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:

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

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

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

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

Computation as Rewriting

- Computation transforms existing values into resulting values
- In λ-Calculus

Computation as Rewriting

- Computation transforms existing values into resulting values
- In λ-Calculus
  - Represent initial values as λ-Calculus expressions

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

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

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

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

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

Preservation of Semantics

- Every transformed expression preserves semantics of expression

Preservation of Semantics

- Every transformed expression preserves semantics of expression
  - Represent: 2+0 as λ-calculus expression E_1

Preservation of Semantics

- Every transformed expression preserves semantics of expression
  - Represent: 2+0 as λ-calculus expression E_1
  - Transform $E_1 \xrightarrow{\rho} E_2$
    - Eg, transform “2+0” into “2”, or transform “4*(2+0)” into “4*2”

Preservation of Semantics

- Every transformed expression preserves semantics of expression
  - Represent: 2+0 as λ-calculus expression E_1
  - Transform $E_1 \xrightarrow{\rho} E_2$
    - 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 | ( )

Syntax

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

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`
$(\lambda x|$ `(x y))`	definition of *function* with parameter `x` and body `(x y)` Body is an application!
$(\lambda y \mid (\lambda x|$ `(y (y x))) )`	definition of *function* with parameter `y` and body $(\lambda$`x	(y (y x)))`

Formal BNF Grammar

$$\langle\texttt{expression}\rangle\texttt{:=}\langle\texttt{identifier}\rangle \texttt{ | } \langle\texttt{application}\rangle \texttt{ | } \langle\texttt{function}\rangle$$

Formal BNF Grammar

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

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

Formal BNF Grammar

⟨expression⟩:=⟨identifier⟩ | ⟨application⟩ | ⟨function⟩

⟨application⟩:= "(" ⟨expression⟩ ⟨expression⟩ ")"

⟨function⟩ := "(λ" ⟨identifier⟩ "|" ⟨expression⟩ ")"

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: $(\lambda\langle\texttt{identifier}\rangle \mid \langle\texttt{expression}\rangle)$

The λ Definition

- Function definitions have the form: $(\lambda\langle\texttt{identifier}\rangle \mid \langle\texttt{expression}\rangle)$
- λ is followed by a *single* identifier,
  called a *formal parameter* or *variable*

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`

  `(` `(`λ`x` `|` $\langle$`body`$\rangle$`)` `y` `)`

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`.

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

Notational Conveniences

- Where order of operations is clear, can drop brackets

Notational Conveniences

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

Notational Conveniences

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

$$(\lambda x| \ \texttt{(x y)}) \quad \equiv \quad (\lambda x| \ \texttt{x y}) \quad \equiv \quad (\lambda x| \ \texttt{xy})$$

Notational Conveniences

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

$$(\lambda x \mid \; (\mathtt{x} \;\; \mathtt{y})) \quad \equiv \quad (\lambda x \mid \; \mathtt{x} \;\; \mathtt{y}) \quad \equiv \quad (\lambda x \mid \; \mathtt{xy})$$

  - In function application

$$(\mathtt{f} \;\; \mathtt{g}) \quad \equiv \quad \mathtt{f} \;\; \mathtt{g} \quad \equiv \quad \mathtt{fg}$$

Associativity of λ-calculus operators

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

  `(1+2)+3 = 1 + (2+3)`

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)

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!)

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)` $\not\equiv$ `(f (g h))`

More Left-associativity Examples

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

More Left-associativity Examples

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

More Left-associativity Examples

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

$$\texttt{(}\lambda\texttt{a | (a (a b)))} \stackrel{?}{\equiv} \texttt{(}\lambda\texttt{a | a a b)}$$

More Left-associativity Examples

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

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

More Left-associativity Examples

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

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

$\texttt{(}\lambda\texttt{z | (a (}\lambda\texttt{y| b)))} \stackrel{?}{\equiv} \texttt{(}\lambda\texttt{z | a (}\lambda\texttt{y| b))}$

More Left-associativity Examples

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

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

$\texttt{(}\lambda\texttt{z | (a (}\lambda\texttt{y| b)))} \stackrel{?}{\equiv} \texttt{(}\lambda\texttt{z | a (}\lambda\texttt{y| b))}$ YES

More Left-associativity Examples

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

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

$\texttt{(}\lambda\texttt{z | (a (}\lambda\texttt{y| b)))} \stackrel{?}{\equiv} \texttt{(}\lambda\texttt{z | a (}\lambda\texttt{y| b))}$ YES

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

More Left-associativity Examples

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

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

$(\lambda\texttt{z | (a (}\lambda\texttt{y| b)))} \stackrel{?}{\equiv} (\lambda\texttt{z | a (}\lambda\texttt{y| b))}$ YES

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

More Left-associativity Examples

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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*

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

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

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

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

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

Free and Bound Variables

- A variable that is not bound is *free*

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

Free and Bound Variables

- A variable that is not bound is *free*

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

$$(\ \underbrace{\lambda\texttt{x}}_{\texttt{bound}}\ |\ \underbrace{\texttt{y}}_{\texttt{free}}\ )$$

Free and Bound Variables

- A variable that is not bound is *free*

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

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

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

Free and Bound Variables

- A variable that is not bound is *free*

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

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

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

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

Free and Bound Variables

- A variable that is not bound is *free*

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

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

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

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

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

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

More on Variables in λ-calculus

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

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

More on Variables in λ-calculus

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

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

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

More on Variables in λ-calculus

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

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

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

- Variables in λ-calculus derive their meaning from the argument the enclosing λ is applied to

More on Variables in λ-calculus

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

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

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

- 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)

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

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)) $\equiv$ (λxy | xy)

More Convenience: Collapsing Enclosing λ's

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

  $(\lambda x \mid (\lambda y | y)\ x))\ ((\lambda w \mid w)\ v\ )$

- Scope of **outer** λ includes $(\lambda y|y)\ x$

- If the scopes of nested λ's coincide, the arguments can be coalesced into a multi-argument λ

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

- Just notational convenience!!

λ-calculus Computation

- λ-calculus computation is ...
  Reducing complex expression to "simpler" form
-

λ-calculus Computation

- λ-calculus computation is ...
  Reducing complex expression to "simpler" form
  - `( (λz | (z y)) w )`
-

λ-calculus Computation

- λ-calculus computation is ...
  Reducing complex expression to "simpler" form
  - `( (λz | (z y)) w ) →(w y)`
    [Every occurance of `z` →`w`]

-

λ-calculus Computation

- λ-calculus computation is ...
  Reducing complex expression to "simpler" form
  - ((λz | (z y)) w) $\longrightarrow$(w y)
    [Every occurance of z $\longrightarrow$w]
  - ((λz | (z y)) (λw | w))
-

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)`]
-

λ-calculus Computation

- λ-calculus computation is ...
  Reducing complex expression to "simpler" form
  - ((λz | (z y)) w) $\longrightarrow$(w y)
    [Every occurance of z $\longrightarrow$w]
  - ((λz | (z y)) (λw | w)) $\longrightarrow$((λw | w) y))
    [Every occurance of z $\longrightarrow$(λw | w)]
    ((λw | w) y)) $\longrightarrow$ y
    [Every occurance of w $\longrightarrow$y]
-

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

Substitution

- λ-calculus rules uses a special substitution
- Generally: write substitution of `x` for `y` in expression $\langle E \rangle$ as: $[x/y] \langle \mathtt{E} \rangle$

  `[x/y] (z y)` $\longrightarrow$

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)` $\longrightarrow$ `(z x)`

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

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) → (z x)
[(a b)/y] (λz |(z y)) → (λz (z (a b)))
```

Substitution

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

  ```
  [x/y] (z y) → (z x)
  [(a b)/y] (λz |(z y)) → (λz (z (a b)))
  ```

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

