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 without 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 without 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 $\lambda$-Calculus

$E_1 \xrightarrow{\rho} E_2$
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 $\lambda$-Calculus
  - Represent initial values as $\lambda$-Calculus expressions
  - Transform $\lambda$-Calculus expressions into new expressions
  - Interpret resulting $\lambda$-Calculus expression to get result
- Transformations $\equiv$ rewriting expressions according to rules
Computation as Rewriting

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

\[
E_1 \xrightarrow{\rho} E_2
\]
Preservation of Semantics

- Every transformed expression preserves semantics of expression
Preservation of Semantics

- Every transformed expression preserves semantics of expression
  - Represent: $2+0$ as $\lambda$-calculus expression $E_1$
Preservation of Semantics

- Every transformed expression preserves semantics of expression
  - Represent: $2+0$ as $\lambda$-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 $\lambda$-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: \( \{ a, b, c, d, \ldots \} \)
  - four symbols: \( \lambda \mid ( ) \)
Syntax

- Lambda Calculus expressions use
  - lower case letters: \{ a, b, c, d, ... \}
  - four symbols: \( \lambda \mid ( ) \)

- Each letter represents a function
Syntax

- Lambda Calculus expressions use
  - lower case letters: \{ a, b, c, d, ... \}
  - four symbols: \( \lambda \mid ( ) \)

- Each letter represents a function

- Three kinds of expressions
  - function constant
  - function definition
  - function application
Examples of Lambda Calculus Expressions

\[ f \]  

\[ (f \ g) \]  

\[ (\lambda x \mid (x \ y)) \]  

definition of function with parameter \( x \) and body \((x \ y)\)  

Body is an application!

\[ (\lambda y \mid (\lambda x \mid (y \ (y \ x)))) \]  

definition of function with parameter \( y \) and body \((\lambda x \mid (y \ (y \ x))))\)
Formal BNF Grammar

\[
\langle \text{expression} \rangle := \langle \text{identifier} \rangle \mid \langle \text{application} \rangle \mid \langle \text{function} \rangle
\]
Formal BNF Grammar

\[ \langle \text{expression} \rangle := \langle \text{identifier} \rangle \mid \langle \text{application} \rangle \mid \langle \text{function} \rangle \]

\[ \langle \text{application} \rangle := "(" \langle \text{expression} \rangle \langle \text{expression} \rangle ")" \]
Formal BNF Grammar

\[ \langle \text{expression} \rangle := \langle \text{identifier} \rangle \ | \ \langle \text{application} \rangle \ | \ \langle \text{function} \rangle \]

\[ \langle \text{application} \rangle := "(\langle \text{expression} \rangle \langle \text{expression} \rangle \)" \]

\[ \langle \text{function} \rangle := "(\lambda \langle \text{identifier} \rangle \ | \ " \langle \text{expression} \rangle \)" \]
Formal BNF Grammar

\[
\langle \text{expression} \rangle : = \langle \text{identifier} \rangle \mid \langle \text{application} \rangle \mid \langle \text{function} \rangle \\
\langle \text{application} \rangle : = "(" \langle \text{expression} \rangle \langle \text{expression} \rangle "\)
\langle \text{function} \rangle : = "(\lambda" \langle \text{identifier} \rangle "|" \langle \text{expression} \rangle "\)
\langle \text{identifier} \rangle : = a \mid b \mid c \mid \ldots
\]
The $\lambda$ Definition

- Function definitions have the form: $(\lambda\langle\text{identifier}\rangle \mid \langle\text{expression}\rangle)$
The $\lambda$ Definition

- Function definitions have the form: $(\lambda\langle\text{identifier}\rangle \mid \langle\text{expression}\rangle)$

- $\lambda$ is followed by a *single* identifier, called a *formal parameter* or *variable*
The $\lambda$ Definition

- Function definitions have the form: $(\lambda\langle\text{identifier}\rangle \mid \langle\text{expression}\rangle)$

- $\lambda$ is followed by a *single* identifier, called a *formal parameter* or *variable*

- When the $\lambda$ is applied to an argument $E$, the *formal parameter* will bind to $E$.
  Below the $\langle\text{identifier}\rangle x$ binds to value $y$
  $(\ (\lambda x \mid \langle\text{body}\rangle \ y \ ) )$
The $\lambda$ Definition

- Function definitions have the form: $(\lambda\langle\text{identifier}\rangle \mid \langle\text{expression}\rangle)$

- $\lambda$ is followed by a *single* identifier, called a *formal parameter or variable*

- When the $\lambda$ is applied to an argument $E$, the *formal parameter will bind to $E$.*
  Below the $\langle\text{identifier}\rangle x$ binds to value $y$
  $( (\lambda x \mid \langle\text{body}\rangle) \ y )$

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

\[
(\lambda x \mid (x \ y)) \equiv (\lambda x \mid x \ y) \equiv (\lambda x \mid xy)
\]
Notational Conveniences

- Where order of operations is clear, can drop brackets
- Can use spacing arbitrarily to aid readability
  - In function definition
    \[
    (\lambda x \mid (x \ y)) \equiv (\lambda x \mid x \ y) \equiv (\lambda x \mid xy)
    \]
  - In function application
    \[
    (f \ g) \equiv f \ g \equiv fg
    \]
Associativity of $\lambda$-calculus operators

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

$$(1+2)+3 = 1 + (2+3)$$
Associativity of $\lambda$-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 $\lambda$-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
  
  \[(5-3)-2 \neq 5-(3-2)\]

- $\lambda$-application is not associative
  
  ($\lambda C$ must be able to represent non-associative functions!)
Associativity of $\lambda$-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

  $$(5-3)-2 \neq 5-(3-2)$$

- $\lambda$-application is not associative
  ($\lambda C$ must be able to represent non-associative functions!)

- By convention, $\lambda$-application is *left-associative*... terms group *from the left*

  $$f \ g \ h \equiv ((f \ g) \ h) \neq (f \ (g \ h))$$
More Left-associativity Examples

\[ f \cdot g \cdot h \equiv (f \cdot g) \cdot h \]
More Left-associativity Examples

\[ f \, g \, h \equiv (f \, g) \, h \quad \text{YES} \]
More Left-associativity Examples

\[ f \, g \, h \equiv (f \, g) \, h \quad \text{YES} \]

\[ (\lambda a \mid (a \, (a \, b))) \equiv (\lambda a \mid a \, a \, b) \]
More Left-associativity Examples

\[ f \ g \ h \ ? \ (f \ g) \ h \quad \text{YES} \]

\[ (\lambda a \ | \ (a \ (a \ b))) \ ? \ (\lambda a \ | \ a \ a \ b) \quad \text{NO} \]
More Left-associativity Examples

\[ f \cdot g \cdot h \equiv (f \cdot g) \cdot h \quad \text{YES} \]

\[ (\lambda a \mid (a \cdot (a \cdot b))) \equiv (\lambda a \mid a \cdot a \cdot b) \quad \text{NO} \]

\[ (\lambda z \mid (a \cdot (\lambda y \mid b))) \equiv (\lambda z \mid a \cdot (\lambda y \mid b)) \]
More Left-associativity Examples

\[ f \cdot g \cdot h \overset{?}{=} (f \cdot g) \cdot h \quad \text{YES} \]

\[(\lambda a \mid (a \cdot (a \cdot b))) \overset{?}{=} (\lambda a \mid a \cdot a \cdot b) \quad \text{NO} \]

\[(\lambda z \mid (a \cdot (\lambda y \mid b))) \overset{?}{=} (\lambda z \mid a \cdot (\lambda y \mid b)) \quad \text{YES} \]
More Left-associativity Examples

\[ f \cdot g \cdot h \equiv (f \cdot g) \cdot h \quad \text{YES} \]

\[ (\lambda a \mid (a \cdot (a \cdot b))) \equiv (\lambda a \mid a \cdot a \cdot b) \quad \text{NO} \]

\[ (\lambda z \mid (a \cdot (\lambda y \mid b))) \equiv (\lambda z \mid a \cdot (\lambda y \mid b)) \quad \text{YES} \]

\[ a \cdot b \cdot (c \cdot d) \equiv a \cdot b \cdot c \cdot d \]
More Left-associativity Examples

\[ f \; g \; h \equiv (f \; g) \; h \quad \text{YES} \]

\[ (\lambda a \; | \; (a \; (a \; b))) \equiv (\lambda a \; | \; a \; a \; b) \quad \text{NO} \]

\[ (\lambda z \; | \; (a \; (\lambda y \; | \; b))) \equiv (\lambda z \; | \; a \; (\lambda y \; | \; b)) \quad \text{YES} \]

\[ a \; b \; (c \; d) \equiv a \; b \; c \; d \quad \text{NO} \]
More Left-associativity Examples

\[ f \ g \ h \ \overset{?}{=} (f \ g) \ h \quad \text{YES} \]

\[ (\lambda a \mid (a \ (a \ b))) \ \overset{?}{=} (\lambda a \mid a \ a \ b) \quad \text{NO} \]

\[ (\lambda z \mid (a \ (\lambda y \mid b))) \ \overset{?}{=} (\lambda z \mid a \ (\lambda y \mid b)) \quad \text{YES} \]

\[ a \ b \ (c \ d) \ \overset{?}{=} a \ b \ c \ d \quad \text{NO} \]

\[ (a \ b) \ c \ d \ \overset{?}{=} a \ b \ c \ d \]
More Left-associativity Examples

\[ f \ g \ h \equiv (f \ g) \ h \quad \text{YES} \]

\[ (\lambda a \mid (a \ (a \ b))) \equiv (\lambda a \mid a \ a \ b) \quad \text{NO} \]

\[ (\lambda z \mid (a \ (\lambda y \mid b))) \equiv (\lambda z \mid a \ (\lambda y \mid b)) \quad \text{YES} \]

\[ a \ b \ (c \ d) \equiv a \ b \ c \ d \quad \text{NO} \]

\[ (a \ b) \ c \ d \equiv a \ b \ c \ d \quad \text{YES} \]
Free and Bound Variables

- An instance of variable $v$ is *bound* in expression $E$ when it is:
  - a formal parameter of a $\lambda$
  - it is enclosed by a $\lambda$ 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 \( \lambda \)
- it is enclosed by a \( \lambda \) with parameter \( v \) within the expression \( E \)

\[
( \underbrace{\lambda x}_\text{bound} \mid z)
\]
Free and Bound Variables

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

( $\lambda x \mid z$ )

bound

( $\lambda x \mid x$ )

bound bound
Free and Bound Variables

- An instance of variable \( v \) is \textit{bound} in expression \( E \) when it is:
  - a formal parameter of a \( \lambda \)
  - it is enclosed by a \( \lambda \) with parameter \( v \) within the expression \( E \)

\[
\begin{align*}
( \lambda x | \; z ) & \quad \text{bound} \\
( \lambda x | \; x ) & \quad \text{bound} \quad \text{bound} \\
( \lambda x | \; y \; x \; z ) & \quad \text{bound} \quad \text{bound}
\end{align*}
\]
Free and Bound Variables

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

\[
(\lambda x \mid z) \quad \text{bound} \\
(\lambda x \mid x) \quad \text{bound, bound} \\
(\lambda x \mid y x z) \quad \text{bound, bound} \\
(\lambda x \mid (\lambda y \mid x y)) \quad \text{bound, bound, bound, bound}
\]
Free and Bound Variables

- An instance of variable \( \nu \) is *bound* in expression \( E \) when it is:
  - a formal parameter of a \( \lambda \)
  - it is enclosed by a \( \lambda \) with parameter \( \nu \) within the expression \( E \)

\[
\begin{align*}
( \lambda x & \mid z ) \\
\text{bound} \\
( \lambda x & \mid x ) \\
\text{bound} & \text{ bound} \\
( \lambda x & \mid y x z ) \\
\text{bound} & \text{ bound} \\
( \lambda x & \mid ( \lambda y \mid x y ) ) \\
\text{bound} & \text{ bound} & \text{ bound} & \text{ bound} \\
( \lambda x & \mid ( \lambda y \mid x y ) ) \\
\text{bound} & \text{ bound} & \text{ bound} & \text{ bound}
\end{align*}
\]
Free and Bound Variables

