[⟨E⟩ / x] (λy | ⟨F⟩) where ⟨E⟩ has no free instances of y
    → (λy | [⟨E⟩ / x] ⟨F⟩)
```

Variable Names in λ-calculus

- The identifier used to represent a BOUND variable is irrelevant
- Meaning of variable based on the λ that introduces it ...
  and how it is used in λ's body
- If we change the identifier used in a formal parameter and all of its bound occurences, the meaning of the expression is unaltered

$(\lambda x \mid x) \stackrel{?}{\equiv} (\lambda y \mid y)$ YES!

$(\lambda x \mid x) \stackrel{?}{\equiv} (\lambda x \mid y)$ No!

$(\lambda x \mid (\lambda y \mid x\ y)) \stackrel{?}{\equiv} (\lambda a \mid (\lambda b \mid a\ b))$ YES!

$(\lambda x \mid (\lambda y \mid x\ y)) \stackrel{?}{\equiv} (\lambda a \mid (\lambda b \mid b\ a))$ NO!

Variables in λ-calculus

$(\lambda x \mid (\lambda y \mid x\ y)) \stackrel{?}{\equiv} (\lambda y \mid (\lambda x \mid y\ x))$ YES!

$(\lambda x \mid (\lambda y \mid x\ y)) \stackrel{?}{\equiv} (\lambda y \mid (\lambda x \mid x\ y))$ NO!

$(\lambda x \mid (\lambda w \mid w)\ x) \stackrel{?}{\equiv} (\lambda y \mid (\lambda w \mid w)\ y)$ YES !

$(\lambda x \mid (\lambda y \mid y)\ x) \stackrel{?}{\equiv} (\lambda y \mid (\lambda y \mid y)\ y)$

YES (same as above!) ... but confusing!

Think ...$(\lambda y \mid (\lambda y \mid y)\ y)$

α (Alpha Rule): Motivation

- Theβ-rule cannot be applied in ...
  ((λy | (λz | yz)) z)
  $\xrightarrow{\beta}$[z / y] (λz | yz)
  $\not\equiv$ (λz | zz) Why not?
  z was free in z but *bound* in the result $\Rightarrow$ substitution is illegal!

- (λy (λz|yz)) (λx|xz)
  $\xrightarrow{\beta}$[(λx|xz)/ y] (λz|yz)
  $\not\equiv$ (λz|(λx|xz)z)

- But, variable identifiers in and of themselves are irrelevant

- The α-rule changes variable identifiers without altering meaning

α (Alpha Rule): Renaming

- α-rule: substitute a new identifier for the instances of any bound variable... as long as the substitution is legal

- Just choose an identifier *not* used in the current expression, ... and substitution guaranteed to be legal

  (λz|yz) $\xrightarrow{\alpha:q/z}$ (λq|yq)

  - Note: Replace the formal parameter *and* EVERY instance of z with q
  - Note: q is a NEW variable, never used ...

α (Alpha Rule): Examples

(λa | b (λc | c a) d) $\stackrel{\alpha:z/a}{\longrightarrow}$ (λz | b (λc | c z) d)
Legal

(λx | (λy | x y z)) $\stackrel{\alpha:y/z}{\longrightarrow}$ (λx | (λy | x y y))
Illegal

Formal definition of α

- Let ⟨E⟩ and⟨F⟩ be λ-calculus expressions;
  x and y be distinct λ-calculus constants
- Let z be a newly generated λ calculus constant

[⟨E⟩ / x] (λy | ⟨F⟩) → (λz | [⟨E⟩ /x] [z/y] ⟨F⟩)

Using α and β Together I

(λy (λz|yz)) (λx|xz)

- We could use β rule to simulate applying the function

$\xrightarrow{\beta}$[(λx|xz)/ y] (λz|yz)

- Legal substitution? No. Free z in (λx|xz) becomes bound
- Use α rule to rename variable

$\xrightarrow{\alpha}$[(λx|xz)/ y] [q/z] (λz|yz)≡[(λx|xz)/ y] (λq|yq)

Using α and β Together II

- Now we can apply β-substitution

[(λx|xz)/ y] (λq|yq)≡ (λq| (λx|xz) q)

(λq| (λx|xz) q)
$\xrightarrow{\beta}$ (λq| [q /x] xz) ≡(λq| q z)

α and β Are Complete

BP: it is still unclear to me if this is more than a conjecture - as in the 'Church-Turing Thesis".