  ```
  [x/y] (z (λy|z y)) →
  ```

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)` $\longrightarrow$ `(z x)`
  `[(a b)/y] (`λ`z |(z y))` $\longrightarrow$ `(`λ`z (z (a b)))`

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

  `[x/y] (z (`λ`y|z y))` $\longrightarrow$ `(z (`λ`y|z y))`

Legal Substitution

- Legal substitutions do not change meaning of an expression

Legal Substitution

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

$$[x/y]\ (\lambda z \mid yz) \rightarrow (\lambda z \mid xz)$$

Legal Substitution

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

    $[x/y]\ (\lambda z \mid yz) \longrightarrow (\lambda z \mid xz)$

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

Legal Substitution

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

    [x/y] (λz | yz) → (λz | xz)

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

    [z/y] (λz | yz) ↛ (λz|zz)

  - Illegal because variable named y in (λz | yz) was free but now, as z, is bound

Legal Substitution

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

    [x/y] (λz | yz) → (λz | xz)

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

    [z/y] (λz | yz) ↛(λz|zz)

  - Illegal because variable named y in (λz | yz) was free but now, as z, is bound

    [(λx|xz) / y] (λz|yz) ↛ (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 (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 $\langle E \rangle$

  eval[$(\lambda$x | $\langle$E$\rangle)$ $\langle$F$\rangle$)]

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 $\langle E \rangle$

  eval[$(\lambda x$ | $\langle E \rangle)$ $\langle F \rangle$)]

  $\equiv [\langle F \rangle / x]_\lambda$ E   *if* $[\langle F \rangle / x]_\lambda$ is legal

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 $\langle E \rangle$

  `eval[ (`λ`x | `$\langle$`E`$\rangle$`) `$\langle$`F`$\rangle$` ) ]`

  $\equiv$ `[`$\langle$`F`$\rangle$`/x]`$_\lambda$ `E` *if* `[`$\langle$`F`$\rangle$`/x]`$_\lambda$ `is legal`
- β defines a relationship between manipulation of symbols and a computation

Substitution Legality and the β-rule

- β-rule starts with application: $(\, (\lambda \mathsf{x} \mid \langle \mathsf{E} \rangle) \, \langle \mathsf{F} \rangle \,)$

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 $\langle E \rangle$

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

$(\lambda \underline{f} \mid (\underline{f}\ x))\ s$

$\xrightarrow{\beta}$

β example: constant argument

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

β example: constant argument

$(\lambda \underline{f} \mid (\underline{f}\ x))\ s$

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

free variables in s that would get bound?

β example: constant argument

$(\lambda \texttt{f} \mid (\texttt{f}\ \texttt{x}))\ \texttt{s}$

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

free variables in s that would get bound?

No, go ahead and substitute

$\equiv(\texttt{s}\ \texttt{x})$

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?

β example: constant argument

$(\lambda \underline{\mathtt{f}} \mid (\underline{\mathtt{f}}\ \mathtt{x}))\ \mathtt{s}$

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

free variables in `s` that would get bound?

No, go ahead and substitute

$\equiv (\mathtt{s}\ \mathtt{x})$

Can we do more? No - normal form

β example: λargument

`( (λf | (f x)) (λy | y) )`

$\xrightarrow{\beta}$

β example: λargument

((λf | (f x)) (λy | y))

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

Free vars in (λy | y) get bound?

β example: λargument

((λ<u>f</u> | (<u>f</u> x)) (λy | y))

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

Free vars in (λy | y) get bound? No.

$\equiv$ ((λy| y) x)

β example: λargument

((λf | (f x)) (λy | y))
$\xrightarrow{\beta}$ [(λy | y)/f] (f x)
Free vars in (λy | y) get bound? No.
$\equiv$ ((λy| y) x)

((λy| y) x)
$\xrightarrow{\beta}$ [x / y] y
Free vars in x get bound?

β example: λargument

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

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

Free vars in $(\lambda y \mid y)$ get bound? No.

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

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

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

Free vars in x get bound? No.

$\equiv$ x

β example: constant substitutions

((λf | (f (f x))) s)

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

Free vars in s get bound?

β example: constant substitutions

((λf | (f (f x))) s)

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

Free vars in s get bound?No.

≡(s (s x))

β example: constant substitutions

((λf | (f (f x))) s)

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

Free vars in s get bound?No.

≡(s (s x))

Can we do more?

β example: constant substitutions

((λf | (f (f x))) s)

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

Free vars in s get bound?No.

≡(s (s x))

Can we do more? No - in normal form

β example: λ substitutions

$$(\ (\lambda f \ | \ (f \ (f \ x))) \ (\lambda y \ | \ y) \ )$$

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

Free vars in $(f \ (f \ x))$ *get bound?*

β example: λ substitutions

$$(\ (\lambda f \ | \ (f \ (f \ x))) \ (\lambda y \ | \ y) \ )$$

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

Free vars in $(f \ (f \ x))$ *get bound?*

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))

β example: λ substitutions

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

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

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

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

Are we done?

β example: λ substitutions

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

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

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

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

Are we done? No

β example: λ substitutions

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

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

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

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

Are we done? No

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

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

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)

β example: λ substitutions

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

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

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

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

Are we done? No

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

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

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

Now are we done?

β example: λ substitutions

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

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

Free vars in $(f \; (f \; x))$ *get bound?* No.

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

Are we done? No

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

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

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

Now are we done? No

β example: λ substitutions

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

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

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

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

Are we done? No

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

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

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

Now are we done? No

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

$\longrightarrow$

β example: λ substitutions

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

$\stackrel{\beta}{\rightarrow} [\ (\lambda y \mid y) \ / \ f \ ] \qquad (f \ (f \ x))$

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

$\equiv(\ (\lambda \mathtt{y} \mid \ \mathtt{y}) \ ((\lambda \mathtt{y} \mid \mathtt{y}) \ \mathtt{x}))$

Are we done? No

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

$\stackrel{\beta}{\rightarrow}(\ (\lambda \mathtt{y} \mid \ \mathtt{y}) \quad [\mathtt{x} \ / \ \mathtt{y}] \ \mathtt{y} \ )$

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

Now are we done? No

$(\ (\lambda \mathtt{y} \mid \ \mathtt{y}) \ \mathtt{x}) \ \stackrel{\beta}{\rightarrow} \ [\mathtt{x} \ / \ \mathtt{y}] \ \mathtt{y}$

$\rightarrow \mathtt{x}$

β example: complex multiple substitution

```
( (λf | (f (f x)))  (λy | (g (g (g y))) ) )
```

β example: complex multiple substitution

((λf | (f (f x))) (λy | (g (g (g y)))))

≡(λf | (f (f x))) (λy | (g (g (g y))))

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))
```

β example: complex multiple substitution

((λf | (f (f x))) (λy | (g (g (g y)))))

≡(λf | (f (f x))) (λy | (g (g (g y))))

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

Free vars in (λy | (g (g (g y)))) get bound?

β example: complex multiple substitution

((λf | (f (f x))) (λy | (g (g (g y)))))

≡(λf | (f (f x))) (λy | (g (g (g y))))

$\xrightarrow{\beta}$ [(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)))

β example: complex multiple substitution

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

β example: complex multiple substitution

```
( (λf | (f (f x)))  (λy | (g (g (g y))) ) )
≡(λf | (f (f x)))  (λy | (g (g (g y))) )
```

$\xrightarrow{\beta}$ `[(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)))
```

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