- A variable that is not bound is *free*

\[
\begin{align*}
y & \text{ free} \\
\end{align*}
\]
Free and Bound Variables

- A variable that is not bound is *free*

  \[
  \text{free} \quad \lambda x \quad \text{bound} \quad \text{free}
  \]

  \[
  y \quad \text{free}
  \]

  \[
  (\lambda x \quad | \quad y \quad ) \quad \text{bound} \quad \text{free}
  \]
Free and Bound Variables

- A variable that is not bound is *free*

\[
\begin{align*}
\text{free} \\
( \lambda x \mid y ) \\
\text{bound free} \\
( y ( \lambda y \mid y ) ) \\
\text{free bound bound}
\end{align*}
\]
Free and Bound Variables

- A variable that is not bound is \textit{free}

```
\[ y \]
\[ (\lambda x \mid y) \]
\[ (y (\lambda y \mid y)) \]
\[ (\lambda x \mid (\mid q \ y) \mid y) \]
\[ \text{bound} \quad \text{free} \quad \text{bound} \quad \text{free} \]
```

Dr. B. Price and Dr. R. Greiner
CMPUT 325 : Lambda Calculus Basics
Free and Bound Variables

- A variable that is not bound is *free*

\[
\begin{align*}
\text{free} & \quad (\lambda x \mid y) \\
\text{bound} & \quad \text{free} \quad (y (\lambda y \mid y)) \\
\text{free} & \quad \text{bound} \quad \text{bound} \quad (\lambda x \mid (\mid q \mid y)) \\
\end{align*}
\]

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

\[
(\lambda x \mid x \mid y) \quad (\lambda y \mid x \mid y \mid z) \quad y
\]

\[
\begin{align*}
\text{bound} & \quad \text{bound} \quad \text{free} \quad \text{bound} \quad \text{bound} \quad \text{bound} \quad \text{bound} \quad \text{free} \quad \text{free}
\end{align*}
\]
More on Variables in λ-calculus

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

\[(\lambda x \mid x \ y)\]

bound \hspace{1cm} bound \hspace{1cm} free
More on Variables in $\lambda$-calculus

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

$$\left( \lambda x \mid x \ y \right) \quad \text{(bound, bound, free)}$$

$$\left( \lambda y \left( \lambda x \mid x \ y \right) \right) \quad \text{(bound, bound, bound, bound)}$$
More on Variables in $\lambda$-calculus

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

\[
(\lambda x \mid x \ y) \\
\text{bound} \quad \text{bound} \quad \text{free}
\]

\[
(\lambda y (\lambda x \mid x \ y)) \\
\text{bound} \quad \text{bound} \quad \text{bound} \quad \text{bound}
\]

- Variables in $\lambda$-calculus derive their meaning from the argument the enclosing $\lambda$ is applied to
More on Variables in $\lambda$-calculus

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

\[
(\lambda x \mid x \ y)
\]

bound \hspace{1cm} bound \hspace{1cm} free

\[
(\lambda y (\lambda x \mid x \ y))
\]

bound \hspace{1cm} bound \hspace{1cm} bound

- Variables in $\lambda$-calculus derive their meaning from the argument the enclosing $\lambda$ is applied to
  - They cannot be "assigned" a new "value"
More Convenience: Collapsing Enclosing $\lambda$’s

- The scope of a $\lambda$ is the expression to which its bindings apply

$$((\lambda x \mid (\lambda y \mid y) \ x) ) \ ((\lambda w \mid w) \ v )$$
More Convenience: Collapsing Enclosing $\lambda$’s

- The scope of a $\lambda$ is the expression to which its bindings apply

  \[(\lambda x \mid (\lambda y | y) x) \quad ((\lambda w \mid w) v)\]

- Scope of outer $\lambda$ includes $(\lambda y | y) x$
More Convenience: Collapsing Enclosing \( \lambda \)'s

- The scope of a \( \lambda \) is the expression to which its bindings apply

\[
(\lambda x \ | \ (\lambda y \ | \ y) \ x) \ ( (\lambda w \ | \ w) \ v )
\]

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

- If the scopes of nested \( \lambda \)'s coincide, the arguments can be coalesced into a multi-argument \( \lambda \)

\[
( \lambda x \ | \ (\lambda y \ | \ x \ y)) \equiv (\lambda y \ | \ x \ y)
\]
More Convenience: Collapsing Enclosing $\lambda$’s

- The scope of a $\lambda$ is the expression to which its bindings apply
  \[(\lambda x \mid (\lambda y \mid y) \ x) \ (\lambda w \mid w) \ v\]

- Scope of outer $\lambda$ includes $(\lambda y \mid y) \ x$

- If the scopes of nested $\lambda$’s coincide, the arguments can be coalesced into a multi-argument $\lambda$
  \[(\lambda x \mid (\lambda y \mid x \ y)) \equiv (\lambda xy \mid xy)\]

- Just notational convenience!!
λ-calculus Computation

- λ-calculus computation is ...
  Reducing complex expression to “simpler” form
\(\lambda\)-calculus Computation

- \(\lambda\)-calculus computation is ...
  - Reducing complex expression to “simpler” form
    - \( ( (\lambda z \mid (z \ y)) \ w ) \)

- Need to Replace every occurrence of \(z\) in \((z \ y)\) with \((\lambda w \mid w)\).
  - Replace every occurrence of \(\langle\text{identifier}\rangle\) in \(\langle\text{expression}\rangle\) with \(\langle\text{expression'}\rangle\).
\(\lambda\)-calculus Computation

- \(\lambda\)-calculus computation is ...
  Reducing complex expression to “simpler” form
  \[
  ( (\lambda z \mid (z \ y)) \ w ) \rightarrow (w \ y)
  \]
  [Every occurrence of \(z \rightarrow w\)]
\(\lambda\)-calculus Computation

- \(\lambda\)-calculus computation is ...
  Reducing complex expression to “simpler” form
  
  \[
  ( (\lambda z \ | \ (z \ y)) \ w ) \rightarrow (w \ y)
  \]
  [Every occurrence of \(z \rightarrow w\)]

  \[
  ( (\lambda z \ | \ (z \ y)) \ (\lambda w \ | \ w) )
  \]
λ-calculus Computation

- λ-calculus computation is ...
  Reducing complex expression to “simpler” form
  - $( (\lambda z \mid (z \ y)) \ w ) \rightarrow (w \ y)$
    [Every occurrence of $z \rightarrow w$]
  - $( (\lambda z \mid (z \ y)) (\lambda w \mid w) ) \rightarrow ( (\lambda w \mid w) \ y )$
    [Every occurrence of $z \rightarrow (\lambda w \mid w)$]
\(\lambda\)-calculus Computation

\(\lambda\)-calculus computation is ...
Reducing complex expression to “simpler” form

\[
( (\lambda z \mid (z \ y)) \ w ) \rightarrow (w \ y)
\]
[Every occurrence of \(z \rightarrow w\)]

\[
( (\lambda z \mid (z \ y)) \ (\lambda w \mid w) ) \rightarrow ( (\lambda w \mid w) \ y )
\]
[Every occurrence of \(z \rightarrow (\lambda w \mid w)\)]
\[
( (\lambda w \mid w) \ y ) \rightarrow y
\]
[Every occurrence of \(w \rightarrow y\)]
λ-calculus Computation

- λ-calculus computation is ...
  Reducing complex expression to “simpler” form
  - \(( (\lambda z \mid (z \ y)) \ w ) \rightarrow (w \ y)\)
    [Every occurrence of \(z \rightarrow w\)]
  - \(( (\lambda z \mid (z \ y)) \ (\lambda w \mid w ) ) \rightarrow ( (\lambda w \mid w ) \ y ) \)
    [Every occurrence of \(z \rightarrow (\lambda w \mid w )\)]
    \(( (\lambda w \mid w ) \ y ) \) \rightarrow y
    [Every occurrence of \(w \rightarrow y\)]

- Need to
  “Replace every occurrence of \(z\) in \((z \ y)\) with \((\lambda w \mid w)\)”
  “Replace every occurrence of \(\langle\text{identifier}\rangle\) in \(\langle\text{expression}\rangle\) with \(\langle\text{expression’}\rangle\)”
Substitution

- $\lambda$-calculus rules use a special substitution

$$\langle E \rangle [x/y] \rightarrow \langle E \rangle \ (z \ x \ y)$$

In $\lambda$-calculus, only free variables are replaced:

$$\langle \lambda y |z \ y \rangle \ (z \ a \ b) [x/y] \rightarrow \langle \lambda y |z \ y \rangle \ (z \ a \ b)$$
Substitution

- \( \lambda \)-calculus rules uses a special substitution

- Generally: write substitution of \( x \) for \( y \) in expression \( \langle E \rangle \) as: \( [x/y] \langle E \rangle \)

\[
[x/y] (z \ y) \rightarrow
\]
Substitution

- λ-calculus rules uses a special substitution

- Generally: write substitution of \( x \) for \( y \) in expression \( \langle E \rangle \) as: \([x/y] \langle E \rangle\)

\[ [x/y] (z \ y) \rightarrow (z \ x) \]
Substitution

- \(\lambda\)-calculus rules uses a special substitution

- Generally: write substitution of \(x\) for \(y\) in expression \(\langle E \rangle\) as: \([x/y] \langle E \rangle\)

\[
[x/y] (z y) \rightarrow (z x)
\]

\[
[(\text{a b})/y] (\lambda z \mid (z y)) \rightarrow
\]
Substitution

- \( \lambda \)-calculus rules uses a special substitution

- Generally: write substitution of \( x \) for \( y \) in expression \( \langle E \rangle \) as: \( [x/y] \langle E \rangle \)

\[
[x/y] (z \ y) \rightarrow (z \ x)
\]
\[
[(a \ b)/y] (\lambda z \ |(z \ y)) \rightarrow (\lambda z (z \ (a \ b)))
\]
Substitution

- \( \lambda \)-calculus rules uses a special substitution

- Generally: write substitution of \( x \) for \( y \) in expression \( \langle E \rangle \) as: \( [x/y] \langle E \rangle \)

  \[
  [x/y] (z \ y) \rightarrow (z \ x)
  \]

  \[
  [(a \ b)/y] (\lambda z \ | (z \ y)) \rightarrow (\lambda z (z (a \ b)))
  \]

- In \( \lambda \)-calculus, only free variables are replaced:

  \[
  [x/y] (z (\lambda y \ | z \ y)) \rightarrow
  \]
Substitution

- $\lambda$-calculus rules uses a special substitution

- Generally: write substitution of $x$ for $y$ in expression $\langle E \rangle$ as: $[x/y] \langle E \rangle$

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

- In $\lambda$-calculus, only free variables are replaced:

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

$$[x/y] \ (\lambda z \ | \ yz) \rightarrow (\lambda z \ | \ xz)$$
Legal Substitution

- Legal substitutions do not change meaning of an expression
  - Legal: Substitute x for y
    
    \[
    [x/y] (\lambda z \mid yz) \rightarrow (\lambda z \mid xz)
    \]

- Illegal substitutions introduce bindings not present in original expressions
  - Illegal because variable named y in \( (\lambda z \mid yz) \) was free but now, as z, is bound
  
  \[
  [x/y] (\lambda x \mid xz) \rightarrow (\lambda z \mid (\lambda x \mid xz) z)
  \]
Legal Substitution

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

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

▷ Illegal substitutions introduce bindings not present in original expressions

$$[z/y] (\lambda z \mid yz) \not\rightarrow (\lambda z \mid zz)$$
  ▷ Illegal because variable named $y$ in $(\lambda z \mid yz)$ was free but now, as $z$, is bound
Legal Substitution

Legal substitutions do not change meaning of an expression

- Legal: Substitute $x$ for $y$

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

Illegal substitutions introduce bindings not present in original expressions

- Illegal because variable named $y$ in $(\lambda z \mid yz)$ was free but now, as $z$, is bound

\[ [z/y] (\lambda z \mid yz) \not\rightarrow (\lambda z \mid zz) \]

- Illegal because $z$ was free in $(\lambda x \mid xz)$ but now is bound

\[ [ (\lambda x \mid xz) / y ] (\lambda z \mid yz) \not\rightarrow (\lambda z \mid (\lambda x \mid xz) z) \]
A function application $((\lambda x \mid E) \mid F)$ has function $(\lambda x \mid E)$ and argument $F$. 