- ***We can represent any calculation as a λ calculus expression!!***
  - Turing Equivalents!
- Computation $\equiv$Apply α, β rules (many times!) to reduce given expression to "unreducible" form
- Interpet value of resulting expression as result of computation
- Computation requires only two rules

η (Eta Rule): Null Application

- Special case of the β-rule: $(\lambda x \,|\langle \mathrm{E}\rangle)\, v \ \stackrel{\beta}{\rightarrow}\ [v/x]\,\langle \mathrm{E}\rangle$
- Accelerates Rule 5 of β substitution
- If x does not appear as a free variable in$\langle$E$\rangle$, then $\langle$E$\rangle$ doesn't change
- η-rule:
  - If x is not free in $\langle$E$\rangle$ then $((\lambda \mathtt{x} \mid \langle \mathrm{E}\rangle)\ \mathtt{v}) \stackrel{\eta}{\rightarrow} \langle \mathrm{E}\rangle$

$((\lambda \mathtt{a} \mid \mathtt{c\ d})\ \mathtt{q}) \rightarrow (\mathtt{c\ d})$
$(\lambda \mathtt{x} \mid (\lambda \mathtt{x} \mid \mathtt{x\ y}))\ \mathtt{v} \rightarrow (\lambda \mathtt{x} \mid \mathtt{x\ y})$

λ-calculus Interpreters

- To implement a λ-calculus interpreter
  - Must determine if each variable is free or bound
    ... to determine potential clashes with free variables
  - Faster to determine the status of variable $\mathtt{x}$ in $\langle \mathtt{E} \rangle$, than to "build" a new expression without any changes
    $\Rightarrow \eta$-rule

On Reductions

- λ-calculus reduces expressions to "simpler" expressions using β and η rules
  - Why scare quotes?
- β-rule and η-rule are called *reductions* (α is not a reduction)
- If we can obtain $\langle \text{N} \rangle$ from $\langle \text{M} \rangle$ using a squence of β and η operations,
  then $\langle \text{M} \rangle$ is *reducible* to $\langle \text{N} \rangle$
- An expression that can be reduced is called a *redux*
- Can only reduce applications that contain function definitions
  - Cannot reduce `f, (f g), (λf | (f g))`
  - Can reduce `( (λx | (w x)) y)`
- An expression containing no reduxes is in *normal form* (i.e. a completed calculation)

Theoretical Questions

- So "interpretation" ≡"reducing to normal form"
- Questions...
  - Is there more than one way to reduce an expression?
  - Is there one unique reduction for every expression?
  - Is every expression reducible?
  - If not, what are the implications?
- First topic: Order of reductions...

Order of Reductions: Normal I

- Normative Order: leftmost application first
- Which is leftmost function?

(λx|(λy |x)) ((λu | z) u)

$\underbrace{\texttt{(λx|(λy|x))}}_{\texttt{leftmost}}$ ((λu | z) u)

(λ$\underline{x}$|(λy |$\underline{x}$)) ((λu | z) u)

$\xrightarrow{\beta}$[((λu | z) u) / $\underline{x}$] (λy |$\underline{x}$)

Free vars in ((λu | z) u)get bound? No

≡(λy | ((λu | z) u))

Order of Reductions: Normal II

```
(λy |  ((λu | z) u ))   Left application?
(λy | ((λu|z) u))
       leftmost
(λy | ((λu | z) u ) )
```
$\xrightarrow{\beta}$ `(λy | [u / u ] z )`

Any free vars in u get bound? No.

`→(λy | z )`

Done? Yes - normal form

Order of Reductions: Applicative I

- Applicative Order: innermost *application* first
- Like LISP: evaulate arguments first, then apply function

```
(λx|(λy |x)) ((λu | z) u )
(λx|(λy |x)) (  (λu|z)   u)
                innermost
(λx|(λy |x)) ((λu | z) u )
```
$\xrightarrow{\beta}$ `(λx|(λy |x)) [u /u ] (λu | z)`

any free vars in `u` get bound?No.

`≡ (λx|(λy |x)) z`

Done? Nope

Order of Reductions: Applicative II

(λx|(λy |x)) z   Innermost?