```
≡ ( (λy | (g (g (g y))))   (g (g (g x))) )
```

β example: complex multiple substitution

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

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

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

Free vars in $(\lambda y \mid (g\ (g\ (g\ y)))\ )$ get bound? No

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

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

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

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

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

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

$\longrightarrow$

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 $\langle \mathsf{E} \rangle$, $\langle \mathsf{F} \rangle$ and $\langle \mathsf{G} \rangle$ be λ-calculus expressions; x and y be distinct λ-calculus identifiers (constants)

A formal definition of β-substitution

- Let $\langle$E$\rangle$, $\langle$F$\rangle$ and $\langle$G$\rangle$ be λ-calculus expressions;
  x and y be distinct λ-calculus identifiers (constants)

  [$\langle$E$\rangle$ / x] x $\longrightarrow$ $\langle$E$\rangle$

A formal definition of β-substitution

- Let $\langle E \rangle$, $\langle F \rangle$ and $\langle G \rangle$ be λ-calculus expressions;
  $\mathtt{x}$ and $\mathtt{y}$ be distinct λ-calculus identifiers (constants)

  $[\langle \mathtt{E} \rangle \ / \ \mathtt{x}] \ \mathtt{x} \rightarrow \langle \mathtt{E} \rangle$
  $[\langle \mathtt{E} \rangle \ / \ \mathtt{x}] \ \mathtt{y} \rightarrow \mathtt{y}$

A formal definition of β-substitution

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

  $[\langle E \rangle \ / \ x] \ x \longrightarrow \langle E \rangle$
  $[\langle E \rangle \ / \ x] \ y \longrightarrow y$
  $[\langle E \rangle \ / \ x] \ (\langle F \rangle \ \langle G \rangle) \longrightarrow (\ [\langle E \rangle \ / \ x] \ \langle F \rangle \quad [\langle E \rangle \ / \ x] \ \langle G \rangle \ )$

A formal definition of β-substitution

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

  $$[\langle E\rangle \ / \ x] \ x \longrightarrow \langle E\rangle$$
  $$[\langle E\rangle \ / \ x] \ y \longrightarrow y$$
  $$[\langle E\rangle \ / \ x] \ (\langle F\rangle \ \langle G\rangle) \longrightarrow (\ [\langle E\rangle \ / \ x] \ \langle F\rangle \quad [\langle E\rangle \ / \ x] \ \langle G\rangle \ )$$
  $$[\langle E\rangle \ / \ x] \ (\lambda x \mid \langle F\rangle) \longrightarrow (\lambda x \mid \langle F\rangle)$$

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

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

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

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

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

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!

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

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

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

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

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

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

$$(\lambda x \mid x) \stackrel{?}{\equiv} (\lambda y \mid y) \qquad \text{YES!}$$
$$(\lambda x \mid x) \stackrel{?}{\equiv} (\lambda x \mid y) \qquad \text{No!}$$
$$(\lambda x \mid (\lambda y \mid x\ y)) \stackrel{?}{\equiv} (\lambda a \mid (\lambda b \mid a\ b))$$

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

$$(\lambda \texttt{x} \mid \texttt{x}) \overset{?}{\equiv} (\lambda \texttt{y} \mid \texttt{y}) \qquad \text{YES!}$$

$$(\lambda \texttt{x} \mid \texttt{x}) \overset{?}{\equiv} (\lambda \texttt{x} \mid \texttt{y}) \qquad \text{No!}$$

$$(\lambda \texttt{x} \mid (\lambda \texttt{y} \mid \texttt{x y})) \overset{?}{\equiv} (\lambda \texttt{a} \mid (\lambda \texttt{b} \mid \texttt{a b})) \qquad \text{YES!}$$

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

$$(\lambda x \mid x) \stackrel{?}{\equiv} (\lambda y \mid y) \qquad \text{YES!}$$
$$(\lambda x \mid x) \stackrel{?}{\equiv} (\lambda x \mid y) \qquad \text{No!}$$
$$(\lambda x \mid (\lambda y \mid x\ y)) \stackrel{?}{\equiv} (\lambda a \mid (\lambda b \mid a\ b)) \qquad \text{YES!}$$
$$(\lambda x \mid (\lambda y \mid x\ y)) \stackrel{?}{\equiv} (\lambda a \mid (\lambda b \mid b\ a))$$

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

$$(\lambda x \mid x) \overset{?}{\equiv} (\lambda y \mid y) \quad \text{YES!}$$
$$(\lambda x \mid x) \overset{?}{\equiv} (\lambda x \mid y) \quad \text{No!}$$
$$(\lambda x \mid (\lambda y \mid x\ y)) \overset{?}{\equiv} (\lambda a \mid (\lambda b \mid a\ b)) \quad \text{YES!}$$
$$(\lambda x \mid (\lambda y \mid x\ y)) \overset{?}{\equiv} (\lambda a \mid (\lambda b \mid b\ a)) \quad \text{NO!}$$

Variables in λ-calculus

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

Variables in λ-calculus

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

Variables in λ-calculus

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

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

Variables in λ-calculus

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

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

Variables in λ-calculus

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

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

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

Variables in λ-calculus

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

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

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

Variables in λ-calculus

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

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

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

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

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 ...
  $(\, (\lambda y \mid (\lambda z \mid \underline{y}z)) \;\; z \,)$
  $\xrightarrow{\beta} [z \,/\, \underline{y}] \;\; (\lambda z \mid \underline{y}z)$
  $\not\equiv (\lambda z \mid zz \,)$ Why not?

α (Alpha Rule): Motivation

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

α (Alpha Rule): Motivation

- The β-rule cannot be applied in ...
  $(\ (\lambda y \mid (\lambda z \mid yz)) \ \ z \ )$
  $\xrightarrow{\beta} [z \ / \ y] \ \ (\lambda z \mid yz)$
  $\neq (\lambda z \mid zz \ )$ Why not?
  z was free in z but *bound* in the result $\Rightarrow$ substitution is illegal!

-

  $(\lambda y \ \ (\lambda z | yz)) \ \ (\lambda x | xz)$
  $\xrightarrow{\beta} [(\lambda x | xz) / \ y] \ \ (\lambda z | yz)$
  $\neq \ \ (\lambda z | (\lambda x | xz) z)$

α (Alpha Rule): Motivation

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

-

  $(\lambda y\ \ (\lambda z | yz))\ \ (\lambda x | xz)$
  $\xrightarrow{\beta} [(\lambda x | xz) /\ y]\ \ (\lambda z | yz)$
  $\not\equiv\ \ (\lambda z | (\lambda x | xz) z)$

- But, variable identifiers in and of themselves are irrelevant

α (Alpha Rule): Motivation

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

  $(\lambda y \;\; (\lambda z | yz)) \;\; (\lambda x | xz)$
  $\xrightarrow{\beta} [(\lambda x | xz) / \;\; y] \;\; (\lambda z | yz)$
  $\not\equiv \;\; (\lambda z | (\lambda x | xz) z)$
- 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

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

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

$$(\lambda \mathtt{z} \,|\, \mathtt{yz}) \overset{\alpha:q/z}{\longrightarrow} (\lambda \mathtt{q} \,|\, \mathtt{yq})$$

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

$$(\lambda \texttt{z} \,|\, \texttt{yz}) \overset{\alpha:q/z}{\longrightarrow} (\lambda \texttt{q} \,|\, \texttt{yq})$$

  - Note: Replace the formal parameter *and* EVERY instance of `z` with `q`

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

  $$(\lambda z \mid yz) \xrightarrow{\alpha:q/z} (\lambda q \mid yq)$$

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

α (Alpha Rule): Examples

$$(\lambda \mathtt{a} \mid \mathtt{b}\ (\lambda \mathtt{c} \mid \mathtt{c}\ \mathtt{a})\ \mathtt{d}) \stackrel{\alpha:z/a}{\longrightarrow} (\lambda \mathtt{z} \mid \mathtt{b}\ (\lambda \mathtt{c} \mid \mathtt{c}\ \mathtt{z})\ \mathtt{d})$$

α (Alpha Rule): Examples

$(\lambda a \mid b\ (\lambda c \mid c\ a)\ d) \stackrel{\alpha:z/a}{\longrightarrow} (\lambda z \mid b\ (\lambda c \mid c\ z)\ d)$

Legal

α (Alpha Rule): Examples

$(\lambda a \mid b\ (\lambda c \mid c\ a)\ d) \stackrel{\alpha: z/a}{\longrightarrow} (\lambda z \mid b\ (\lambda c \mid c\ z)\ d)$

Legal

$(\lambda x \mid (\lambda y \mid x\ y\ z)\ ) \stackrel{\alpha: y/z}{\longrightarrow} (\lambda x \mid (\lambda y \mid x\ y\ y)\ )$

α (Alpha Rule): Examples

$(\lambda \texttt{a} \mid \texttt{b} \ (\lambda \texttt{c} \mid \texttt{c} \ \texttt{a}) \ \texttt{d}) \stackrel{\alpha:z/a}{\longrightarrow} (\lambda \texttt{z} \mid \texttt{b} \ (\lambda \texttt{c} \mid \texttt{c} \ \texttt{z}) \ \texttt{d})$
Legal

$(\lambda \texttt{x} \mid (\lambda \texttt{y} \mid \texttt{x} \ \texttt{y} \ \texttt{z}) \ ) \stackrel{\alpha:y/z}{\longrightarrow} (\lambda \texttt{x} \mid (\lambda \texttt{y} \mid \texttt{x} \ \texttt{y} \ \texttt{y}) \ )$
Illegal

Formal definition of α

- Let $\langle$E$\rangle$ and$\langle$F$\rangle$ be λ-calculus expressions;
  x and y be distinct λ-calculus constants

Formal definition of α

- Let $\langle E \rangle$ and $\langle F \rangle$ be λ-calculus expressions; x and y be distinct λ-calculus constants
- Let z be a newly generated λ calculus constant

$$[\langle \mathtt{E} \rangle \ / \ \mathtt{x}] \ (\lambda \mathtt{y} \ | \ \langle \mathtt{F} \rangle) \rightarrow (\lambda \mathtt{z} \ | \ [\langle \mathtt{E} \rangle \ / \mathtt{x}] \ [\mathtt{z}/\mathtt{y}] \ \langle \mathtt{F} \rangle \ )$$

Using α and β Together I

(λy (λz|yz)) (λx|xz)

Using α and β Together I

$(\lambda y\ (\lambda z|yz))\ (\lambda x|xz)$

- We could use β rule to simulate applying the function

$\xrightarrow{\beta}[\ (\lambda x|xz)/\ y\ ]\ (\lambda z|yz)$

Using α and β Together I

$(\lambda\mathtt{y}\ (\lambda\mathtt{z}|\mathtt{yz}))\ (\lambda\mathtt{x}|\mathtt{xz})$

- We could use β rule to simulate applying the function

$\xrightarrow{\beta}[\ (\lambda\mathtt{x}|\mathtt{xz})/\ \mathtt{y}\ ]\ (\lambda\mathtt{z}|\mathtt{yz})$

- Legal substitution?

Using α and β Together I

$(\lambda y\ (\lambda z|yz))\ (\lambda x|xz)$

- We could use β rule to simulate applying the function

$\xrightarrow{\beta}[\ (\lambda x|xz)/\ y\ ]\ (\lambda z|yz)$

- Legal substitution? No. Free z in $(\lambda x|xz)$ becomes bound

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)

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)

Using α and β Together II

- Now we can apply β-substitution

  [(λx|xz)/ y] (λq|yq)$\equiv$ (λq| (λx|xz) q)

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)

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)

Using α and β Together II

- Now we can apply β-substitution

$$[\ (\lambda x|xz)/\ y\ ]\ (\lambda q|yq) \equiv (\lambda q|\ (\lambda x|xz)\ q)$$

$$(\lambda q|\ (\lambda x|xz)\ q)$$
$$\xrightarrow{\beta}\ (\lambda q|\ [q\ /x\ ]\ xz)\ \equiv (\lambda q|\ q\ z)$$

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!

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

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

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: $(\lambda x \,|\langle \mathrm{E}\rangle)\, v \;\stackrel{\beta}{\rightarrow}\; [v/x]\,\langle \mathrm{E}\rangle$

η (Eta Rule): Null Application

- Special case of the β-rule: $(\lambda x \,|\langle \mathsf{E} \rangle)\, v \ \overset{\beta}{\rightarrow}\ [v/x]\,\langle \mathsf{E} \rangle$
- Accelerates Rule 5 of β substitution

η (Eta Rule): Null Application

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

η (Eta Rule): Null Application

- Special case of the β-rule: $(\lambda x \,|\, \langle E \rangle)\, v \;\xrightarrow{\beta}\; [v/x]\, \langle E \rangle$
- 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 $((\lambda \mathtt{x} \;|\; \langle \mathtt{E} \rangle)\;\mathtt{v}) \xrightarrow{\eta} \langle E \rangle$

η (Eta Rule): Null Application

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

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

η (Eta Rule): Null Application

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

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

η (Eta Rule): Null Application

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

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

η (Eta Rule): Null Application

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

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

λ-calculus Interpreters

- To implement a λ-calculus interpreter

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

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 `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?

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)

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 $\langle$N$\rangle$ from $\langle$M$\rangle$ using a squence of β and η operations,
  then $\langle$M$\rangle$ is *reducible* to $\langle$N$\rangle$

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 $\langle \mathrm{N} \rangle$ from $\langle \mathrm{M} \rangle$ using a squence of β and η operations,
  then $\langle \mathrm{M} \rangle$ is *reducible* to $\langle \mathrm{N} \rangle$
- An expression that can be reduced is called a *redux*

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 $\langle$N$\rangle$ from $\langle$M$\rangle$ using a squence of β and η operations,
  then $\langle$M$\rangle$ is *reducible* to $\langle$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)`

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 ⟨N⟩ from ⟨M⟩ using a squence of β and η operations,
  then ⟨M⟩ is *reducible* to ⟨N⟩
- 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” $\equiv$”reducing to normal form”

Theoretical Questions

- So "interpretation" $\equiv$"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?

Theoretical Questions

- So "interpretation" $\equiv$"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

Order of Reductions: Normal I

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

$(\lambda x | (\lambda y\ | x))\ ((\lambda u\ |\ z)\ u\ )$

Order of Reductions: Normal I

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

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

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

Order of Reductions: Normal I

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

```
(λx|(λy |x)) ((λu | z) u )
(λx|(λy|x)) ((λu | z) u )
 leftmost
(λx|(λy |x)) ((λu | z) u )
```

Order of Reductions: Normal I

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