This represents applying the function to the argument, which involves substituting the argument for every free occurrence of the variable in the body of the function.
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 occurrence of \(x\) in body \langle E \rangle

\[\text{eval}\left[(\lambda x \mid \langle E \rangle) \langle F \rangle\right]\]
\(\beta\) (Beta Rule): Function Application

- A function application \(\left( (\lambda x \mid \langle E \rangle) \langle F \rangle \right)\) has function \(\lambda x \mid \langle E \rangle\) and argument \(\langle F \rangle\)

- \(\beta\)-rule: apply \(\lambda x \mid \langle E \rangle\) to \(\langle F \rangle\)
  \[\equiv\]
  substitute \(\langle F \rangle\) for every free occurrence of \(x\) in body \(\langle E \rangle\)

\[\text{eval}\left[ (\lambda x \mid \langle E \rangle) \langle F \rangle \right] \equiv [\langle F \rangle/x]_\lambda E \quad \text{if} \ [\langle F \rangle/x]_\lambda \text{ is legal}\]
\( \beta \) (Beta Rule): Function Application

- A function application \(((\lambda x | \langle E \rangle) \langle F \rangle)\) has function \((\lambda x | \langle E \rangle)\) and argument \(\langle F \rangle\)

- \(\beta\)-rule: apply \((\lambda x | \langle E \rangle)\) to \(\langle F \rangle\)
  \[ \equiv \]
  substitute \(\langle F \rangle\) for every free occurrence of \(x\) in body \(\langle E \rangle\)

\[
\text{eval}[ (\lambda x | \langle E \rangle) \langle F \rangle ) ] \\
\equiv [\langle F \rangle/x]_\lambda E \quad \text{if } [\langle F \rangle/x]_\lambda \text{ is legal}
\]

- \(\beta\) defines a relationship between manipulation of symbols and a computation
Substitution Legality and the $\beta$-rule

$\beta$-rule starts with application: $\left( \left( \lambda x \mid \langle E \rangle \right) \langle F \rangle \right)$
Substitution Legality and the $\beta$-rule

- $\beta$-rule starts with application: $((\lambda x \mid \langle E \rangle) \langle F \rangle)$

- Substitution is *illegal* only if
  $\exists$ free occurrences of variables in $\langle F \rangle$ that would become bound in $\langle E \rangle$
Substitution Legality and the $\beta$-rule

- $\beta$-rule starts with application: \(( (\lambda x \mid \langle E \rangle) \langle F \rangle )\)

- Substitution is *illegal* only if
  \[ \exists \text{ free occurrences of variables in } \langle F \rangle \text{ that would become bound in } \langle E \rangle \]

- Later, a way to fix things when a substitution would be illegal
\( \beta \) example: constant argument

\[
(\lambda f \mid (f \ x)) \ s \\
\beta \\
\rightarrow
\]
\( \beta \) example: constant argument

\[
(\lambda f \ | \ (f \ x)) \ s \\
\xRightarrow{\beta} \ [s/f] \ (f \ x)
\]
$\beta$ example: constant argument

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

$\beta\rightarrow [s/f] (f \ x)$

free variables in $s$ that would get bound?
$\beta$ example: constant argument

\[(\lambda f \mid (f \ x)) \ s\n\]

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

free variables in $s$ that would get bound?

No, go ahead and substitute

$\equiv(s \ x)$
\[ (\lambda f \; | \; (f \; x)) \; s \]

\[ \beta \rightarrow [s/f] \; (f \; x) \]

free variables in \( s \) that would get bound?
No, go ahead and substitute
\[ \equiv (s \; x) \]
Can we do more?
\( \beta \) example: constant argument

\[
(\lambda f \ | \ (f \ x)) \ s
\] \[\beta \rightarrow [s/f] \ (f \ x)\]

free variables in \( s \) that would get bound?
No, go ahead and substitute
\[\equiv (s \ x)\]
Can we do more? No - normal form
\( \beta \) example: \( \lambda \)argument

\[
( (\lambda f \mid (f \ x)) \ (\lambda y \mid y) )
\]

\( \beta \)

\[
\rightarrow
\]

Free var\(s\) in \( (\lambda y \mid y) \) get bound? No.

\[
\equiv ( (\lambda y \mid y) x )
\]

Free var\(s\) in \( x \) get bound? No.

Dr. B. Price and Dr. R. Greiner
CMPUT 325 : Lambda Calculus Basics
β example: λargument

\[(\lambda f \mid (f \ x)) \ (\lambda y \mid y)\]

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

Free vars in (λy \mid y) get bound?
\( \beta \) example: \( \lambda \)argument

\[
\left( (\lambda f \mid f \; x) \; (\lambda y \mid y) \right)
\]

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

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

\( \equiv \left( (\lambda y \mid y) \; x \right) \)
\( \beta \) example: \( \lambda \)argument

\[
( (\lambda f \mid (f \ x)) (\lambda y \mid y) )
\]
\[
\beta \rightarrow \left[ (\lambda y \mid y)/f \right] (f \ x)
\]
Free vars in \( (\lambda y \mid y) \) get bound? No.
\[
\equiv ( (\lambda y \mid y) \ x)
\]

\[
( (\lambda y \mid y) \ x)
\]
\[
\beta \rightarrow \left[ x / y \right] y
\]
Free vars in \( x \) get bound?
\( \beta \) example: \( \lambda \)argument

\[
( (\lambda f \mid (f \, x)) \, (\lambda y \mid y) )
\]

\[
\beta \rightarrow [(\lambda y \mid y) / f] \, (f \, x)
\]

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

\[\equiv ( (\lambda y \mid y) \, x)\]

\[
( (\lambda y \mid y) \, x)
\]

\[
\beta \rightarrow [x / y] \, y
\]

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

\[\equiv x\]
\[ (\lambda f \mid (f (f\ x))) \ s) \]
\[ \xrightarrow{\beta} [s /f] (f (f\ x)) \]
Free vars in \( s \) get bound?
\( \beta \) example: constant substitutions

\[
( (\lambda f \mid (f (f \ x))) \ s) \\
\beta \rightarrow \ [s /f] (f (f \ x))
\]
Free vars in \( s \) get bound? No.
\( \equiv (s (s \ x)) \)
\[ (\lambda f \mid (f (f \ x))) \ s) \]
\[ \xrightarrow[\beta]{} [s / f] (f (f \ x)) \]
Free vars in \( s \) get bound? No.
\[ \equiv (s (s \ x)) \]

Can we do more?
\[ (\lambda f \, (f \, (f \, x))) \, s) \]
\[ \xrightarrow{\beta} [s \, /f] \, (f \, (f \, x)) \]
Free vars in \( s \) get bound? No.
\[ \equiv (s \, (s \, x)) \]

Can we do more? No - in normal form
\(\beta\) example: \(\lambda\) substitutions

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

Free vars in \((f \ (f \ x))\) get bound?
\( \beta \) example: \( \lambda \) substitutions

\[
((\lambda f \, \, (f \, (f \, x))) \, \, (\lambda y \, \, y))
\]

\( \beta \rightarrow [ (\lambda y \, \, y) / f ] \, \, (f \, (f \, x)) \)

Free vars in \( f \, (f \, x) \) get bound?
\[ (\lambda f \; ((f \; (f \; x))) \; (\lambda y \; y)) \]

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

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

\[ \equiv (\lambda y \; y) \; ((\lambda y \; y) \; x) \]
$\beta$ example: $\lambda$ substitutions

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

$\beta \rightarrow [(\lambda y \; l \; y) \; / \; f ] \; (f \; (f \; x))$

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

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

Are we done?
\(\beta\) example: \(\lambda\) substitutions

\[
\begin{align*}
( (\lambda f \ f (f \ x)) ) & \ (\lambda y \ y) ) \\
\xrightarrow{\beta} [ (\lambda y \ y) / f ] \ (f \ (f \ x)) \\
\text{Free vars in } (f \ (f \ x)) \text{ get bound? No.} \\
\equiv ( (\lambda y \ y) \ ((\lambda y \ y) \ x)) \\
\text{Are we done? No}
\end{align*}
\]
\[ \beta \text{ example: } \lambda \text{ substitutions} \]

\[
( (\lambda f \mid (f \ (f \ x))) \ (\lambda y \mid y) ) \quad \beta \rightarrow [ (\lambda y \mid y) / f ] \ (f \ (f \ x))
\]

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

\[
\equiv ( (\lambda y \mid y) \ ((\lambda y \mid y) \ x) )
\]

Are we done? No

\[
( (\lambda y \mid y) \ ((\lambda y \mid y) \ x) ) \quad \beta \rightarrow ( (\lambda y \mid y) \ [x / y] \ y )
\]
\( \beta \) example: \( \lambda \) substitutions

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

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

\[
\equiv ( (\lambda y | y) \ ((\lambda y | y) \ x) )
\]

Are we done? No

\[
( (\lambda y | y) \ ((\lambda y | y) \ x) ) \\
\xrightarrow{\beta} ( (\lambda y | y) \ [x / y] y ) \\
( (\lambda y | y) \ x )
\]
\( \beta \) example: \( \lambda \) substitutions

\[
\beta \rightarrow [ (\lambda y \mid y) / f ] \ (f \ (f \ x))
\]

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

\[
\equiv ( (\lambda y \mid y) ((\lambda y \mid y) x))
\]

Are we done? No

\[
(\ (\lambda y \mid y) \ ((\lambda y \mid y) x))
\]

\[
\beta \rightarrow ( (\lambda y \mid y) [x / y] y )
\]

( (\lambda y \mid y) x )

Now are we done?
\[ \beta \text{ example: } \lambda \text{ substitutions} \]

\[ ( (\lambda f \, (f \, (f \, x))) \, (\lambda y \, y) ) \]
\[ \xrightarrow{\beta} [ (\lambda y \, y) \, / \, f ] \, \, (f \, (f \, x)) \]

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

\[ \equiv ( (\lambda y \, y) \, ((\lambda y \, y) \, x)) \]

Are we done? No

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

Now are we done? No
\[ \beta \text{ example: } \lambda \text{ substitutions} \]

\[
\left( \left( \lambda f \mid (f \ (f \ x)) \right) \right) \ (\lambda y \mid y)
\]
\[\xrightarrow{\beta} \left[ (\lambda y \mid y) / f \right] \ (f \ (f \ x)) \]

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

\[\equiv (\lambda y \mid y) \ ((\lambda y \mid y) \ x)\]

*Are we done? No*

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

*Now are we done? No*

\[
\left( \lambda y \mid y \right) \ x \xrightarrow{\beta} \ [x / y] \ y
\]
\[\rightarrow \]
\[ ( (\lambda f \ (f \ (f \ x))) \ (\lambda y \ y) ) \]

\[ \beta \rightarrow [ (\lambda y \ y) / f ] \ (f \ (f \ x)) \]

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

\[ \equiv ( (\lambda y \ y) \ ((\lambda y \ y) \ x)) \]

*Are we done?* No

\[ ( (\lambda y \ y) \ ((\lambda y \ y) \ x)) \]

\[ \beta \rightarrow ( (\lambda y \ y) \ [x / y] \ y ) \]

\[ ( (\lambda y \ y) \ x) \]