$\underbrace{\texttt{(λx|(λy|x))}}_{\texttt{innermost}}$ z

(λx|(λy |x)) z

$\xrightarrow{\beta}$[z / x] (λy |x)

Any free vars in x get bound? No

≡ (λy | z)

Done? Yes - normal form

Order of Reductions: Comment

- You may choose
  - normative (left-most legal application) or
  - applicative order (innermost legal application) or
  - ...
- However, since λ calculus is left-associative,
  - at any given level within an expression, you must reduce the leftmost of a series of applications first
- So in: abc(cde)
  - May apply c to d (applicative) or a to b (normative)
  - CANNOT apply b to c nor c to (cde) nor d to e (violation of left-associativity)

Church and Rosser Theorem

- Let ⟨A⟩,⟨B⟩,⟨C⟩,⟨D⟩ be λ-calculus expressions and $\xrightarrow{1}$, $\xrightarrow{2}$, $\xrightarrow{3}$ and $\xrightarrow{4}$ be reductions of *zero* or more steps
- Church and Rosser Theorem I
  - If $\langle A\rangle \xrightarrow{1} \langle B\rangle$ and $\langle A\rangle \xrightarrow{2} \langle C\rangle$,
  - Then $\exists$ ⟨D⟩ $\xrightarrow{3}$ and $\xrightarrow{4}$ s.t. $\langle B\rangle \xrightarrow{3} \langle D\rangle$ and $\langle C\rangle \xrightarrow{4} \langle D\rangle$
- i.e., different reductions of ⟨A⟩ can always be reduced to the same expression (function)

Uniqueness Corollary

- Corollary: Given two reductions $\langle A\rangle \xrightarrow{1} \langle B\rangle$ and $\langle A\rangle \xrightarrow{2} \langle C\rangle$
  - If ⟨B⟩ and ⟨C⟩ are in normal form, neither can be reduced further
  - By Church and Rosser I, we can reduce ⟨C⟩ and ⟨B⟩ to an identical form in *zero* or more steps
  - Since both ⟨C⟩ and ⟨B⟩ are irreducible, the required reduction must be of length zero and ⟨C⟩ and ⟨B⟩ are identical
  - **All reductions that result in a normal form, result in the same unique normal form !**
- ... does every reduction result in normal form???

Existence Theorem

- Church and Rosser Theorem II
  - If ⟨A⟩→⟨B⟩ and ⟨B⟩ is in normal form
    then ⟨A⟩→⟨B⟩ by *normative order* reduction
- If ⟨A⟩ can be reduced to a normal form,
  it can be found by normal order reduction
- Not every expression has a normal form

$(\lambda x|\ x\ x)\ (\lambda\ x|\ x\ x)$
$(\lambda \underline{x}|\ \underline{x}\ \underline{x})\ (\lambda\ x|\ x\ x)$
$\rightarrow\ (\lambda x|\ x\ x)\ (\lambda\ x|\ x\ x)$

- Because reductions are not guaranteed to terminate,
  the equivalence of λ-calculus expressions is undecidable
- This result predates the halting problem !

Reduction Orders as Parameter Types

- Applicative order reduction evaluates innermost applications first
  - ≈ evaluating arguments before passing them
  - Can be interpreted as "call by value"
- Normative order reduction evaluates leftmost applications first
  - ≈passing unevaluated expressions to function
  - Can be interpreted as "call by name"
  - Passed-in expressions must still be evaluated in body of function

Completeness of Applicative vs. Normal Order

- The argument to $(\lambda x \mid y)$ does not matter
  - $((\lambda x \mid y)\langle E\rangle) \to y$ for any $\langle E\rangle$
  - Here, expression $\langle E\rangle$ is an *unneeded* argument
  - η-reductions
- Applicative order may evaluate *unneeded* arguments
  - If argument does not have a normal form, evaluation of arguments will not halt
- Normal order does not evaluate unneeded arguments
  - If only unneeded arguments lack a normal form, then Normal order will find a normal form
- $\exists$ formulas that have a normal form that can be found by normal order reduction,
  but that cannot be found by applicative order reduction

Reducible by Normal Example

(λz (λy| y)) ((λx | x x) (λx | x x))

$\underbrace{\texttt{(}\lambda\texttt{z (}\lambda\texttt{y | y))}}_{\text{leftmost application}}$ ((λx | x x) (λx | x x))

(λz (λy| y)) ((λx | x x) (λx | x x))

$\xrightarrow{\beta}$ [((λx | x x) (λx | x x))/ z] (λz (λy| y))

Any free vars get bound? No.