```
(λx|(λy |x)) ((λu | z) u )
(λx|(λy|x)) ((λu | z) u )
 leftmost
(λx|(λy |x)) ((λu | z) u )
```

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

Order of Reductions: Normal I

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

(λx|(λy |x)) ((λu | z) u)

$\underbrace{\texttt{(}\lambda\texttt{x|(}\lambda\texttt{y|x))}}_{\texttt{leftmost}}$ ((λu | z) u)

(λx|(λy |x)) ((λu | z) u)

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

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

Order of Reductions: Normal I

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

$(\lambda x|(\lambda y\ |x))\ ((\lambda u\ |\ z)\ u\ )$

$\underbrace{(\lambda x|(\lambda y|x))}_{\text{leftmost}}\ ((\lambda u\ |\ z)\ u\ )$

$(\lambda \underline{x}|(\lambda y\ |\underline{x}))\ ((\lambda u\ |\ z)\ u\ )$

$\xrightarrow{\beta}[((\lambda u\ |\ z)\ u\ )\ /\ \underline{x}]\quad (\lambda y\ |\underline{x})$

Free vars in $((\lambda u\ |\ z)\ u\ )$ get bound? No

$\equiv$

Order of Reductions: Normal I

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

$(\lambda x | (\lambda y \ | x)) \ ((\lambda u \ | \ z) \ u \ )$

$\underbrace{(\lambda x | (\lambda y | x))}_{\text{leftmost}} \ ((\lambda u \ | \ z) \ u \ )$

$(\lambda \underline{x} | (\lambda y \ | \underline{x})) \ ((\lambda u \ | \ z) \ u \ )$

$\xrightarrow{\beta} [((\lambda u \ | \ z) \ u \ ) \ / \ \underline{x}] \quad (\lambda y \ | \underline{x})$

Free vars in $((\lambda u \ | \ z) \ u \ )$ get bound? No

$\equiv (\lambda y \ | \quad ((\lambda u \ | \ z) \ u \ ))$

Order of Reductions: Normal II

$(\lambda y \mid ((\lambda u \mid z)\ u\ ))$ Left application?

Order of Reductions: Normal II

$(\lambda \texttt{y} \mid ((\lambda \texttt{u} \mid \texttt{z})\ \texttt{u}))$ Left application?

$(\lambda \texttt{y} \mid \underbrace{((\lambda \texttt{u}|\texttt{z})\ \texttt{u})}_{\texttt{leftmost}})$

Order of Reductions: Normal II

$(\lambda y \mid ((\lambda u \mid z)\ u))$ Left application?

$(\lambda y \mid \underbrace{((\lambda u|z)\ u)}_{\texttt{leftmost}})$

$(\lambda y \mid ((\lambda \underline{u} \mid z)\ u\ )\ )$

Order of Reductions: Normal II

$(\lambda y \mid ((\lambda u \mid z)\ u\ ))$ Left application?

$(\lambda y \mid \underbrace{((\lambda u|z)\ u)}_{\texttt{leftmost}})$

$(\lambda y \mid ((\lambda \underline{u} \mid z)\ u\ )\ )$

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

Order of Reductions: Normal II

(λy | ((λu | z) u)) Left application?

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

(λy | ((λu | z) u))

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

Any free vars in u get bound? No.

Order of Reductions: Normal II

$(\lambda y \mid ((\lambda u \mid z)\ u\ ))$ Left application?

$(\lambda y \mid \underbrace{((\lambda u|z)\ u)}_{\texttt{leftmost}})$

$(\lambda y \mid ((\lambda \underline{u} \mid z)\ u\ )\ )$

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

Any free vars in u get bound? No.

$\rightarrow$

Order of Reductions: Normal II

$(\lambda y \mid ((\lambda u \mid z)\ u\ ))$ Left application?

$(\lambda y \mid \underbrace{((\lambda u|z)\ u)}_{\texttt{leftmost}})$

$(\lambda y \mid ((\lambda \underline{u} \mid z)\ u\ )\ )$

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

Any free vars in u get bound? No.

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

Order of Reductions: Normal II

$(\lambda y \mid ((\lambda u \mid z)\ u\ ))$ Left application?

$(\lambda y \mid \underbrace{((\lambda u|z)\ u)}_{\text{leftmost}})$

$(\lambda y \mid ((\lambda \underline{u} \mid z)\ u\ )\ )$

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

Any free vars in u get bound? No.

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

Done?

Order of Reductions: Normal II

$(\lambda y \mid ((\lambda u \mid z)\ u))$ Left application?

$(\lambda y \mid \underbrace{((\lambda u|z)\ u)}_{\texttt{leftmost}})$

$(\lambda y \mid ((\lambda \underline{u} \mid z)\ u\ )\ )$

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

Any free vars in u get bound? No.

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

Done? Yes - normal form

Order of Reductions: Applicative I

- Applicative Order: innermost *application* first

Order of Reductions: Applicative I

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

Order of Reductions: Applicative I

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

$(\lambda x|(\lambda y\ |x))\ ((\lambda u\ |\ z)\ u\ )$

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)
```

$$\underbrace{(\lambda u|z)}_{\texttt{innermost}}$$

Order of Reductions: Applicative I

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

$$(\lambda x | (\lambda y \; | x)) \; ((\lambda u \; | \; z) \; u \;)$$

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

$$(\lambda x | (\lambda y \; | x)) \; ((\lambda \underline{u} \; | \; z) \; u \;)$$

Order of Reductions: Applicative I

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

$$(\lambda x|(\lambda y\ |x))\ ((\lambda u\ |\ z)\ u\ )$$

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

$$(\lambda x|(\lambda y\ |x))\ ((\lambda \underline{u}\ |\ z)\ u\ )$$

$$\stackrel{\beta}{\rightarrow}\ (\lambda x|(\lambda y\ |x))\ [u\ /u\ ]\ (\lambda \underline{u}\ |\ z)$$

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?

Order of Reductions: Applicative I

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

$(\lambda x | (\lambda y\ | x))\ ((\lambda u\ |\ z)\ u\ )$

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

$(\lambda x | (\lambda y\ | x))\ ((\lambda \underline{u}\ |\ z)\ u\ )$

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

any free vars in u get bound?No.

$\equiv$

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.

$\equiv$ (λx|(λy |x)) z

Order of Reductions: Applicative I

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

$(\lambda x|(\lambda y\ |x))\ ((\lambda u\ |\ z)\ u\ )$

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

$(\lambda x|(\lambda y\ |x))\ ((\lambda \underline{u}\ |\ z)\ u\ )$

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

any free vars in u get bound?No.

$\equiv\ (\lambda x|(\lambda y\ |x))\ z$

Done?

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.

$\equiv$ (λx|(λy |x)) z

Done? Nope

Order of Reductions: Applicative II

`(λx|(λy |x)) z` Innermost?

Order of Reductions: Applicative II

$(\lambda x | (\lambda y \ | x)) \ z$ Innermost?

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

Order of Reductions: Applicative II

$(\lambda x|(\lambda y\ |x))\ z$ Innermost?

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

$(\lambda \underline{x}|(\lambda y\ |\underline{x}))\ z$

Order of Reductions: Applicative II

$(\lambda x|(\lambda y\ |x))\ z$ Innermost?

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

$(\lambda \underline{x}|(\lambda y\ |\underline{x}))\ z$

$\xrightarrow{\beta}[z\ /\ \underline{x}]\quad(\lambda y\ |\underline{x})$

Order of Reductions: Applicative II

$(\lambda x | (\lambda y \ | x))$ z   Innermost?

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

$(\lambda \underline{x} | (\lambda y \ | \underline{x}))$ z

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

Any free vars in x get bound? No

Order of Reductions: Applicative II

$(\lambda \mathtt{x} | (\lambda \mathtt{y}\ | \mathtt{x}))\ \mathtt{z}$ Innermost?

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

$(\lambda \underline{\mathtt{x}} | (\lambda \mathtt{y}\ | \underline{\mathtt{x}}))\ \mathtt{z}$

$\xrightarrow{\beta} [\mathtt{z}\ /\ \underline{\mathtt{x}}]\quad (\lambda \mathtt{y}\ | \underline{\mathtt{x}})$

Any free vars in $\mathtt{x}$ get bound? No

$\equiv$

Order of Reductions: Applicative II

$(\lambda x | (\lambda y \ | x))\ z$ Innermost?

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

$(\lambda \underline{x} | (\lambda y \ | \underline{x}))\ z$

$\xrightarrow{\beta} [z \ / \ \underline{x}] \quad (\lambda y \ | \underline{x})$

Any free vars in x get bound? No

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

Order of Reductions: Applicative II

$(\lambda x|(\lambda y\ |x))\ z$ Innermost?

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

$(\lambda \underline{x}|(\lambda y\ |\underline{x}))\ z$

$\xrightarrow{\beta}[z\ /\ \underline{x}]\quad(\lambda y\ |\underline{x})$

Any free vars in x get bound? No

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

Done?

Order of Reductions: Applicative II

$(\lambda x | (\lambda y \; | x)) \; z$ Innermost?

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

$(\lambda \underline{x} | (\lambda y \; | \underline{x})) \; z$

$\xrightarrow{\beta} [z \; / \; \underline{x}] \quad (\lambda y \; | \underline{x})$

Any free vars in x get bound? No

$\equiv \; (\lambda y \; | \; z)$

Done? Yes - normal form

Order of Reductions: Comment

- You may choose
  - normative (left-most legal application) or
  - applicative order (innermost legal application) or
  - ...

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

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 $\langle \mathrm{A} \rangle, \langle \mathrm{B} \rangle, \langle \mathrm{C} \rangle, \langle \mathrm{D} \rangle$ be λ-calculus expressions and $\overset{1}{\rightarrow}$, $\overset{2}{\rightarrow}$, $\overset{3}{\rightarrow}$ and $\overset{4}{\rightarrow}$ be reductions of *zero* or more steps

Church and Rosser Theorem

- Let $\langle A \rangle, \langle B \rangle, \langle C \rangle, \langle D \rangle$ 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 $\exists$ $\langle D \rangle$ $\xrightarrow{3}$ and $\xrightarrow{4}$ s.t. $\langle B \rangle \xrightarrow{3} \langle D \rangle$ and $\langle C \rangle \xrightarrow{4} \langle D \rangle$

Church and Rosser Theorem

- Let $\langle A \rangle, \langle B \rangle, \langle C \rangle, \langle D \rangle$ 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 $\exists$ $\langle D \rangle$ $\xrightarrow{3}$ 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 $\langle A \rangle$ can always be reduced to the same expression (function)

Uniqueness Corollary

- Corollary: Given two reductions $\langle A \rangle \xrightarrow{1} \langle B \rangle$ and $\langle A \rangle \xrightarrow{2} \langle C \rangle$

Uniqueness Corollary

- Corollary: Given two reductions $\langle A \rangle \xrightarrow{1} \langle B \rangle$ and $\langle A \rangle \xrightarrow{2} \langle C \rangle$
  - If $\langle B \rangle$ and $\langle C \rangle$ are in normal form, neither can be reduced further

Uniqueness Corollary

- Corollary: Given two reductions $\langle A \rangle \xrightarrow{1} \langle B \rangle$ and $\langle A \rangle \xrightarrow{2} \langle C \rangle$
  - If $\langle B \rangle$ and $\langle C \rangle$ are in normal form, neither can be reduced further
  - By Church and Rosser I, we can reduce $\langle C \rangle$ and $\langle B \rangle$ to an identical form in *zero* or more steps