*Now are we done?* No

\[ ( (\lambda y \ y) \ x) \ \beta \rightarrow [x / y] \ y \]

\[ \rightarrow x \]
\[ \beta \text{ example: complex multiple substitution} \]

\[ ( (\lambda f \mid (f (f \, x))) \mid (\lambda y \mid (g (g (g \, y)))) ) \]
\[ \beta \text{ example: complex multiple substitution} \]

\[
( (\lambda f \mid (f \ (f \ x))) \ (\lambda y \mid (g \ (g \ (g \ y)))) ) \\
\equiv (\lambda f \mid (f \ (f \ x))) \ (\lambda y \mid (g \ (g \ (g \ y))))
\]
\( \beta \) example: complex multiple substitution

\[
( (\lambda f \mid (f (f \ x))) \ (\lambda y \mid (g (g (g y)))) )
\]
\[
\equiv (\lambda f \mid (f (f \ x))) \ (\lambda y \mid (g (g (g y))))
\]
\[
\beta \rightarrow [(\lambda y \mid (g (g (g y)))) / f] \ (f (f \ x))
\]
\( \beta \) example: complex multiple substitution

\[
\left( \left( \lambda f \mid (f \ (f \ x)) \right) \right) \left( \lambda y \mid (g \ (g \ (g \ y))) \right)
\]

\[
\equiv \left( \lambda f \mid (f \ (f \ x)) \right) \left( \lambda y \mid (g \ (g \ (g \ y))) \right)
\]

\[
\beta \rightarrow [(\lambda y \mid (g \ (g \ (g \ y))) \ )/f] \ (f \ (f \ x))
\]

Free vars in \( \left( \lambda y \mid (g \ (g \ (g \ y))) \right) \ ) get bound?
\[ \beta \text{ example: complex multiple substitution} \]

\[
( (\lambda f \mid (f \ (f \ x))) \ (\lambda y \mid (g \ (g \ (g \ y)))) ) \equiv (\lambda f \mid (f \ (f \ x))) \ (\lambda y \mid (g \ (g \ (g \ y))))
\]

\[
\beta \rightarrow [((\lambda y \mid (g \ (g \ (g \ y)))) \ / f] \ (f \ (f \ x))
\]

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

\[
\equiv ( (\lambda y \mid (g \ (g \ (g \ y)))) \ ( (\lambda y \mid (g \ (g \ (g \ y)))) \ x)))
\]
\( \beta \) example: complex multiple substitution

\[
( (\lambda f \mid (f (f x))) \quad (\lambda y \mid (g (g (g y)))) \quad )
\]
\[
\equiv( (\lambda f \mid (f (f x))) \quad (\lambda y \mid (g (g (g y)))) \quad )
\]
\[
\beta \rightarrow [ (\lambda y \mid (g (g (g y)))) / f ] \quad (f (f x))
\]
Free vars in \((\lambda y \mid (g (g (g y))))\) get bound? No
\[
\equiv( (\lambda y \mid (g (g (g y)))) \quad (\lambda y \mid (g (g (g y)))) \quad x))
\]
\[
( (\lambda y \mid (g (g (g y)))) \quad (\lambda y \mid (g (g (g y)))) \quad x))
\]
\[
\beta \rightarrow ( (\lambda y \mid (g (g (g y)))) \quad [x/y] \quad (g (g (g y)))) \quad )
\]
\[
\equiv
\]
\[ (\lambda f \mid (f \, (f \, x))) \, (\lambda y \mid (g \, (g \, (g \, y)))) \]

\[ \equiv (\lambda f \mid (f \, (f \, x))) \, (\lambda y \mid (g \, (g \, (g \, y)))) \]

\[ \beta \rightarrow [(\lambda y \mid (g \, (g \, (g \, y)))) \, /f] \, (f \, (f \, x)) \]

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

\[ \equiv ( (\lambda y \mid (g \, (g \, (g \, y)))) \, ( (\lambda y \mid (g \, (g \, (g \, y)))) \, x)) ) \]

\[ (\lambda y \mid (g \, (g \, (g \, y)))) \, (((\lambda y \mid (g \, (g \, (g \, y)))) \, x)) ) \]

\[ \beta \rightarrow ( (\lambda y \mid (g \, (g \, (g \, y)))) \, [x/y] \, (g \, (g \, (g \, y)))) ) \]

\[ \equiv ( (\lambda y \mid (g \, (g \, (g \, y)))) \, (g \, (g \, (g \, x)))) ) \]
**β example: complex multiple substitution**

\[
( (\lambda f \mid (f \ (f \ x))) \ (\lambda y \mid (g \ (g \ (g \ y)))) )
\]

\[
\equiv ( (\lambda f \mid (f \ (f \ x))) \ (\lambda y \mid (g \ (g \ (g \ y)))) )
\]

\[
\beta \rightarrow [((\lambda y \mid (g \ (g \ (g \ y)))) \ / f] (f \ (f \ x))
\]

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

\[
\equiv ( (\lambda y \mid (g \ (g \ (g \ y)))) \ ((\lambda y \mid (g \ (g \ (g \ y)))) \ x)) )
\]

\[
\beta \rightarrow ( (\lambda y \mid (g \ (g \ (g \ y)))) \ [x/y] (g \ (g \ (g \ y))))
\]

\[
\equiv ( (\lambda y \mid (g \ (g \ (g \ y)))) \ (g \ (g \ (g \ x))))
\]

\[
( (\lambda y \mid (g \ (g \ (g \ y)))) \ (g \ (g \ (g \ x))))
\]

\[
\beta \rightarrow [(g \ (g \ (g \ x))) \ / y] (g \ (g \ (g \ y)))
\]

\[
\rightarrow
\]
\( \beta \) example: complex multiple substitution

\[
\begin{align*}
( (\lambda f \mid (f (f \ x))) (\lambda y \mid (g (g (g \ y)))) ) &
\equiv ( (\lambda f \mid (f (f \ x))) (\lambda y \mid (g (g (g \ y)))) ) \\
\beta &\rightarrow [(\lambda y \mid (g (g (g \ y))))/f] (f (f \ x)) \\
\text{Free vars in } (\lambda y \mid (g (g (g \ y)))) \text{ get bound? No} \\
\equiv ( (\lambda y \mid (g (g (g \ y)))) ((\lambda y \mid (g (g (g \ y)))) \ x))) \\
\end{align*}
\]

\[
\begin{align*}
( (\lambda y \mid (g (g (g \ y)))) ((\lambda y \mid (g (g (g \ y)))) \ x)) &
\equiv ( (\lambda y \mid (g (g (g \ y)))) (g (g (g \ x)))) \\
\beta &\rightarrow (\lambda y \mid (g (g (g \ y)))) [x/y] (g (g (g \ y))) \\
\equiv ( (\lambda y \mid (g (g (g \ y)))) (g (g (g \ x)))) \\
\end{align*}
\]

\[
\begin{align*}
( (\lambda y \mid (g (g (g \ y)))) (g (g (g \ x)))) &
\beta \rightarrow [(g (g (g \ x))) / y] (g (g (g \ y))) \\
\rightarrow (g (g (g (g (g \ x)))))) \\
\end{align*}
\]
A formal definition of $\beta$-substitution

- Let $\langle E \rangle$, $\langle F \rangle$ and $\langle G \rangle$ be $\lambda$-calculus expressions; $x$ and $y$ be distinct $\lambda$-calculus identifiers (constants)
A formal definition of $\beta$-substitution

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

$$\left[\langle E \rangle / x\right] x \rightarrow \langle E \rangle$$
A formal definition of $\beta$-substitution

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

$\left[ \langle E \rangle \;/\; x \right] x \rightarrow \langle E \rangle$

$\left[ \langle E \rangle \;/\; x \right] y \rightarrow y$
A formal definition of $\beta$-substitution

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

$\left[\langle E \rangle / x\right] x \rightarrow \langle E \rangle$

$\left[\langle E \rangle / x\right] y \rightarrow y$

$\left[\langle E \rangle / x\right] (\langle F \rangle \langle G \rangle) \rightarrow (\left[\langle E \rangle / x\right] \langle F \rangle \left[\langle E \rangle / x\right] \langle G \rangle)$
A formal definition of $\beta$-substitution

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

\[
\begin{align*}
[\langle E \rangle \ / \ x] \ x & \rightarrow \langle E \rangle \\
[\langle E \rangle \ / \ x] \ y & \rightarrow \ y \\
[\langle E \rangle \ / \ x] \ (\langle F \rangle \ \langle G \rangle) & \rightarrow ( [\langle E \rangle \ / \ x] \ \langle F \rangle \ [\langle E \rangle \ / \ x] \ \langle G \rangle ) \\
[\langle E \rangle \ / \ x] \ (\lambda x \ | \ \langle F \rangle) & \rightarrow (\lambda x \ | \ \langle F \rangle)
\end{align*}
\]
A formal definition of $\beta$-substitution

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

- $\left[ \langle E \rangle / \ x \right] \ x \rightarrow \langle E \rangle$
- $\left[ \langle E \rangle / \ x \right] \ y \rightarrow \ y$
- $\left[ \langle E \rangle / \ x \right] (\langle F \rangle \langle G \rangle) \rightarrow (\left[ \langle E \rangle / \ x \right] \langle F \rangle \left[ \langle E \rangle / \ x \right] \langle G \rangle)$
- $\left[ \langle E \rangle / \ x \right] (\lambda x \mid \langle F \rangle) \rightarrow (\lambda x \mid \langle F \rangle)$
- $\left[ \langle E \rangle / \ x \right] (\lambda y \mid \langle F \rangle)$ where $\langle E \rangle$ has no free instances of $y$
  \[ \rightarrow (\lambda y \mid \left[ \langle E \rangle / \ x \right] \langle F \rangle) \]
Variable Names in $\lambda$-calculus

- The identifier used to represent a BOUND variable is irrelevant
Variable Names in $\lambda$-calculus

- The identifier used to represent a BOUND variable is irrelevant
- Meaning of variable based on the $\lambda$ that introduces it ... and how it is used in $\lambda$’s body
Variable Names in \( \lambda \)-calculus

- The identifier used to represent a BOUND variable is irrelevant.
- Meaning of variable based on the \( \lambda \) that introduces it ... and how it is used in \( \lambda \)'s body.
- If we change the identifier used in a formal parameter and all of its bound occurrences, the meaning of the expression is unaltered.
Variable Names in \( \lambda \)-calculus

- The identifier used to represent a BOUND variable is irrelevant.

- Meaning of variable based on the \( \lambda \) that introduces it ... and how it is used in \( \lambda \)'s body.

- If we change the identifier used in a formal parameter and all of its bound occurrences, the meaning of the expression is unaltered.

\[
(\lambda x \mid x) \equiv (\lambda y \mid y)
\]
Variable Names in $\lambda$-calculus

- The identifier used to represent a BOUND variable is irrelevant.

- Meaning of variable based on the $\lambda$ that introduces it ... and how it is used in $\lambda$'s body.

- If we change the identifier used in a formal parameter and all of its bound occurrences, the meaning of the expression is unaltered.

\[(\lambda x \mid x) \equiv (\lambda y \mid y) \quad \text{YES!}\]
Variable Names in $\lambda$-calculus

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

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

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

$$(\lambda x \mid (\lambda y \mid x \ y)) \equiv (\lambda a \mid (\lambda b \mid a \ b)) \quad \text{YES!}$$

$$(\lambda x \mid (\lambda y \mid x \ y)) \not\equiv (\lambda a \mid (\lambda b \mid b \ a))$$
Variable Names in $\lambda$-calculus

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

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

$$(\lambda x \mid x) \equiv (\lambda x \mid y) \quad \text{No!}$$
Variable Names in $\lambda$-calculus

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

\[
(\lambda x \mid x) \equiv (\lambda y \mid y) \quad \text{YES!}
\]
\[
(\lambda x \mid x) \not\equiv (\lambda x \mid y) \quad \text{No!}
\]
\[
(\lambda x \mid (\lambda y \mid x \ y)) \not\equiv (\lambda a \mid (\lambda b \mid a \ b))
\]
Variable Names in $\lambda$-calculus

- The identifier used to represent a BOUND variable is irrelevant.