$\equiv$ (λy| y)

Irreducible by Applicative Example

- Under Applicative order

(λz (λy| y)) ((λx | x x) (λx | x x))

(λz (λy| y)) ((λx | x x) (λx | x x))

innermost application

(λz (λy| y)) ((λx | x x) (λx | x x))

$\xrightarrow{\beta}$(λz (λy| y)) [(λx | x x) / x] x x

Any free vars in (λx | x x) get bound? No.

≡(λz (λy| y)) ((λx | x x) (λx | x x))

Notice anything fishy here?

We are back to what we started with!

Example 1 : Normal Order

(λx | (λy x | x) z)) (λx | x y)

First step? Identify *leftmost* applicable function

(λx | (λy x| x) z)) (λx | x y)

leftmost

(λx | (λy x | x) z)) (λx | x y)

Recall (λy x | x) means (λy | (λx | x))

(λx | (λy | (λx | x) z))) (λx | x y)

$\xrightarrow{\beta}$[(λx | x y) / x] (λy | (λx | x) z)

Free vars in (λx | x y)? get bound?

No free instances of x within (λy | (λx | x) z)

≡ (λ y | (λx | x) z)

(λy | (λx | x) z) $\xrightarrow{\eta}$ (λx | x)

Example 1 : Applicative

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

First step? Identify *innermost* applicable function

$(\lambda x \mid \underbrace{(\lambda y\ x \mid x)\ z)}_{\texttt{innermost}}\ (\lambda x \mid x\ y)$

Recall: $(\lambda y\ x \mid x)$ means $(\lambda y \mid (\lambda x \mid x))$

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

$\xrightarrow{\beta}\ (\lambda x \mid [z\ /\ y\ ](\lambda x \mid x)\ (\lambda x \mid x\ y)$

No free instances of y in $(\lambda x \mid x)$

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

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

$\xrightarrow{\eta}\ (\lambda x \mid x)$

Example 2: Normal Order I

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

First task: find leftmost applicable function

$\underbrace{(\lambda x \mid (\lambda y \mid x))}_{\texttt{leftmost}}\ ((\lambda x \mid y)\ x)$

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

$\xrightarrow{\beta}[((\lambda x \mid y)\ x)\ /\ x]\ (\lambda y \mid \underline{x})$

Free vars in $((\lambda x \mid y)\ x)$ get bound? YES!

$\not\rightarrow(\lambda y \mid ((\lambda x \mid y)\ x))$

Use α rule.

$\xrightarrow{\alpha}[((\lambda x \mid y)\ x)\ /x\ ]\ [z/y]\ (\lambda y \mid \underline{x})$

$\equiv\ [((\lambda x \mid y)\ x)\ /x\ ]\ (\lambda z \mid \underline{x})$

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

Example 2: Normal Order II

(λz | (λx | y) x)

(λz | $\underbrace{\texttt{(λx|y)}}_{\texttt{leftmost}}$ x)

(λz | ((λx | y) x))

$\xrightarrow{\eta}$ (λz | y)

Example 2: Applicative I

(λx | (λy | x)) ((λx | y) x)

First step? Find innermost application

(λx | (λy | x)) $\underbrace{\texttt{((λx | y) x)}}_{\texttt{innermost}}$

(λx | (λy | x)) ((λx | y) x)

$\xrightarrow{\eta}$(λx | (λy | x)) y

Example 2: Applicative II

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

$\underbrace{(\lambda x|(\lambda y|x))}_{\text{innermost}}\ y$

$\xrightarrow{\beta} [y/x]\ (\lambda y \mid x)$

Free vars get bound? `Yes`

$\xrightarrow{\alpha} [y/x][z/y](\lambda y \mid x)$

$\equiv [y/x](\lambda z \mid x)$

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

Example 3: Normal I

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

First step? `Find leftmost application`

$\underbrace{(\lambda x\ y \mid y)}_{\text{leftmost}}\ ((\lambda x \mid x\ x)\ (\lambda x \mid x\ x))\ a$ Re-call: $(\lambda x\ y \mid y) \equiv (\lambda x \mid (\lambda y \mid y))$

$(\underbrace{(\lambda x \mid (\lambda y \mid y))}_{\text{leftmost}}\ ((\lambda x \mid x\ x)\ (\lambda x \mid x\ x)))\ a$

$\xrightarrow{\eta} (\lambda y \mid y)\ a$

$\xrightarrow{\beta} [a/y]\ y \equiv a$

Example 3: Applicative I

$(\lambda$x y | y) ((λx | x x) (λx | x x)) a
First step? Find innermost application.