Uniqueness Corollary

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

Uniqueness Corollary

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

Uniqueness Corollary

- Corollary: Given two reductions $\langle A \rangle \xrightarrow{1} \langle B \rangle$ and $\langle A \rangle \xrightarrow{2} \langle C \rangle$
  - If $\langle B \rangle$ and $\langle C \rangle$ are in normal form, neither can be reduced further
  - By Church and Rosser I, we can reduce $\langle C \rangle$ and $\langle B \rangle$ to an identical form in *zero* or more steps
  - Since both $\langle C \rangle$ and $\langle B \rangle$ are irreducible, the required reduction must be of length zero and $\langle C \rangle$ and $\langle B \rangle$ 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 $\langle A \rangle \rightarrow \langle B \rangle$ and $\langle B \rangle$ is in normal form then $\langle A \rangle \rightarrow \langle B \rangle$ by *normative order* reduction

Existence Theorem

- Church and Rosser Theorem II
  - If $\langle A \rangle \rightarrow \langle B \rangle$ and $\langle B \rangle$ is in normal form
    then $\langle A \rangle \rightarrow \langle B \rangle$ by *normative order* reduction
- If $\langle A \rangle$ can be reduced to a normal form,
  it can be found by normal order reduction

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

  (λx| x x) (λ x| x x)

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

```
(λx| x x) (λ x| x x)
(λx| x x) (λ x| x x)
→
```

Existence Theorem

- Church and Rosser Theorem II
  - If $\langle A\rangle \longrightarrow \langle B\rangle$ and $\langle B\rangle$ is in normal form then $\langle A\rangle \longrightarrow \langle B\rangle$ by *normative order* reduction
- If $\langle A\rangle$ can be reduced to a normal form, it can be found by normal order reduction
- Not every expression has a normal form

$(\lambda x|\ x\ x)\ (\lambda\ x|\ x\ x)$

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

$\rightarrow\ (\lambda x|\ x\ x)\ (\lambda\ x|\ x\ x)$

Existence Theorem

- Church and Rosser Theorem II
  - If $\langle A \rangle \longrightarrow \langle B \rangle$ and $\langle B \rangle$ is in normal form
    then $\langle A \rangle \longrightarrow \langle B \rangle$ by *normative order* reduction
- If $\langle A \rangle$ can be reduced to a normal form,
  it can be found by normal order reduction
- Not every expression has a normal form

  $(\lambda \mathtt{x} \mid \mathtt{x}\ \mathtt{x})\ (\lambda\ \mathtt{x} \mid \mathtt{x}\ \mathtt{x})$
  $(\lambda \underline{\mathtt{x}} \mid \underline{\mathtt{x}}\ \underline{\mathtt{x}})\ (\lambda\ \mathtt{x} \mid \mathtt{x}\ \mathtt{x})$
  $\rightarrow\ (\lambda \mathtt{x} \mid \mathtt{x}\ \mathtt{x})\ (\lambda\ \mathtt{x} \mid \mathtt{x}\ \mathtt{x})$

- Because reductions are not guaranteed to terminate,
  the equivalence of λ-calculus expressions is undecidable

Existence Theorem

- Church and Rosser Theorem II
  - If $\langle A \rangle \rightarrow \langle B \rangle$ and $\langle B \rangle$ is in normal form then $\langle A \rangle \rightarrow \langle B \rangle$ by *normative order* reduction
- If $\langle A \rangle$ can be reduced to a normal form, it can be found by normal order reduction
- Not every expression has a normal form

  (λx| x x) (λ x| x x)
  (λx| x x) (λ x| x x)
  → (λx| x x) (λ 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

Reduction Orders as Parameter Types

- Applicative order reduction evaluates innermost applications first
  - $\approx$ evaluating arguments before passing them

Reduction Orders as Parameter Types

- Applicative order reduction evaluates innermost applications first
  - $\approx$ evaluating arguments before passing them
  - Can be interpreted as "call by value"

Reduction Orders as Parameter Types

- Applicative order reduction evaluates innermost applications first
  - $\approx$ evaluating arguments before passing them
  - Can be interpreted as "call by value"
- Normative order reduction evaluates leftmost applications first

Reduction Orders as Parameter Types

- Applicative order reduction evaluates innermost applications first
  - $\approx$ evaluating arguments before passing them
  - Can be interpreted as "call by value"
- Normative order reduction evaluates leftmost applications first
  - $\approx$passing unevaluated expressions to function

Reduction Orders as Parameter Types

- Applicative order reduction evaluates innermost applications first
  - $\approx$ evaluating arguments before passing them
  - Can be interpreted as "call by value"
- Normative order reduction evaluates leftmost applications first
  - $\approx$passing unevaluated expressions to function
  - Can be interpreted as "call by name"

Reduction Orders as Parameter Types

- Applicative order reduction evaluates innermost applications first
  - $\approx$ evaluating arguments before passing them
  - Can be interpreted as "call by value"
- Normative order reduction evaluates leftmost applications first
  - $\approx$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
  - $((\lambda x \mid y) \langle E \rangle) \rightarrow y$ for any $\langle E \rangle$
  - Here, expression $\langle E \rangle$ is an *unneeded* argument
  - η-reductions
- Applicative order may evaluate *unneeded* arguments

Completeness of Applicative vs. Normal Order

- The argument to $(\lambda x \mid y)$ does not matter
  - $((\lambda x \mid y)\langle E\rangle) \rightarrow y$ 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

Completeness of Applicative vs. Normal Order

- The argument to $(\lambda x \mid y)$ does not matter
  - $((\lambda x \mid y)\langle E\rangle) \rightarrow y$ 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

Completeness of Applicative vs. Normal Order

- The argument to $(\lambda x \mid y)$ does not matter
  - $((\lambda x \mid y)\langle E \rangle) \rightarrow y$ 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

Completeness of Applicative vs. Normal Order

- The argument to $(\lambda x \mid y)$ does not matter
  - $((\lambda x \mid y)\langle E\rangle) \rightarrow y$ 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

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

Reducible by Normal Example

(λz (λy| y)) ((λx | x x) (λx | x x))

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

Reducible by Normal Example

```
(λz (λy| y)) ( (λx | x x) (λx | x x) )
   (λz (λy | y))      ( (λx | x x) (λx | x x) )
```
$\underbrace{\texttt{(}\lambda\texttt{z (}\lambda\texttt{y | y))}}$

```
leftmost application
(λz (λy| y)) ( (λx | x x) (λx | x x) )
```

Reducible by Normal Example

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

$\underbrace{(\lambda z\ (\lambda y\ |\ y))}\quad (\ (\lambda x\ |\ x\ x)\ (\lambda x\ |\ x\ x)\ )$

leftmost application

$(\lambda \underline{z}\ (\lambda y |\ y))\ (\ (\lambda x\ |\ x\ x)\ (\lambda x\ |\ x\ x)\ )$

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

Reducible by Normal Example

(λz (λy| y)) ((λx | x x) (λx | x x))

$\underbrace{\texttt{(}\lambda\texttt{z (}\lambda\texttt{y | y))}}_{\texttt{leftmost application}}$ ((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?

Reducible by Normal Example

(λz (λy| y)) ((λx | x x) (λx | x x))

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

(λ<u>z</u> (λy| y)) ((λx | x x) (λx | x x))

$\stackrel{\beta}{\rightarrow}$ [((λx | x x) (λx | x x))/ <u>z</u>] (λ<u>z</u> (λy| y))

Any free vars get bound? No.

$\equiv$

Reducible by Normal Example

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

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

$(\lambda \underline{z}\ (\lambda y|\ y))\ (\ (\lambda x\ |\ x\ x)\ (\lambda x\ |\ x\ x)\ )$

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

Any free vars get bound? No.

$\equiv\ (\lambda y|\ y)$

Irreducible by Applicative Example

- Under Applicative order

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

Irreducible by Applicative Example

- Under Applicative order

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

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

Irreducible by Applicative Example

- Under Applicative order

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

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

$$(\lambda z\ (\lambda y|\ y))\ (\ (\lambda \underline{x}\ |\ \underline{x}\ \underline{x})\ (\lambda x\ |\ x\ x)\ )$$

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) )
-β->(λz (λy| y))  [ (λx | x x) / x  ] x x
```

Irreducible by Applicative Example

- Under Applicative order

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

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

$(\lambda z\ (\lambda y|\ y))\ (\ (\lambda \underline{x}\ |\ \underline{x}\ \underline{x})\ (\lambda x\ |\ x\ x)\ )$

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

Any free vars in $(\lambda x\ |\ x\ x)$ get bound?

Irreducible by Applicative Example

- Under Applicative order

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

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

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

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

Any free vars in $(\lambda x \mid x\ x)$ get bound? No.

$\equiv$

Irreducible by Applicative Example

- Under Applicative order

(λz (λy| y)) ((λx | x x) (λx | x x))

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

(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))

Irreducible by Applicative Example

- Under Applicative order

(λz (λy| y)) ((λx | x x) (λx | x x))
(λz (λy| y)) ($\underbrace{\text{(λx | x x)}}_{\text{innermost application}}$ (λx | x x))
(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?

Irreducible by Applicative Example

- Under Applicative order

(λz (λy| y)) ((λx | x x) (λx | x x))

(λz (λy| y)) ($\underbrace{\text{(λx | x x)}}_{\text{innermost application}}$ (λx | x x))

(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)

Example 1 : Normal Order

`(λx | ( λy x | x) z)) (λx | x y)`

First step?

Example 1 : Normal Order

`(λx | ( λy x | x) z)) (λx | x y)`

First step? Identify *leftmost* applicable function

Example 1 : Normal Order