- Meaning of variable based on the $\lambda$ that introduces it ... and how it is used in $\lambda$’s body.

- If we change the identifier used in a formal parameter and all of its bound occurrences, the meaning of the expression is unaltered.

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

$$(\lambda x \mid x) \not\equiv (\lambda x \mid y) \quad \text{No!}$$

$$(\lambda x \mid (\lambda y \mid x \ y)) \equiv (\lambda a \mid (\lambda b \mid a \ b)) \quad \text{YES!}$$
Variable Names in $\lambda$-calculus

- The identifier used to represent a BOUND variable is irrelevant.

- Meaning of variable based on the $\lambda$ that introduces it ... and how it is used in $\lambda$’s body.

- If we change the identifier used in a formal parameter and all of its bound occurrences, the meaning of the expression is unaltered.

\[
(\lambda x \mid x) \not\equiv (\lambda y \mid y) \quad \text{YES!}
\]
\[
(\lambda x \mid x) \not\equiv (\lambda x \mid y) \quad \text{No!}
\]
\[
(\lambda x \mid (\lambda y \mid x \ y)) \equiv (\lambda a \mid (\lambda b \mid a \ b)) \quad \text{YES!}
\]
\[
(\lambda x \mid (\lambda y \mid x \ y)) \not\equiv (\lambda a \mid (\lambda b \mid b \ a))
\]
Variable Names in $\lambda$-calculus

- The identifier used to represent a BOUND variable is irrelevant

- Meaning of variable based on the $\lambda$ that introduces it ... and how it is used in $\lambda$’s body

- If we change the identifier used in a formal parameter and all of its bound occurrences, 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 $\lambda$-calculus

$$(\lambda x \mid (\lambda y \mid x \ y)) \equiv (\lambda y \mid (\lambda x \mid y \ x))$$
Variables in λ-calculus

\[(\lambda x \mid (\lambda y \mid x \ y)) \equiv (\lambda y \mid (\lambda x \mid y \ x))\]  YES!
Variables in $\lambda$-calculus

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

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

Think...
Variables in $\lambda$-calculus

\[
(\lambda x \ | \ (\lambda y \ | \ x \ y)) \ ? \ (\lambda y \ | \ (\lambda x \ | \ y \ x)) \quad \text{YES!}
\]

\[
(\lambda x \ | \ (\lambda y \ | \ x \ y)) \ ? \ (\lambda y \ | \ (\lambda x \ | \ x \ y)) \quad \text{NO!}
\]
Variables in \( \lambda \)-calculus

\[
(\lambda x \mid (\lambda y \mid x \ y)) \equiv (\lambda y \mid (\lambda x \mid y \ x)) \quad \text{YES!}
\]

\[
(\lambda x \mid (\lambda y \mid x \ y)) \not\equiv (\lambda y \mid (\lambda x \mid x \ y)) \quad \text{NO!}
\]

\[
(\lambda x \mid (\lambda w \mid w) \ x) \equiv (\lambda y \mid (\lambda w \mid w) \ y)
\]
Variables in $\lambda$-calculus

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

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

$$(\lambda x \mid (\lambda w \mid w) \ x) \equiv (\lambda y \mid (\lambda w \mid w) \ y) \quad \text{YES!}$$
Variables in $\lambda$-calculus

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

$$(\lambda x \mid (\lambda y \mid x y)) \not\equiv (\lambda y \mid (\lambda x \mid x y)) \quad \text{NO!}$$

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

$$(\lambda x \mid (\lambda y \mid y) x) \not\equiv (\lambda y \mid (\lambda y \mid y) y)$$
Variables in $\lambda$-calculus

\[
(\lambda x \mid (\lambda y \mid x \ y)) \equiv (\lambda y \mid (\lambda x \mid y \ x)) \quad \text{YES!}
\]

\[
(\lambda x \mid (\lambda y \mid x \ y)) \equiv (\lambda y \mid (\lambda x \mid x \ y)) \quad \text{NO!}
\]

\[
(\lambda x \mid (\lambda w \mid w) \ x) \equiv (\lambda y \mid (\lambda w \mid w) \ y) \quad \text{YES!}
\]

\[
(\lambda x \mid (\lambda y \mid y) \ x) \equiv (\lambda y \mid (\lambda y \mid y) \ y) \quad \text{YES! (same as above!)} \ldots \text{but confusing!}
\]

Think ...\((\lambda y \mid (\lambda y \mid y) \ y)\)
\( \alpha \) (Alpha Rule): Motivation

- The \( \beta \)-rule cannot be applied in ... 
  \[
  ( (\lambda y \mid (\lambda z \mid yz)) \ z ) \\
  \xrightarrow{\beta} [z / y] (\lambda z \mid yz) \\
  \not\equiv (\lambda z \mid zz) \quad \text{Why not?}
  \]
$\alpha$ (Alpha Rule): Motivation

- The $\beta$-rule cannot be applied in ...

\[
( (\lambda y \mid (\lambda z \mid yz)) \ z )
\]

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

\[
\not\equiv (\lambda z \mid zz) \quad \text{Why not?}
\]

$z$ was free in $z$ but \textit{bound} in the result $\Rightarrow$ substitution is illegal!
α (Alpha Rule): Motivation

- The β-rule cannot be applied in ...

\[
( (\lambda y \ | \ (\lambda z \ | \ yz)) \ z )
\]

\[
\beta \rightarrow [z / y] (\lambda z \ | \ yz)
\]

\[
\not\equiv (\lambda z \ | \ zz)
\]

Why not?

\( z \) was free in \( z \) but \textit{bound} in the result \( \Rightarrow \) substitution is illegal!

- \[
(\lambda y (\lambda z \ | \ yz)) (\lambda x | xz)
\]

\[
\beta \rightarrow [ (\lambda x | xz) / y ] (\lambda z \ | \ yz)
\]

\[
\not\equiv (\lambda z | (\lambda x | xz) z)
\]
α (Alpha Rule): Motivation

- The β-rule cannot be applied in ...
  \[
  ( (\lambda y \ | \ (\lambda z \ | \ yz)) \ z )
  \]
  \[\xrightarrow{\beta} [z / y] (\lambda z \ | \ yz) \]
  \[\not\equiv (\lambda z \ | \ zz) \quad \text{Why not?}\]
  \(z\) was free in \(z\) but \textit{bound} in the result ⇒ 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
\( \alpha \) (Alpha Rule): Motivation

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

- But, variable identifiers in and of themselves are irrelevant

- The \( \alpha \)-rule changes variable identifiers without altering meaning
\[ \alpha \text{ (Alpha Rule): Renaming} \]

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

\[ \text{Example:} \quad \alpha: \quad \lambda z . q \rightarrow \lambda y . q \]

- Note: Replace the formal parameter and every instance of \( z \) with \( q \)

- Note: \( q \) is a new variable, never used...
\( \alpha \) (Alpha Rule): Renaming

- \( \alpha \)-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
\(\alpha\) (Alpha Rule): Renaming

- \(\alpha\)-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

\[(\lambda z\ |\ yz) \xrightarrow{\alpha:q/z} (\lambda q\ |\ yq)\]
\( \alpha \) (Alpha Rule): Renaming

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

- Just choose an identifier \textit{not} used in the current expression, ... and substitution guaranteed to be legal

\[(\lambda z \mid yz)^{\alpha : q/z} \rightarrow (\lambda q \mid yq)\]

- Note: Replace the formal parameter \textit{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

\[(\lambda z \mid yz)^{\alpha:q/z} \rightarrow (\lambda q \mid yq)\]

- Note: Replace the formal parameter *and* EVERY instance of \( z \) with \( q \)

- Note: \( q \) is a NEW variable, never used ...
\( \alpha \) (Alpha Rule): Examples

\[
(\lambda a \mid b (\lambda c \mid c a) d) \xrightarrow{\alpha: z/a} (\lambda z \mid b (\lambda c \mid c z) d)
\]
\( \alpha \) (Alpha Rule): Examples

\[
(\lambda a \mid b (\lambda c \mid c a) \ d) \xrightarrow{\alpha:z/a} (\lambda z \mid b (\lambda c \mid c z) \ d)
\]

Legal
\( \alpha \) (Alpha Rule): Examples

\[
(\lambda a \mid b (\lambda c \mid c a) \ d) \xrightarrow{\alpha : z / a} (\lambda z \mid b (\lambda c \mid c z) \ d)
\]

Legal

\[
(\lambda x \mid (\lambda y \mid x y z)) \xrightarrow{\alpha : y / z} (\lambda x \mid (\lambda y \mid x y y))
\]
\( \alpha \) (Alpha Rule): Examples

\[
(\lambda a \mid b (\lambda c \mid c a) \ d) \xrightarrow{\alpha : z/a} (\lambda z \mid b (\lambda c \mid c z) \ d)
\]
Legal

\[
(\lambda x \mid (\lambda y \mid x y z)) \xrightarrow{\alpha : y/z} (\lambda x \mid (\lambda y \mid x y y))
\]
Illegal
Formal definition of $\alpha$

Let $\langle E \rangle$ and $\langle F \rangle$ be $\lambda$-calculus expressions; $x$ and $y$ be distinct $\lambda$-calculus constants.
Formal definition of $\alpha$

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

\[
[\langle E \rangle / x] (\lambda y \mid \langle F \rangle) \rightarrow (\lambda z \mid [\langle E \rangle / x] [z/y] \langle F \rangle )
\]
Using $\alpha$ and $\beta$ Together I

\[(\lambda y (\lambda z | yz)) (\lambda x | xz)\]
Using $\alpha$ and $\beta$ Together I

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

- We could use $\beta$ rule to simulate applying the function

$$\beta \rightarrow[(\lambda x \ xz)/y] \ (\lambda z \ yz)$$
Using $\alpha$ and $\beta$ Together 1

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

- We could use $\beta$ rule to simulate applying the function

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

- Legal substitution?
Using $\alpha$ and $\beta$ Together I

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

- We could use $\beta$ rule to simulate applying the function

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

- Legal substitution? No. Free $z$ in $(\lambda x \mid xz)$ becomes bound
Using $\alpha$ and $\beta$ Together I

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

- We could use $\beta$ rule to simulate applying the function
  $$\beta \rightarrow [(\lambda x | xz)/y] (\lambda z | yz)$$

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

- Use $\alpha$ rule to rename variable
  $$\alpha \rightarrow [(\lambda x | xz)/y] [q/z] (\lambda z | yz)$$
Using $\alpha$ and $\beta$ Together I

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

- We could use $\beta$ rule to simulate applying the function

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

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

- Use $\alpha$ rule to rename variable

  $$\frac{\alpha}{\rightarrow [ (\lambda x | xz) / y ] [q/z] \ (\lambda z | yz) \equiv [ (\lambda x | xz) / y ] (\lambda q | yq)}$$
Using $\alpha$ and $\beta$ Together II

- Now we can apply $\beta$-substitution

$$\left[ \frac{\lambda x \cdot xz}{y} \right] (\lambda q \cdot yq)$$
Now we can apply β-substitution

\[ \left( \lambda x | xz \right) / \ y \ \left( \lambda q | yq \right) \equiv \left( \lambda q | \left( \lambda x | xz \right) q \right) \]
Using \( \alpha \) and \( \beta \) Together II

- Now we can apply \( \beta \)-substitution

\[
\left[ \frac{\lambda x \, | \, xz}{y} \right] (\lambda q \, | \, yq) \equiv (\lambda q \, | \, (\lambda x \, | \, xz) \, q)
\]

\[(\lambda q \, | \, (\lambda x \, | \, xz) \, q)\]
Using $\alpha$ and $\beta$ Together II

Now we can apply $\beta$-substitution

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

$$( \lambda q | ( \lambda x | xz ) q )$$

$$\xrightarrow{\beta} ( \lambda q | [ q / x ] xz )$$
Now we can apply $\beta$-substitution

$$\boxed{\left[ \frac{\lambda x \mid xz}{y} \right] (\lambda q \mid yq) \equiv (\lambda q \mid (\lambda x \mid xz) \ q)}$$

$$(\lambda q \mid (\lambda x \mid xz) \ q)$$

$$\beta \rightarrow (\lambda q \mid \left[ q \ / x \right] \ xz) \equiv (\lambda q \mid q \ z)$$
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!
$\alpha$ and $\beta$ 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 $\lambda$ calculus expression!!
  - Turing Equivalents!