$(\lambda$x y | y) $\underbrace{((\lambda\text{x | x x})}_{\text{innermost}}$ (λx | x x)) a
$\xrightarrow{\beta}$((λx y | y) [(λx | x x)/ x] (x x)) a
Will free vars in get (λx | x x) bound? No free vars!
$\equiv$ (λx y | y) ((λx | x x)(λx | x x)) a
We get the original expression back again!

Shortcuts for Multi-argument λ's

(λx y z | $\langle$E$\rangle$) $\langle$A$\rangle$ $\langle$B$\rangle$ $\langle$C$\rangle$
$\equiv$ (λx | (λy | (λz | $\langle$E$\rangle$)) $\langle$A$\rangle$ $\langle$B$\rangle$ $\langle$C$\rangle$

$\xrightarrow{\beta}$ [$\langle$A$\rangle$/x] (λy | (λz | $\langle$E$\rangle$)) $\langle$B$\rangle$ $\langle$C$\rangle$
If $\langle$A$\rangle$ has free y or z, must rename (λy | (λz | $\langle$E$\rangle$))
$\xrightarrow{\beta}$ [$\langle$B$\rangle$/y] (λz | $\langle$E$\rangle$)
If $\langle$B$\rangle$ has free z, must rename (λz | $\langle$E$\rangle$)
$\xrightarrow{\beta}$ [$\langle$C$\rangle$/z] $\langle$E$\rangle$
If $\langle$C$\rangle$ has free var bound in $\langle$E$\rangle$, must rename ...

Example of Multi-argument λ's

- Our basic solution method

  (λ x y | x y) (⟨N⟩ y) ⟨M⟩
  ≡ (λ x | (λy | x y)) (⟨N⟩ y)⟨M⟩
  $\xrightarrow{\beta}$[(⟨N⟩ y) / x] (λy | x y) ⟨M⟩
  Free vars in (⟨N⟩ y) get bound? Yes!
  Must rename y in (λy | x y). Say z
  $\xrightarrow{\alpha}$[(⟨N⟩ y) / x] [z/y] (λy | x y) ⟨M⟩
  ≡[(⟨N⟩ y) / x] (λz | x z) ⟨M⟩
  ≡(λz | (⟨N⟩ y) z) ⟨M⟩
  $\xrightarrow{\beta}$[⟨M⟩ / z] (⟨N⟩ y) z ≡ (⟨N⟩ y) ⟨M⟩

- Note: we replaced y with z,
  but then immediately replace z with ⟨M⟩

Example of Multi-argument λ's

- In general, can perform multiple substitutions in parallel

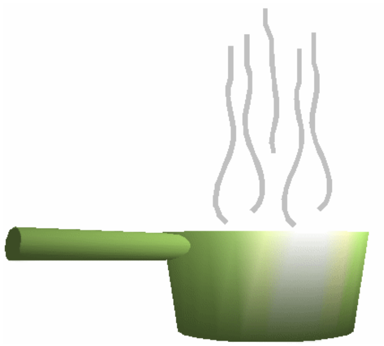
- *If substituting in parallel*,
  given (λx | (λy ...)) ⟨A⟩ ⟨B⟩,
  we do not have to check for free y's in ⟨A⟩ as ⟨B⟩ will be substituted for the "(λy" and any free y's in ⟨A⟩ will remain free.

- Example done with multiple substitution

  (λ x y | x y) (⟨N⟩ y) ⟨M⟩
  $\xrightarrow{\beta}$[(⟨N⟩ y)/x, ⟨M⟩/y] (x y)
  ≡ (⟨N⟩ y) ⟨M⟩

- N.B: still need to check for free vars that get bound when considering substitution of ⟨B⟩ in the body of the (λy ...) clause.

Curried functions



- Can represent n-ary functions as nested unary functions
- (λx y | ⟨E⟩) a b
  ≡ (λx (λy ⟨E⟩)) a b
- Can treat an *n*-ary function as a unary function that returns an *n-1*-ary function
- Treating *n*-ary function as unary function that returns a function is called *currying*