```
(λx | ( λy x | x) z)) (λx | x y)
```

First step? Identify *leftmost* applicable function

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

Example 1 : Normal Order

(λx | (λy x | x) z)) (λx | x y)

First step? Identify *leftmost* applicable function

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

(λx | (λy x | x) z)) (λx | x y)

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)

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

(λx | (λy x | x) z)) (λx | x y)

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

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)

Example 1 : Normal Order

(λx | (λy x | x) z)) (λx | x y)

First step? Identify *leftmost* applicable function

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

(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)

Example 1 : Normal Order

(λx | (λy x | x) z)) (λx | x y)

First step? Identify *leftmost* applicable function

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

(λx | (λy x | x) z)) (λx | x y)

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

(λx | (λy | (λx | x) z))) (λx | x y)

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

Free vars in (λx | x y)? get bound?

Example 1 : Normal Order

(λx | (λy x | x) z)) (λx | x y)

First step? Identify *leftmost* applicable function

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

(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)

Example 1 : Normal Order

(λx | (λy x | x) z)) (λx | x y)

First step? Identify *leftmost* applicable function

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

(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)

Example 1 : Normal Order

(λx | (λy x | x) z)) (λx | x y)

First step? Identify *leftmost* applicable function

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

(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)

Example 1 : Normal Order

(λx | (λy x | x) z)) (λx | x y)

First step? Identify *leftmost* applicable function

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

(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

```
(λx | ( λy x | x) z) (λx | x y)
```

Example 1 : Applicative

(λx | (λy x | x) z) (λx | x y)

First step?

Example 1 : Applicative

(λx | (λy x | x) z) (λx | x y)

First step? Identify *innermost* applicable function

Example 1 : Applicative

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

First step? Identify *innermost* applicable function

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

Example 1 : Applicative

(λx | (λy x | x) z) (λx | x y)

First step? Identify *innermost* applicable function

(λx | (λy x| x) z) (λx | x y)

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

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

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

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

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

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

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 $(\lambda x \mid x)$

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 $(\lambda x \mid x)$

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

Example 1 : Applicative

$(\lambda$x | (λy x | x) z) (λx | x y)

First step? Identify *innermost* applicable function

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

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

$(\lambda$x | (λy (λx | x)) z) (λx | x y)

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

No free instances of y in (λx | x)

$\equiv$ (λx | (λx | x)) (λx | x y)

$(\lambda$x | (λx | x)) (λx | x y)

Example 1 : Applicative

(λx | (λy x | x) z) (λx | 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: (λy x | x) means (λy | (λx | x))

(λx | (λy (λx | x)) z) (λx | x y)

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

No free instances of y in (λx | x)

≡ (λx | (λx | x)) (λx | x y)

(λx | (λx | x)) (λx | x y)

$\xrightarrow{\eta}$ (λx | x)

Example 2: Normal Order I

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

Example 2: Normal Order I

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

First task: find leftmost applicable function

Example 2: Normal Order I

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

First task: find leftmost applicable function

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

Example 2: Normal Order I

(λx | (λy | x)) ((λx | y) x)

First task: find leftmost applicable function

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

(λx | (λy | x)) ((λx | y) x)

Example 2: Normal Order I

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

First task: find leftmost applicable function

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

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

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

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?

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!

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

Example 2: Normal Order I

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

First task: find leftmost applicable function

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

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

$\stackrel{\beta}{\rightarrow} [((\lambda x \mid y)\ x) \,/\, x]\ (\lambda y \mid \underline{x})$

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

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

Use α rule.

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.

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

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!

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

Use α rule.

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

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

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

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

Example 2: Normal Order II

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

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

Example 2: Normal Order II

```
(λz | (λx | y) x )
(λz |  (λx|y)  x)
       leftmost
(λz | ((λx | y) x))
```

Example 2: Normal Order II

$$
\begin{array}{l}
(\lambda z \mid (\lambda x \mid y)\ x\ ) \\
(\lambda z \mid \underbrace{(\lambda x|y)}_{\texttt{leftmost}}\ x) \\
(\lambda z \mid ((\lambda {\color{red}x} \mid y)\ {\color{red}x})) \\
\xrightarrow{\eta}\ (\lambda z \mid y\ )
\end{array}
$$

Example 2: Applicative I

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

Example 2: Applicative I

$(\lambda$x | $(\lambda$y | x)) ((λx | y) x)

First step?

Example 2: Applicative I

`(λx | (λy | x)) ((λx | y) x)`

First step? `Find innermost application`

Example 2: Applicative I

$(\lambda$x | $(\lambda$y | x)) $((\lambda$x | y) x)

First step? `Find innermost application`

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

Example 2: Applicative I

```
(λx | (λy | x)) ((λx | y) x)
```

First step? `Find innermost application`

```
(λx | (λy | x)) ((λx | y) x)
                 \_________/
                  innermost
(λx | (λy | x)) ((λx | y) x)
```

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

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

Example 2: Applicative II

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

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

Example 2: Applicative II

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

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

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

Example 2: Applicative II

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

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

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

Free vars get bound?

Example 2: Applicative II

$$\texttt{(}\lambda\texttt{x | (}\lambda\texttt{y | x)) y}$$

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

$$\stackrel{\beta}{\longrightarrow}\texttt{[y/x] (}\lambda\texttt{y | x)}$$

Free vars get bound? `Yes`

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

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

Example 2: Applicative II

$$\texttt{(}\lambda\texttt{x | (}\lambda\texttt{y | x)) y}$$

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

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

Free vars get bound? `Yes`

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

$$\equiv\texttt{[y/x](}\lambda\texttt{z | x)}$$

Example 2: Applicative II

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

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

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

Free vars get bound? Yes

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

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

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

Example 3: Normal I

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

Example 3: Normal I

((λx y | y) ((λx | x x) (λx | x x))) a

First step?

Example 3: Normal I

((λx y | y) ((λx | x x) (λx | x x))) a

First step? `Find leftmost application`

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`

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

Example 3: Normal I

((λx y | y) ((λx | x x) (λx | x x))) a

First step? Find leftmost application

$\underbrace{\texttt{(λx y | y)}}_{\texttt{leftmost}}$ ((λx | x x) (λx | x x)) a Re-

call: (λx y | y) ≡ (λx | (λy | y))

Example 3: Normal I

((λx y | y) ((λx | x x) (λx | x x))) a

First step? **Find leftmost application**

(λx y | y) ((λx | x x) (λx | x x)) a   Re-

leftmost

call: (λx y | y) ≡ (λx | (λy | y))

((λx |(λy | y)) ((λx | x x) (λx | x x))) a

leftmost

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**

$\underbrace{(\lambda x\ y \mid y)}_{\texttt{leftmost}}\ ((\lambda x \mid x\ x)\ (\lambda x \mid x\ x))\ a$ Recall: $(\lambda x\ y \mid y) \equiv (\lambda x \mid (\lambda y \mid y))$

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

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

Example 3: Normal I

$((\lambda\texttt{x y | y}) \; ((\lambda\texttt{x | x x}) \; (\lambda\texttt{x | x x}))) \; \texttt{a}$

First step? **Find leftmost application**

$\underbrace{(\lambda\texttt{x y | y})}_{\texttt{leftmost}} \; ((\lambda\texttt{x | x x}) \; (\lambda\texttt{x | x x})) \; \texttt{a}$ Re-call: $(\lambda\texttt{x y | y}) \equiv (\lambda\texttt{x | }(\lambda\texttt{y | y}))$

$(\underbrace{(\lambda\texttt{x |}(\lambda\texttt{y | y}))}_{\texttt{leftmost}} \; ((\lambda\texttt{x | x x}) \; (\lambda\texttt{x | x x}))) \; \texttt{a}$

$\xrightarrow{\eta} (\lambda\texttt{y | y}) \; \texttt{a}$

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

Example 3: Applicative I

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

Example 3: Applicative I

`(λx y | y) ((λx | x x) (λx | x x)) a`

First step? Find innermost application.

Example 3: Applicative I

`(λx y | y) ((λx | x x) (λx | x x)) a`

First step? Find innermost application.

$\texttt{(}\lambda\texttt{x y | y)} \quad \texttt{(}\underbrace{\texttt{(}\lambda\texttt{x | x x)}}_{\texttt{innermost}}\ \texttt{(}\lambda\texttt{x | x x)) a}$

Example 3: Applicative I

(λx y | y) ((λx | x x) (λx | x x)) a

First step? Find innermost application.

(λx y | y) ((λx | x x) (λx | x x)) a

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

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

Example 3: Applicative I

(λx y | y) ((λx | x x) (λx | x x)) a

First step? Find innermost application.

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

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

Will free vars in get (λx | x x) bound?

Example 3: Applicative I

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

First step? Find innermost application.

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

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

Will free vars in get $(\lambda x \mid x\ x)$ bound? No free vars!

Example 3: Applicative I

(λx y | y) ((λx | x x) (λx | x x)) a

First step? Find innermost application.

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

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

Will free vars in get (λx | x x) bound? No free vars!

≡ (λx y | y) ((λx | x x)(λx | x x)) a

Example 3: Applicative I

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

First step? Find innermost application.

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

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

Will free vars in get $(\lambda x \mid x\ x)$ bound? No free vars!

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

We get the original expression back again!

Shortcuts for Multi-argument λ's

$$(\lambda x\ y\ z\ |\ \langle E \rangle\ )\ \langle A \rangle\ \langle B \rangle\ \langle C \rangle$$

Shortcuts for Multi-argument λ's

$$
\begin{aligned}
&\texttt{(}\lambda\texttt{x y z | }\langle\texttt{E}\rangle\texttt{) }\langle\texttt{A}\rangle\ \langle\texttt{B}\rangle\ \langle\texttt{C}\rangle \\
&\equiv\ \texttt{(}\lambda\texttt{x | (}\lambda\texttt{y | (}\lambda\texttt{z | }\langle\texttt{E}\rangle\texttt{)) }\langle\texttt{A}\rangle\ \langle\texttt{B}\rangle\ \langle\texttt{C}\rangle
\end{aligned}
$$

Shortcuts for Multi-argument λ's

$$(\lambda \texttt{x y z} \mid \langle \texttt{E} \rangle\ )\ \langle \texttt{A} \rangle\ \langle \texttt{B} \rangle\ \langle \texttt{C} \rangle$$

$$\equiv (\lambda \texttt{x} \mid (\lambda \texttt{y} \mid (\lambda \texttt{z} \mid \langle \texttt{E} \rangle)\ \langle \texttt{A} \rangle\ \langle \texttt{B} \rangle\ \langle \texttt{C} \rangle$$

$$\xrightarrow{\beta} [\langle \texttt{A} \rangle / \texttt{x}]\ (\lambda \texttt{y} \mid (\lambda \texttt{z} \mid \langle \texttt{E} \rangle)\ \langle \texttt{B} \rangle\ \langle \texttt{C} \rangle$$

Shortcuts for Multi-argument λ's

$(\lambda\texttt{x y z | } \langle\texttt{E}\rangle\texttt{) } \langle\texttt{A}\rangle\ \langle\texttt{B}\rangle\ \langle\texttt{C}\rangle$

$\equiv\ (\lambda\texttt{x | }(\lambda\texttt{y | }(\lambda\texttt{z | }\langle\texttt{E}\rangle))\ \langle\texttt{A}\rangle\ \langle\texttt{B}\rangle\ \langle\texttt{C}\rangle$

$\stackrel{\beta}{\rightarrow}\ [\langle\texttt{A}\rangle/\texttt{x}]\ (\lambda\texttt{y | }(\lambda\texttt{z | }\langle\texttt{E}\rangle))\ \langle\texttt{B}\rangle\ \langle\texttt{C}\rangle$

If $\langle\texttt{A}\rangle$ has free y or z, must rename $(\lambda\texttt{y | }(\lambda\texttt{z | }\langle\texttt{E}\rangle))$

Shortcuts for Multi-argument λ's

$(\lambda \texttt{x y z} \mid \langle \texttt{E} \rangle)\ \langle \texttt{A} \rangle\ \langle \texttt{B} \rangle\ \langle \texttt{C} \rangle$

$\equiv (\lambda \texttt{x} \mid (\lambda \texttt{y} \mid (\lambda \texttt{z} \mid \langle \texttt{E} \rangle))\ \langle \texttt{A} \rangle\ \langle \texttt{B} \rangle\ \langle \texttt{C} \rangle$

$\xrightarrow{\beta} [\langle \texttt{A} \rangle / \texttt{x}]\ (\lambda \texttt{y} \mid (\lambda \texttt{z} \mid \langle \texttt{E} \rangle))\ \langle \texttt{B} \rangle\ \langle \texttt{C} \rangle$

If $\langle \texttt{A} \rangle$ has free $\texttt{y}$ or $\texttt{z}$, must rename $(\lambda \texttt{y} \mid (\lambda \texttt{z} \mid \langle \texttt{E} \rangle))$

$\xrightarrow{\beta}\ [\langle \texttt{B} \rangle / \texttt{y}]\ (\lambda \texttt{z} \mid \langle \texttt{E} \rangle)$

Shortcuts for Multi-argument λ's

$(\lambda \texttt{x y z | } \langle \texttt{E} \rangle \texttt{) } \langle \texttt{A} \rangle \langle \texttt{B} \rangle \langle \texttt{C} \rangle$
$\equiv (\lambda \texttt{x | } (\lambda \texttt{y | } (\lambda \texttt{z | } \langle \texttt{E} \rangle)) \langle \texttt{A} \rangle \langle \texttt{B} \rangle \langle \texttt{C} \rangle$

$\xrightarrow{\beta} [\langle \texttt{A} \rangle/\texttt{x}] (\lambda \texttt{y | } (\lambda \texttt{z | } \langle \texttt{E} \rangle)) \langle \texttt{B} \rangle \langle \texttt{C} \rangle$
If $\langle \texttt{A} \rangle$ has free `y` or `z`, must rename $(\lambda \texttt{y | } (\lambda \texttt{z | } \langle \texttt{E} \rangle))$
$\xrightarrow{\beta} [\langle \texttt{B} \rangle/\texttt{y}] (\lambda \texttt{z | } \langle \texttt{E} \rangle)$
If $\langle \texttt{B} \rangle$ has free `z`, must rename $(\lambda \texttt{z | } \langle \texttt{E} \rangle)$

Shortcuts for Multi-argument λ's

$(\lambda \texttt{x y z | } \langle\texttt{E}\rangle \texttt{) } \langle\texttt{A}\rangle \; \langle\texttt{B}\rangle \; \langle\texttt{C}\rangle$

$\equiv (\lambda \texttt{x | } (\lambda \texttt{y | } (\lambda \texttt{z | } \langle\texttt{E}\rangle)) \; \langle\texttt{A}\rangle \; \langle\texttt{B}\rangle \; \langle\texttt{C}\rangle$

$\xrightarrow{\beta} [\langle\texttt{A}\rangle/\texttt{x}] \; (\lambda \texttt{y | } (\lambda \texttt{z | } \langle\texttt{E}\rangle)) \; \langle\texttt{B}\rangle \; \langle\texttt{C}\rangle$

If $\langle\texttt{A}\rangle$ has free `y` or `z`, must rename $(\lambda \texttt{y | } (\lambda \texttt{z | } \langle\texttt{E}\rangle))$

$\xrightarrow{\beta} \; [\langle\texttt{B}\rangle/\texttt{y}] \; (\lambda \texttt{z | } \langle\texttt{E}\rangle)$

If $\langle\texttt{B}\rangle$ has free `z`, must rename $(\lambda \texttt{z | } \langle\texttt{E}\rangle)$

$\xrightarrow{\beta} \; [\langle\texttt{C}\rangle/\texttt{z}] \; \langle\texttt{E}\rangle$

Shortcuts for Multi-argument λ's

$(\lambda \texttt{x y z | } \langle \texttt{E} \rangle \texttt{) } \langle \texttt{A} \rangle \ \langle \texttt{B} \rangle \ \langle \texttt{C} \rangle$
$\equiv \texttt{ (}\lambda\texttt{x | (}\lambda\texttt{y | (}\lambda\texttt{z | } \langle \texttt{E} \rangle \texttt{)) } \langle \texttt{A} \rangle \ \langle \texttt{B} \rangle \ \langle \texttt{C} \rangle$

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

Example of Multi-argument λ's

- Our basic solution method