- Computation $\equiv$ Apply $\alpha$, $\beta$ 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 ≡Apply α, β rules (many times!) to reduce given expression to “unreducible” form

- Interpret value of resulting expression as result of computation
κ and θ 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 ≡Apply κ, θ rules (many times!) to reduce given expression to “unreducible” form

- Interpret value of resulting expression as result of computation

- Computation requires only two rules
Special case of the $\beta$-rule: $(\lambda x | \langle E \rangle) v \xrightarrow{\beta} [v/x] \langle E \rangle$
\( \eta \) (Eta Rule): Null Application

- Special case of the \( \beta \)-rule: 
  \[ (\lambda x | \langle E \rangle) \nu \xrightarrow{\beta} [\nu/x] \langle E \rangle \]

- Accelerates Rule 5 of \( \beta \) substitution
η (Eta Rule): Null Application

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

- Special case of the \( \beta \)-rule: \( (\lambda x | \langle E \rangle) \nu \xrightarrow{\beta} [\nu/x] \langle E \rangle \)

- Accelerates Rule 5 of \( \beta \) substitution

- If \( x \) does not appear as a free variable in \( \langle E \rangle \), then \( \langle E \rangle \) doesn’t change

- \( \eta \)-rule:
  - If \( x \) is not free in \( \langle E \rangle \) then \( ((\lambda x | \langle E \rangle) \nu) \xrightarrow{\eta} \langle E \rangle \)
η (Eta Rule): Null Application

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

- Accelerates Rule 5 of $\beta$ substitution

- If $x$ does not appear as a free variable in $\langle E \rangle$, then $\langle E \rangle$ doesn’t change

- $\eta$-rule:
  - If $x$ is not free in $\langle E \rangle$ then $(\lambda x | \langle E \rangle) v \xrightarrow{\eta} \langle E \rangle$

\[(\lambda a | c d) q \rightarrow \]
η (Eta Rule): Null Application

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

- $\eta$-rule:
  - If $x$ is not free in $\langle E \rangle$ then $((\lambda x | \langle E \rangle) \ nu) \xrightarrow{\eta} \langle E \rangle$
  - $((\lambda a | c d) \ q) \rightarrow (c d)$
\( \eta \) (Eta Rule): Null Application

- Special case of the \( \beta \)-rule: \((\lambda x \mid \langle E \rangle) \nu \xrightarrow{\beta} [\nu/x] \langle E \rangle\)

- Accelerates Rule 5 of \( \beta \) substitution

- If \( x \) does not appear as a free variable in \( \langle E \rangle \), then \( \langle E \rangle \) doesn’t change

- \( \eta \)-rule:
  - If \( x \) is not free in \( \langle E \rangle \) then \(((\lambda x \mid \langle E \rangle) \nu) \xrightarrow{\eta} \langle E \rangle\)

\[
((\lambda a \mid c \ d) q) \rightarrow (c \ d)
\]

\[
(\lambda x \mid (\lambda x \mid x \ y)) \nu \rightarrow
\]
η (Eta Rule): Null Application

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

- Accelerates Rule 5 of $\beta$ substitution

- If $x$ does not appear as a free variable in $\langle E \rangle$, then $\langle E \rangle$ doesn’t change

- $\eta$-rule:
  - If $x$ is not free in $\langle E \rangle$ then $((\lambda x | \langle E \rangle) v) \xrightarrow{\eta} \langle E \rangle$
  
  $((\lambda a | c d) q) \rightarrow (c d)$
  
  $(\lambda x | (\lambda x | x y)) v \rightarrow (\lambda x | x y)$
To implement a \( \lambda \)-calculus interpreter

- Faster to determine the status of variable \( x \) in \( \langle E \rangle \), than to build a new expression without any changes

\( \eta \)-rule
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
To implement a $\lambda$-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 $\langle E \rangle$, than to “build” a new expression without any changes
  $\Rightarrow \eta$-rule
On Reductions

- $\lambda$-calculus reduces expressions to “simpler” expressions using $\beta$ and $\eta$ rules
  - Why scare quotes?

- An expression that can be reduced is called a redux
- Can only reduce applications that contain function definitions
  - Cannot reduce $f$, $(f \ g)$, $(\lambda f \ (f \ g))$
  - Can reduce $( (\lambda x \ ((w x)) \ y))$

- An expression containing no reduxes is in normal form (i.e. a completed calculation)
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

- \( \lambda \)-calculus reduces expressions to “simpler” expressions using \( \beta \) and \( \eta \) rules
  - Why scare quotes?

- \( \beta \)-rule and \( \eta \)-rule are called reductions (\( \alpha \) is not a reduction)

- If we can obtain \( \langle N \rangle \) from \( \langle M \rangle \) using a sequence of \( \beta \) and \( \eta \) operations, then \( \langle M \rangle \) is reducible to \( \langle N \rangle \)
On Reductions

- \(\lambda\)-calculus reduces expressions to “simpler” expressions using \(\beta\) and \(\eta\) rules
  - Why scare quotes?

- \(\beta\)-rule and \(\eta\)-rule are called reductions (\(\alpha\) is not a reduction)

- If we can obtain \(\langle N \rangle\) from \(\langle M \rangle\) using a sequence of \(\beta\) and \(\eta\) operations, then \(\langle M \rangle\) is reducible to \(\langle N \rangle\)

- 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 ⟨N⟩ from ⟨M⟩ using a sequence 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 \( f, (f \ g), (\lambda f \mid (f \ g)) \)
  - Can reduce \( ( (\lambda x \mid (w \ x)) \ y) \)
On Reductions

- $\lambda$-calculus reduces expressions to “simpler” expressions using $\beta$ and $\eta$ rules
  - Why scare quotes?

- $\beta$-rule and $\eta$-rule are called reductions ($\alpha$ is not a reduction)

- If we can obtain $\langle N \rangle$ from $\langle M \rangle$ using a sequence of $\beta$ and $\eta$ 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 $f, (f \ g), (\lambda f \mid (f \ g))$
  - Can reduce $( (\lambda x \mid (w \ x)) \ y)$

- An expression containing no reduxes is in normal form
  (i.e. a completed calculation)
Theoretical Questions

- So “interpretation” ≡ ”reducing to normal form”
Theoretical Questions

▶ So “interpretation” ≡”reducing to normal form”

▶ Questions...
  ▶ Is there more than one way to reduce an expression?
  ▶ Is there one unique reduction for every expression?
  ▶ Is every expression reducible?
  ▶ If not, what are the implications?
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?

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

$$\text{leftmost}$$
Order of Reductions: Normal I

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

\[
\begin{align*}
& (\lambda x (\lambda y \mid x)) ((\lambda u \mid z) u) \\
& (\lambda x (\lambda y \mid x)) ((\lambda u \mid z) u) \\
& \underline{\text{leftmost}} \\
& (\lambda x (\lambda y \mid x)) ((\lambda u \mid z) u)
\end{align*}
\]
Order of Reductions: Normal I

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

\[
\begin{align*}
(\lambda x | (\lambda y | x )) & \ ((\lambda u | z ) \ u ) \\
(\lambda x | (\lambda y | x )) & \ ((\lambda u | z ) \ u ) \\
\text{leftmost} & \ \\
(\lambda x | (\lambda y | x )) & \ ((\lambda u | z ) \ u ) \\
\beta & \left[ ((\lambda u | z ) \ u ) \ / \ x \right] \ (\lambda y | x )
\end{align*}
\]
Order of Reductions: Normal I

- Normative Order: leftmost application first

- Which is leftmost function?

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

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

$$\overset{\text{leftmost}}{\left(\lambda x \left(\lambda y | x\right)\right) \left(\left(\lambda u | z\right) u\right)}$$

$$\overset{\beta}{\rightarrow} \left[\left(\left(\lambda u | z\right) u \right) / x\right] \left(\lambda y | x\right)$$

Free vars in $$\left(\left(\lambda u | z\right) u \right)$$ get bound? No
Order of Reductions: Normal I

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

\[
(\lambda x | (\lambda y | x)) \ ((\lambda u | z) \ u) \\
(\lambda x | (\lambda y | x)) \ ((\lambda u | z) \ u)
\]

\[
\text{leftmost} \\
(\lambda x | (\lambda y | x)) \ ((\lambda u | z) \ u)
\]

\[
\beta \rightarrow [((\lambda u | z) \ u) / x] \ (\lambda y | x)
\]

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?**

\[
(\lambda x | (\lambda y | x)) \ ( (\lambda u | z) \ u ) \\
(\lambda x | (\lambda y | x)) \ ( (\lambda u | z) \ u )
\]

leftmost

\[
(\lambda x | (\lambda y | x)) \ ( (\lambda u | z) \ u )
\]

\[
\beta \rightarrow [((\lambda u | z) \ u ) / x] \ (\lambda y | x)
\]

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

\[
\equiv (\lambda y | ((\lambda u | z) \ u ))
\]
Order of Reductions: Normal II

\( (\lambda y \mid ((\lambda u \mid z) u)) \) Left application?
Order of Reductions: Normal II

\[(\lambda y \mid ((\lambda u \mid z) u))\]  Left application?

\[(\lambda y \mid ((\lambda u \mid z) u)\) _leftmost_

\[\rightarrow (\lambda y \mid \left[\frac{u}{u}\right] z)\]

Any free vars in u get bound? No.

\[\rightarrow (\lambda y \mid z)\]

Done? Yes - normal form.
Order of Reductions: Normal II

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

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

\[\text{leftmost}\]

\((\lambda y \mid ((\lambda \underline{u} \mid z) u))\)
Order of Reductions: Normal II

\[(\lambda y \mid ((\lambda u \mid z) u)) \quad \text{Left application?}\]

\[(\lambda y \mid ((\lambda u|z) u))\]

\(\text{leftmost}\)\n
\[(\lambda y \mid ((\lambda u \mid z) u))\]

\[\beta \rightarrow (\lambda y \mid [u / u] z)\]
Order of Reductions: Normal II

\[ (\lambda y \ | \ ((\lambda u \ | \ z) \ u)) \]  
Left application?

\[ (\lambda y \ | \ ((\lambda u | z) \ u)) \]

leftmost

\[ (\lambda y \ | \ ((\lambda u \ | \ z) \ u)) \]

\[ \beta \rightarrow (\lambda y \ | \ [u / u] \ z) \]

Any free vars in u get bound? No.
Order of Reductions: Normal II

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

leftmost
\((\lambda y \mid (((\lambda u \mid z) \ u) \ ))\)
\(\beta \rightarrow (\lambda y \mid [u / u ] \ z \ )\)
Any free vars in u get bound? No.
\(\rightarrow\)
Order of Reductions: Normal II

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

leftmost

\((\lambda y \mid ((\lambda \underline{u} | z) \ u))\)
\(\beta\rightarrow (\lambda y \mid [u / u] \ z)\)
Any free vars in u get bound? No.
\(\rightarrow (\lambda y \mid z)\)
Order of Reductions: Normal II

\((\lambda y \ | \ ((\lambda u \ | \ z) \ u))\)  \hspace{1em} \text{Left application?}

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

\text{leftmost}

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

\[\beta \rightarrow (\lambda y \ | \ [u / u] \ z)\]

Any free vars in u get bound? No.

\[\rightarrow (\lambda y \ | \ z)\]

Done?
Order of Reductions: Normal II

\[(\lambda y \ | \ ((\lambda u \ | \ z) \ u ))\]  Left application?
\[(\lambda y \ | \ ((\lambda u | z) \ u ))\]

leftmost

\[(\lambda y \ | \ ((\lambda u | z) \ u ))\]

\[\beta \rightarrow (\lambda y \ | \ [u / u ] \ z)\]

Any free vars in \(u\) get bound? No.

\[\rightarrow (\lambda y \ | \ z)\]

Done? Yes - normal form
Order of Reductions: Applicative I

- Applicative Order: innermost \textit{application} first
Order of Reductions: Applicative I

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

\[ (\lambda x | (\lambda y | x)) ( (\lambda u | z) u ) \]
Applicative Order: innermost \textit{application} first

Like LISP: evaluate arguments first, then apply function

\[(\lambda x \, (\lambda y \, |x\rangle) \, ((\lambda u \, | z\rangle) \, u) \]
Order of Reductions: Applicative I

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

\[
(\lambda x((\lambda y \; x)) \; ((\lambda u \; z) \; u)) \\
(\lambda x((\lambda y \; x)) \; (\lambda u \; z) \; u)
\]
innermost
Applicative Order: innermost application first

Like LISP: evaluate arguments first, then apply function

\[(\lambda x | (\lambda y \ | x)) \ ( (\lambda u \ | z) \ u )\]

\[(\lambda x | (\lambda y \ | x)) \ ( (\lambda u | z) \ u )\]

innermost

\[(\lambda x | (\lambda y \ | x)) \ ( (\lambda u \ | z) \ u )\]
Order of Reductions: Applicative I

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

\[\lambda x \left( \lambda y \ | x \right) \left( \lambda u \ | z \right) u \]
\[\left( \lambda x \left( \lambda y \ | x \right) \left( \lambda u \mid z \right) u \right)\]

innermost

\[\left( \lambda x \left( \lambda y \mid x \right) \left( \lambda u \mid z \right) u \right)\]

\[\beta \rightarrow \left( \lambda x \left( \lambda y \ | x \right) \right) \left[ u / u \right] \left( \lambda u \mid z \right)\]
Order of Reductions: Applicative I

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

\[(\lambda x \,(\lambda y \, |x)) \ ((\lambda u \ | z) \ u)\]
\[(\lambda x \,(\lambda y \, |x)) \ (\ (\lambda u \,|z) \ u)\] innermost
\[(\lambda x \,(\lambda y \, |x)) \ ((\lambda u \ | z) \ u)\] 
\[\beta \rightarrow (\lambda x \,(\lambda y \, |x)) \ [u / u ] \ (\lambda u \ | z)\]
any free vars in \(u\) get bound?
Order of Reductions: Applicative I

- Applicative Order: innermost *application* first

- Like LISP: evaluate arguments first, then apply function

\[
(\lambda x \,(\lambda y \, x)) \, ((\lambda u \, z) \, u) \\
(\lambda x \,(\lambda y \, x)) \, (\lambda u \, z) \, u
\]

innermost

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

any free vars in \(u\) get bound? No.

\[\equiv\]
Order of Reductions: Applicative I

- Applicative Order: innermost *application* first

- Like LISP: evaluate arguments first, then apply function

\[
(\lambda x | (\lambda y | x)) \ ((\lambda u | z) \ u ) \\
(\lambda x | (\lambda y | x)) \ ( (\lambda u | z) \ u ) \\
\text{innermost} \\
(\lambda x | (\lambda y | x)) \ ((\lambda | u z) \ u ) \\
\beta \rightarrow (\lambda x | (\lambda y | x)) \ [u / u ] \ (\lambda u | z) \\
\text{any free vars in } u \text{ get bound?No.} \\
\equiv (\lambda x | (\lambda y | x)) \ z
\]
Order of Reductions: Applicative I

- Applicative Order: innermost *application* first

- Like LISP: evaluate arguments first, then apply function

\[
(\lambda x \cdot (\lambda y \cdot x)) \ ((\lambda u \cdot z) \ u) \\
(\lambda x \cdot (\lambda y \cdot x)) \ (\lambda u \cdot z) \ u
\]

innermost

\[
\beta \rightarrow (\lambda x \cdot (\lambda y \cdot x)) \ [u / u] \ (\lambda u \cdot z)
\]

any free vars in u get bound? No.

\[
\equiv (\lambda x \cdot (\lambda y \cdot x)) \ z
\]

Done?
Order of Reductions: Applicative I

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

\[
\lambda x(\lambda y \, x) \, ((\lambda u \, z) \, u)
\]
\[
\lambda x(\lambda y \, x) \, (\lambda u \, z) \, u
\]
\[
\beta \rightarrow (\lambda x(\lambda y \, x)) \, [(u \, u) \, (\lambda u \, z)]
\]

any free vars in u get bound? No.

\[
\equiv (\lambda x(\lambda y \, x)) \, z
\]

Done? Nope
Order of Reductions: Applicative II

\[(\lambda x \,(\lambda y \, x)) \, z \quad \text{Innermost?}\]
Order of Reductions: Applicative II

\[ (\lambda x \, (\lambda y \, |x)) \, z \quad \text{Innermost?} \]
\[ (\lambda x \, (\lambda y \, |x)) \, z \]

innermost
Order of Reductions: Applicative II

\[(\lambda x | (\lambda y | x)) \ z\]  Innermost?

\[(\lambda x | (\lambda y | x)) \ z\]

innermost

\[(\lambda x | (\lambda y | x)) \ z\]
Order of Reductions: Applicative II

\[
(\lambda x | (\lambda y \mid x)) \ z \quad \text{Innermost?}
\]
\[
(\lambda x | (\lambda y \mid x)) \ z
\]
\[
\text{innermost}
\]
\[
(\lambda x | (\lambda y \mid x)) \ z
\]
\[
\beta \rightarrow [z / x] \ (\lambda y \mid x)
\]
Order of Reductions: Applicative II

\[(\lambda x | (\lambda y | x)) \ z\]  Innermost?

\[(\lambda x | (\lambda y | x)) \ z\]  innermost

\[(\lambda x | (\lambda y | x)) \ z\]

\[\beta \rightarrow [z / x] \ (\lambda y | x)\]

Any free vars in \(x\) get bound? No
Order of Reductions: Applicative II

\[(\lambda x | (\lambda y | x)) \ z\]  Innermost?
\[(\lambda x | (\lambda y | x)) \ z\]

innermost
\[(\lambda x | (\lambda y | x)) \ z\]
\[\beta\]
\[\rightarrow [z / x] \ (\lambda y | x)\]
Any free vars in \(x\) get bound? No
\[\equiv\]
Order of Reductions: Applicative II

\[
(\lambda x \,(\lambda y \, x)) \, z \quad \text{Innermost?}
\]

\[
(\lambda x \,(\lambda y \, x)) \, z
\]

innermost

\[
(\lambda x \,(\lambda y \, x)) \, z
\]

\[
\beta \rightarrow [z \, / \, x] \quad (\lambda y \, x)
\]

Any free vars in \( x \) get bound? No

\[
\equiv (\lambda y \, z)
\]
Order of Reductions: Applicative II

\[(\lambda x \,(\lambda y \, x)) \, z\]  Innermost?
\[(\lambda x \,(\lambda y \, x)) \, z\]

innermost
\[(\lambda x \,(\lambda y \, x)) \, z\]

\[\beta \rightarrow [z \;/ \, x] \, (\lambda y \, | x)\]

Any free vars in \( x \) get bound? No
\[\equiv (\lambda y \, | x)\]

Done?
Order of Reductions: Applicative II

\[(\lambda x | (\lambda y | x)) \; z\]  Innermost?
\[(\lambda x | (\lambda y | x)) \; z\]
innermost
\[(\lambda x | (\lambda y | x)) \; z\]

\[\beta\rightarrow [z / x] \; (\lambda y | x)\]

Any free vars in \(x\) get bound? No
\[\equiv (\lambda y | z)\]

Done? Yes - normal form
You may choose

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

However, since $\lambda$-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)

You 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)
Order of Reductions: Comment

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

▶ However, since \( \lambda \) 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 \( \langle A \rangle, \langle B \rangle, \langle C \rangle, \langle D \rangle \) be \( \lambda \)-calculus expressions and \( \xrightarrow{1}, \xrightarrow{2}, \xrightarrow{3} \) and \( \xrightarrow{4} \) 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 \( \lambda \)-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 $\lambda$-calculus expressions and $\rightarrow_1, \rightarrow_2, \rightarrow_3$ and $\rightarrow_4$ be reductions of zero or more steps.

- Church and Rosser Theorem I
  - If $\langle A \rangle \rightarrow_1 \langle B \rangle$ and $\langle A \rangle \rightarrow_2 \langle C \rangle$,
  - Then $\exists \langle D \rangle \rightarrow_3$ and $\rightarrow_4$ s.t. $\langle B \rangle \rightarrow_3 \langle D \rangle$ and $\langle C \rangle \rightarrow_4 \langle D \rangle$

- 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 $\langle A \rangle \xrightarrow{2} \langle C \rangle$. 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???
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

... does every reduction result in 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
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!**
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 \textit{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

\[ (\lambda x | x \ x) (\lambda x | x \ x) \rightarrow (\lambda x | x \ x) (\lambda x | x \ x) \]

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

\[
(\lambda x \mid x \ x \ x) \ (\lambda x \mid x \ x)
\]
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

\[
(\lambda x \mid x \ x) (\lambda x \mid x \ x) \\
(\lambda x \mid x \ x) (\lambda x \mid x \ x) \rightarrow
\]

Dr. B. Price and Dr. R. Greiner

CMPUT 325 : Lambda Calculus Basics
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

\[
(\lambda x | x \ x) \ (\lambda x | x \ x)
\]
\[
(\lambda x | x \ x) \ (\lambda x | x \ x)
\]
\[
\rightarrow (\lambda x | x \ x) \ (\lambda x | x \ x)
\]
Existence Theorem

- Church and Rosser Theorem II
  - If $\langle \text{A} \rangle \rightarrow \langle \text{B} \rangle$ and $\langle \text{B} \rangle$ is in normal form then $\langle \text{A} \rangle \rightarrow \langle \text{B} \rangle$ by normative order reduction

- If $\langle \text{A} \rangle$ can be reduced to a normal form, it can be found by normal order reduction

- Not every expression has a normal form

  \[
  (\lambda x \mid x \ x) \ (\lambda x \mid x \ x)
  \]

  \[
  (\lambda x \mid x \ x) \ (\lambda x \mid x \ x)
  \rightarrow (\lambda x \mid x \ x) \ (\lambda x \mid x \ x)
  \]

- Because reductions are not guaranteed to terminate, the equivalence of $\lambda$-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

\[
(\lambda x | x x) (\lambda x | x x)
\]
\[
(\lambda x | x x) (\lambda x | x x)
\]
\[
\rightarrow (\lambda x | x x) (\lambda x | x x)
\]

- Because reductions are not guaranteed to terminate,
  the equivalence of $\lambda$-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
  - ≈ 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
  - ≈ evaluating arguments before passing them
  - Can be interpreted as "call by value"

- Normative order reduction evaluates leftmost applications first
  - ≈ 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
  - ≈ evaluating arguments before passing them
  - Can be interpreted as "call by value"

- Normative order reduction evaluates leftmost applications first
  - ≈ passing unevaluated expressions to function
  - Can be interpreted as "call by name"
  - Passed-in expressions must still be evaluated in body of function
Completeness of Applicative vs. Normal Order

- The argument to $(\lambda x \mid y)$ does not matter
  - $(\langle \lambda x \mid y \rangle \langle E \rangle) \rightarrow y$ for any $\langle E \rangle$
  - Here, expression $\langle E \rangle$ is an *unneeded* argument
  - $\eta$-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
  - \(\eta\)-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 \textit{unneeded} argument
  - \(\eta\)-reductions