```
(λ x y | x y)  (⟨N⟩ y) ⟨M⟩
```

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⟩
```

Example of Multi-argument λ's

- Our basic solution method

$$(\lambda\ \mathtt{x\ y}\ \mid\ \mathtt{x\ y})\quad ((\langle \mathtt{N}\rangle\ \mathtt{y})\ \langle \mathtt{M}\rangle$$

$$\equiv\ (\lambda\ \mathtt{x}\ \mid\ (\lambda \mathtt{y}\ \mid\ \mathtt{x\ y}))\quad (\langle \mathtt{N}\rangle\ \mathtt{y})\langle \mathtt{M}\rangle$$

$$\xrightarrow{\beta}[\ (\langle \mathtt{N}\rangle\quad \mathtt{y})\ /\ \mathtt{x}\ ]\quad (\lambda \mathtt{y}\ \mid\ \mathtt{x\ y})\ \langle \mathtt{M}\rangle$$

Example of Multi-argument λ's

- Our basic solution method

$$(\lambda\ \mathtt{x}\ \mathtt{y}\ |\ \mathtt{x}\ \mathtt{y})\quad (\langle\mathtt{N}\rangle\ \mathtt{y})\ \langle\mathtt{M}\rangle$$
$$\equiv\ (\lambda\ \mathtt{x}\ |\ (\lambda\mathtt{y}\ |\ \mathtt{x}\ \mathtt{y}))\quad (\langle\mathtt{N}\rangle\ \mathtt{y})\langle\mathtt{M}\rangle$$
$$\xrightarrow{\beta}[\ (\langle\mathtt{N}\rangle\ \ \mathtt{y})\ /\ \mathtt{x}\ ]\quad (\lambda\mathtt{y}\ |\ \mathtt{x}\ \mathtt{y})\ \langle\mathtt{M}\rangle$$

Free vars in $(\langle\mathtt{N}\rangle\ \ \mathtt{y})$ get bound?

Example of Multi-argument λ's

- Our basic solution method

  $(\lambda\ \texttt{x y | x y})\ \ (\langle\texttt{N}\rangle\ \texttt{y})\ \langle\texttt{M}\rangle$
  $\equiv\ (\lambda\ \texttt{x | }(\lambda\texttt{y | x y}))\ \ (\langle\texttt{N}\rangle\ \texttt{y})\langle\texttt{M}\rangle$
  $\xrightarrow{\beta}[\ (\langle\texttt{N}\rangle\ \ \texttt{y})\ /\ \texttt{x}\ ]\ \ (\lambda\texttt{y | x y})\ \langle\texttt{M}\rangle$
  Free vars in $(\langle\texttt{N}\rangle\ \ \texttt{y})$ get bound? Yes!

Example of Multi-argument λ's

- Our basic solution method

$(\lambda\ \texttt{x y | x y})\ \ (\langle\texttt{N}\rangle\ \texttt{y})\ \langle\texttt{M}\rangle$
$\equiv\ (\lambda\ \texttt{x | }(\lambda\texttt{y | x y}))\ \ (\langle\texttt{N}\rangle\ \texttt{y})\langle\texttt{M}\rangle$
$\stackrel{\beta}{\rightarrow}\texttt{[}(\langle\texttt{N}\rangle\ \ \texttt{y) / x]}\ \ (\lambda\texttt{y | x y})\ \langle\texttt{M}\rangle$
Free vars in $(\langle\texttt{N}\rangle\ \ \texttt{y})$ get bound? Yes!
Must rename `y` in $(\lambda\texttt{y | x y})$. Say `z`

Example of Multi-argument λ's

- Our basic solution method

  $(\lambda\ \texttt{x y | x y})\ \ (\langle\texttt{N}\rangle\ \texttt{y})\ \langle\texttt{M}\rangle$

  $\equiv\ (\lambda\ \texttt{x}\ |\ (\lambda\texttt{y | x y}))\ \ (\langle\texttt{N}\rangle\ \texttt{y})\langle\texttt{M}\rangle$

  $\xrightarrow{\beta}$ [$(\langle\texttt{N}\rangle\ \ \texttt{y})$ / x] $(\lambda\texttt{y | x y})\ \langle\texttt{M}\rangle$

  Free vars in $(\langle\texttt{N}\rangle\ \ \texttt{y})$ get bound? Yes!

  Must rename y in $(\lambda\texttt{y | x y})$. Say z

  $\xrightarrow{\alpha}$ [$(\langle\texttt{N}\rangle\ \ \texttt{y})$ / x] [z/y] $(\lambda\texttt{y | x y})\ \langle\texttt{M}\rangle$

Example of Multi-argument λ's

- Our basic solution method

  $(\lambda$ x y | x y) ($\langle$N$\rangle$ y) $\langle$M$\rangle$
  $\equiv$ $(\lambda$ x | $(\lambda$y | x y)) ($\langle$N$\rangle$ y)$\langle$M$\rangle$
  $\xrightarrow{\beta}$[($\langle$N$\rangle$ y) / x] $(\lambda$y | x y) $\langle$M$\rangle$
  Free vars in ($\langle$N$\rangle$ y) get bound? Yes!
  Must rename y in $(\lambda$y | x y). Say z
  $\xrightarrow{\alpha}$[($\langle$N$\rangle$ y) / x] [z/y] $(\lambda$y | x y) $\langle$M$\rangle$
  $\equiv$[($\langle$N$\rangle$ y) / x] $(\lambda$z | x z) $\langle$M$\rangle$

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⟩

Example of Multi-argument λ's

- Our basic solution method

$(\lambda\ x\ y\ |\ x\ y)\ \ (\langle N\rangle\ y)\ \langle M\rangle$

$\equiv\ (\lambda\ x\ |\ (\lambda y\ |\ x\ y))\ \ (\langle N\rangle\ y)\langle M\rangle$

$\xrightarrow{\beta}[\ (\langle N\rangle\ \ y)\ /\ x\ ]\ \ (\lambda y\ |\ x\ y)\ \langle M\rangle$

Free vars in $(\langle N\rangle\ \ y)$ get bound? Yes!

Must rename y in $(\lambda y\ |\ x\ y)$. Say z

$\xrightarrow{\alpha}[\ (\langle N\rangle\ \ y)\ /\ x\ ]\ [z/y]\ (\lambda y\ |\ x\ y)\ \langle M\rangle$

$\equiv[\ (\langle N\rangle\ y)\ /\ x\ ]\ (\lambda z\ |\ x\ z)\ \langle M\rangle$

$\equiv(\lambda z\ |\ (\langle N\rangle\ \ y)\ z)\ \langle M\rangle$

$\xrightarrow{\beta}[\langle M\rangle\ /\ z]\ (\langle N\rangle\ y)\ z\ \equiv\ (\langle N\rangle\ y)\ \langle M\rangle$

Example of Multi-argument λ's

- Our basic solution method

$(\lambda\ \mathtt{x}\ \mathtt{y}\ |\ \mathtt{x}\ \mathtt{y})\quad (\langle \mathtt{N}\rangle\ \mathtt{y})\ \langle \mathtt{M}\rangle$

$\equiv\ (\lambda\ \mathtt{x}\ |\ (\lambda \mathtt{y}\ |\ \mathtt{x}\ \mathtt{y}))\quad (\langle \mathtt{N}\rangle\ \mathtt{y})\langle \mathtt{M}\rangle$

$\stackrel{\beta}{\rightarrow}[\ (\langle \mathtt{N}\rangle\ \ \mathtt{y})\ /\ \mathtt{x}\ ]\quad (\lambda \mathtt{y}\ |\ \mathtt{x}\ \mathtt{y})\ \langle \mathtt{M}\rangle$

Free vars in $(\langle \mathtt{N}\rangle\ \ \mathtt{y})$ get bound? Yes!

Must rename $\mathtt{y}$ in $(\lambda \mathtt{y}\ |\ \mathtt{x}\ \mathtt{y})$. Say $\mathtt{z}$

$\stackrel{\alpha}{\rightarrow}[\ (\langle \mathtt{N}\rangle\ \ \mathtt{y})\ /\ \mathtt{x}\ ]\ [\mathtt{z}/\mathtt{y}]\ (\lambda \mathtt{y}\ |\ \mathtt{x}\ \mathtt{y})\ \langle \mathtt{M}\rangle$

$\equiv[\ (\langle \mathtt{N}\rangle\ \mathtt{y})\ /\ \mathtt{x}\ ]\ (\lambda \mathtt{z}\ |\ \mathtt{x}\ \mathtt{z})\ \langle \mathtt{M}\rangle$

$\equiv(\lambda \mathtt{z}\ |\ (\langle \mathtt{N}\rangle\ \ \mathtt{y})\ \mathtt{z})\ \langle \mathtt{M}\rangle$

$\stackrel{\beta}{\rightarrow}[\langle \mathtt{M}\rangle\ /\ \mathtt{z}]\ (\langle \mathtt{N}\rangle\ \mathtt{y})\ \mathtt{z}\ \equiv\ (\langle \mathtt{N}\rangle\ \mathtt{y})\ \langle \mathtt{M}\rangle$

- Note: we replaced $\mathtt{y}$ with $\mathtt{z}$,
  but then immediately replace $\mathtt{z}$ with $\langle \mathrm{M}\rangle$

Example of Multi-argument λ's

- In general, can perform multiple substitutions in parallel

Example of Multi-argument λ's

- In general, can perform multiple substitutions in parallel
- *If substituting in parallel*,
  given `(`λ`x | (`λ`y ... ))` $\langle$`A`$\rangle$ $\langle$`B`$\rangle$,
  we do not have to check for free `y`'s in $\langle$A$\rangle$ as $\langle$B$\rangle$ will be substituted for the "(λy" and any free `y`'s in $\langle$A$\rangle$ will remain free.

Example of Multi-argument λ's

- In general, can perform multiple substitutions in parallel
- *If substituting in parallel*,
  given `(`λ`x | (`λ`y ... ))` $\langle$A$\rangle$ $\langle$B$\rangle$,
  we do not have to check for free `y`'s in $\langle$A$\rangle$ as $\langle$B$\rangle$ will be substituted for the "(λy" and any free `y`'s in $\langle$A$\rangle$ will remain free.
- Example done with multiple substitution

  `(`λ` x y | x y)  (`$\langle$N$\rangle$` y)` $\langle$M$\rangle$

Example of Multi-argument λ's

- In general, can perform multiple substitutions in parallel
- *If substituting in parallel*,
  given $(\lambda x \mid (\lambda y \; \ldots \;)) \; \langle A \rangle \; \langle B \rangle$,
  we do not have to check for free y's in $\langle A \rangle$ as $\langle B \rangle$ will be substituted for the "$(\lambda y$" and any free y's in $\langle A \rangle$ will remain free.
- Example done with multiple substitution

  $(\lambda \; x \; y \mid x \; y) \; \; (\langle N \rangle \; y) \; \langle M \rangle$
  $\xrightarrow{\beta} [(\langle N \rangle \; \; y)/x, \; \langle M \rangle/y] \; (x \; y)$

Example of Multi-argument λ's

- In general, can perform multiple substitutions in parallel
- *If substituting in parallel,*
  given $(\lambda\mathtt{x}\ |\ (\lambda\mathtt{y}\ \ldots\ ))\ \langle\mathtt{A}\rangle\ \langle\mathtt{B}\rangle$,
  we do not have to check for free y's in $\langle A\rangle$ as $\langle B\rangle$ will be substituted for the "$(\lambda$y" and any free y's in $\langle A\rangle$ will remain free.
- Example done with multiple substitution

$$(\lambda\ \mathtt{x}\ \mathtt{y}\ |\ \mathtt{x}\ \mathtt{y})\ \ (\langle\mathtt{N}\rangle\ \mathtt{y})\ \langle\mathtt{M}\rangle$$
$$\xrightarrow{\beta}[(\langle\mathtt{N}\rangle\ \ \mathtt{y})/\mathtt{x},\ \langle\mathtt{M}\rangle/\mathtt{y}]\ (\mathtt{x}\ \mathtt{y})$$
$$\equiv\ (\langle\mathtt{N}\rangle\ \ \mathtt{y})\ \langle\mathtt{M}\rangle$$

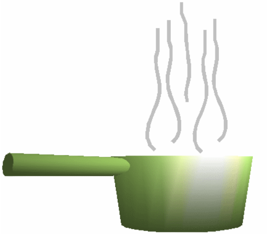
Example of Multi-argument λ's

- In general, can perform multiple substitutions in parallel
- *If substituting in parallel,*
  given $(\lambda \mathtt{x} \mid (\lambda \mathtt{y} \ \ldots\ ))\ \langle \mathtt{A} \rangle\ \langle \mathtt{B} \rangle$,
  we do not have to check for free y's in $\langle A \rangle$ as $\langle B \rangle$ will be substituted for the "(λy" and any free y's in $\langle A \rangle$ will remain free.
- Example done with multiple substitution

  $(\lambda\ \mathtt{x}\ \mathtt{y} \mid \mathtt{x}\ \mathtt{y})\ \ (\langle \mathtt{N} \rangle\ \mathtt{y})\ \langle \mathtt{M} \rangle$
  $\xrightarrow{\beta} [(\langle \mathtt{N} \rangle\ \ \mathtt{y})/\mathtt{x},\ \langle \mathtt{M} \rangle/\mathtt{y}]\ (\mathtt{x}\ \mathtt{y})$
  $\equiv\ (\langle \mathtt{N} \rangle\ \ \mathtt{y})\ \langle \mathtt{M} \rangle$

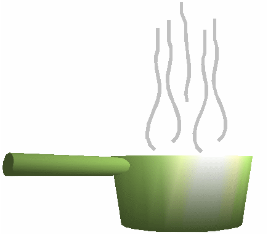
- N.B: still need to check for free vars that get bound when considering substitution of $\langle B \rangle$ in the body of the (λy ...) clause.

Curried functions



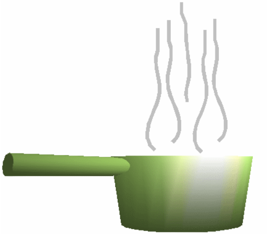
- Can represent n-ary functions as nested unary functions

Curried functions



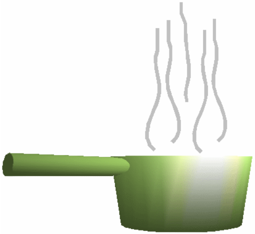
- Can represent n-ary functions as nested unary functions
- (λx y | ⟨E⟩) a b
  ≡ (λx (λy ⟨E⟩)) a b

Curried functions



- Can represent n-ary functions as nested unary functions
- $(\lambda$x y | $\langle$E$\rangle$) a b
  $\equiv$ (λx (λy $\langle$E$\rangle$)) a b
- Can treat an *n*-ary function as a unary function that returns an *n-1*-ary function

Curried functions



- Can represent n-ary functions as nested unary functions
- $(\lambda x\ y\ |\ \langle E \rangle)\ a\ b$
  $\equiv (\lambda x\ (\lambda y\ \langle E \rangle))\ a\ b$
- 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*