- Applicative order may evaluate \textit{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
  - \(\eta\)-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
  - \( \eta \)-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

\[ (\lambda z (\lambda y \mid y)) \ ( (\lambda x \mid x \ x) \ (\lambda x \mid x \ x) ) \]
\[ (\lambda z (\lambda y \mid y)) \ ( (\lambda x \mid x \ x) \ (\lambda x \mid x \ x) ) \]

leftmost application
Reducible by Normal Example

\[
(\lambda z (\lambda y \, y)) \ ( (\lambda x \mid x \, x) \ (\lambda x \mid x \, x) ) \\
(\lambda z (\lambda y \mid y)) \ ( (\lambda x \mid x \, x) \ (\lambda x \mid x \, x) )
\]

leftmost application

\[
(\lambda z (\lambda y \mid y)) \ ( (\lambda x \mid x \, x) \ (\lambda x \mid x \, x) )
\]
Reducible by Normal Example

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

leftmost application

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

$$\xrightarrow{\beta} \ [ \ (\ (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ )/ z] \ (\lambda z \ (\lambda y \ | \ y))$$
Reducible by Normal Example

\[(\lambda z \ (\lambda y \ | \ y)) \ ( (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ )\]
\[
\left(\lambda z \ (\lambda y \ | \ y)\right) \ ( (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ )
\]
leftmost application

\[(\lambda z \ (\lambda y \ | \ y)) \ ( (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ )\]
\[
\stackrel{\beta}{\rightarrow} \ [ \ ( (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ )/ \ z ] \ (\lambda z \ (\lambda y \ | \ y))
\]
Any free vars get bound?
Reducible by Normal Example

\[(\lambda z (\lambda y \mid y)) \ (\ (\lambda x \mid x \ x) 
\ (\lambda x \mid x \ x))\ \ (\lambda z \ (\lambda y \mid y))\ \ (\ (\lambda x \mid x \ x) 
\ (\lambda x \mid x \ x) )\ \]

leftmost application

\[(\lambda z (\lambda y \mid y)) \ (\ (\lambda x \mid x \ x) 
\ (\lambda x \mid x \ x))\ \beta\rightarrow \ [\ (\ (\lambda x \mid x \ x) 
\ (\lambda x \mid x \ x)) / z] \ (\lambda z \ (\lambda y \mid y))\]

Any free vars get bound? No.

≡
Reducible by Normal Example

\[(\lambda z (\lambda y \mid y)) ( (\lambda x \mid x x) (\lambda x \mid x x) )\]  \[\text{leftmost application}\]

\[(\lambda z (\lambda y \mid y)) ( (\lambda x \mid x x) (\lambda x \mid x x) )\]

\[\beta \rightarrow [ ( (\lambda x \mid x x) (\lambda x \mid x x) )/ z] (\lambda z (\lambda y \mid y))\]

Any free vars get bound? No.

\[\equiv (\lambda y \mid y)\]
Irreducible by Applicative Example

- Under Applicative order

\[
(\lambda z (\lambda y \mid y)) \ ( (\lambda x \mid x \ x) \ (\lambda x \mid x \ x) )
\]
Irreducible by Applicative Example

Under Applicative order

\[(\lambda z (\lambda y \mid y)) (\lambda x \mid x x) (\lambda x \mid x x)\]  
\[(\lambda z (\lambda y \mid y)) (\lambda x \mid x x) (\lambda x \mid x x)\]  
innermost application
Irreducible by Applicative Example

- Under Applicative order

\[
\begin{align*}
(\lambda z (\lambda y \mid y)) & \ ( (\lambda x \mid x\ x) \ (\lambda x \mid x\ x) ) \\
(\lambda z (\lambda y \mid y)) & \ ( (\lambda x \mid x\ x) \ (\lambda x \mid x\ x) )
\end{align*}
\]

innermost application

\[
(\lambda z (\lambda y \mid y)) \ ( (\lambda x \mid x\ x) \ (\lambda x \mid x\ x) )
\]
Irreducible by Applicative Example

- Under Applicative order

\[
(\lambda z (\lambda y \mid y)) ( (\lambda x \mid x \ x) (\lambda x \mid x \ x) ) \\
(\lambda z (\lambda y \mid y)) ( (\lambda x \mid x \ x) (\lambda x \mid x \ x) ) \\
\text{innermost application} \\
(\lambda z (\lambda y \mid y)) ( (\lambda x \mid x \ x) (\lambda x \mid x \ x) ) \\
\beta \rightarrow (\lambda z (\lambda y \mid y)) \ [ (\lambda x \mid x \ x) / x ] \ x \ x
\]
Irreducible by Applicative Example

- Under Applicative order

\[
(\lambda z \ (\lambda y \ y)) \ (\ (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ ) \\
(\lambda z \ (\lambda y \ y)) \ (\ (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ )
\]

innermost application

\[
(\lambda z \ (\lambda y \ y)) \ (\ (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ )
\]

\[
\beta \rightarrow (\lambda z \ (\lambda y \ y)) \ [\ (\lambda x \ | \ x \ x) \ / \ x \ ] \ x \ x
\]

Any free vars in (\lambda x \ | \ x \ x) get bound?
Irreducible by Applicative Example

Under Applicative order

\[
(\lambda z \ (\lambda y \ y) \ ) \ ( \ (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ ) \\
(\lambda z \ (\lambda y \ y) \ ) \ ( \ (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ ) \\
\text{innermost application} \\
(\lambda z \ (\lambda y \ y) \ ) \ ( \ (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ ) \\
\beta \rightarrow (\lambda z \ (\lambda y \ y) \ ) \ [ \ (\lambda x \ | \ x \ x) \ / \ x \ ] \ x \ x \\
\text{Any free vars in } (\lambda x \ | \ x \ x) \text{ get bound? No.} \\
\equiv
\]
**Irreducible by Applicative Example**

- **Under Applicative order**

  \[(\lambda z \ (\lambda y \ | \ y)) \ ( (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) ) \]

\[(\lambda z \ (\lambda y \ | \ y)) \ ( (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) ) \]

  \[\text{innermost application}\]

\[(\lambda z \ (\lambda y \ | \ y)) \ ( (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) ) \]

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

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

\[\equiv (\lambda z \ (\lambda y \ | \ y)) \ ( (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) ) \]
Irreducible by Applicative Example

- Under Applicative order

\[
(\lambda z (\lambda y \ y)) \ ( (\lambda x \ x x) \ (\lambda x \ x x) )
\]

innermost application

\[
(\lambda z (\lambda y \ y)) \ ( (\lambda x \ x x) \ (\lambda x \ x x) )
\]

\[
\beta \rightarrow (\lambda z (\lambda y \ y)) \ [ (\lambda x \ x x) / x ] \ x x
\]

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

\[
\equiv (\lambda z (\lambda y \ y)) \ ( (\lambda x \ x x) \ (\lambda x \ x x) )
\]

Notice anything fishy here?
Irreducible by Applicative Example

Under Applicative order

\[
(\lambda z \ (\lambda y \mid y)) \ (\ (\lambda x \mid x \ x) \ (\lambda x \mid x \ x) \ )
\]
\[
(\lambda z \ (\lambda y \mid y)) \ (\ (\lambda x \mid x \ x) \ (\lambda x \mid x \ x) \ )
\]

innermost application

\[
(\lambda z \ (\lambda y \mid y)) \ (\ (\lambda x \mid x \ x) \ (\lambda x \mid x \ x) \ )
\]

\[
\frac{\beta}{\rightarrow} (\lambda z \ (\lambda y \mid y)) \ [\ (\lambda x \mid x \ x) \ / x \ ] \ x \ x
\]

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

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

Notice anything fishy here?

We are back to what we started with!
Example 1: Normal Order

\[(\lambda x \mid (\lambda y \ x \mid x) \ z)) \ (\lambda x \mid x \ y)\]
Example 1: Normal Order

\[(\lambda x \mid (\lambda y \ x \mid x) \ z)) \ (\lambda x \mid x \ y)\]
First step?
Example 1: Normal Order

\[(\lambda x \mid (\lambda y \mid x \mid x) \mid z) \mid (\lambda x \mid x \mid y)\]

First step? Identify *leftmost* applicable function
Example 1: Normal Order

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

First step? Identify \textit{leftmost} applicable function

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

\underline{leftmost}
Example 1: Normal Order

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

First step? Identify *leftmost* applicable function

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

**leftmost**

\((\lambda x \mid (\lambda y \ x \mid x) \ z)) \ (\lambda x \mid x \ y)\)
Example 1 : Normal Order

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

First step? Identify *leftmost* applicable function

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

\underline{leftmost}

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

Recall \((\lambda y \mid x \mid x)\) means \((\lambda y \mid (\lambda x \mid x))\)
Example 1 : Normal Order

\[(\lambda x \mid (\lambda y \mid x) \mid x) \mid z)\) \(\lambda x \mid x \mid y)\]

First step? Identify *leftmost* applicable function

\[(\lambda x \mid (\lambda y \mid x) \mid x) \mid z)\) \(\lambda x \mid x \mid y)\]

leftmost

\[(\lambda x \mid (\lambda y \mid x) \mid x) \mid z)\) \(\lambda x \mid x \mid y)\]

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

\[(\lambda x \mid (\lambda y \mid (\lambda x \mid x) \mid z)))\) \(\lambda x \mid x \mid y)\]
Example 1: Normal Order

\[(\lambda x \mid (\lambda y \ x \mid x) \ z)) \ (\lambda x \mid x \ y)\]

First step? Identify \textit{leftmost} applicable function

\[(\lambda x \mid (\lambda y \ x \mid x) \ z)) \ (\lambda x \mid x \ y)\]

\textit{leftmost}

\[(\lambda x \mid (\lambda y \ x \mid x) \ z)) \ (\lambda x \mid x \ y)\]

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

\[(\lambda x \mid (\lambda y \mid (\lambda x \mid x) \ z))\)) \ (\lambda x \mid x \ y)\]

\[
\beta \rightarrow [(\lambda x \mid x \ y) / x] \ (\lambda y \mid (\lambda x \mid x) \ z)\]
Example 1: Normal Order

\[(\lambda x \mid (\lambda y \ x \mid x) \ z)) \ (\lambda x \mid x \ y)\]

First step? Identify \textit{leftmost} applicable function

\[(\lambda x \mid (\lambda y \ x \mid x) \ z)) \ (\lambda x \mid x \ y)\]

\underline{leftmost}

\[(\lambda x \mid (\lambda y \ x \mid x) \ z)) \ (\lambda x \mid x \ y)\]

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

\[(\lambda x \mid (\lambda y \mid (\lambda x \mid x) \ z))) \ (\lambda x \mid x \ y)\]

\[\beta \rightarrow [(\lambda x \mid x \ y) \ / \ x] \ (\lambda y \mid (\lambda x \mid x) \ z)\]

Free vars in \((\lambda x \mid x \ y)\)? get bound?
Example 1: Normal Order

\[(\lambda x \mid (\lambda y \ | \ x \ | \ x) \ z) \ (\lambda x \mid x \ y)\]

First step? Identify \textit{leftmost} applicable function

\[(\lambda x \mid (\lambda y \ | \ x) \ z) \ (\lambda x \mid x \ y)\]

Recall \((\lambda y \ | \ x)\) means \((\lambda y \mid (\lambda x \mid x))\)

\[(\lambda x \mid (\lambda y \mid (\lambda x \mid x) \ z)) \ (\lambda x \mid x \ y)\]

\[\beta \rightarrow [(\lambda x \mid x \ y) \ / \ x] \ (\lambda y \mid (\lambda x \mid x) \ z)\]

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

No free instances of \(x\) within \((\lambda y \mid (\lambda x \mid x) \ z)\)
Example 1: Normal Order

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

First step? Identify *leftmost* applicable function

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

leftmost

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

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

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

\[\beta\rightarrow\left[\frac{(\lambda x \mid x \ y)}{x}\right] (\lambda y \mid (\lambda x \mid x) \ z)\]

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

No free instances of \(x\) within \((\lambda y \mid (\lambda x \mid x) \ z)\)

\[\equiv (\lambda y \mid (\lambda x \mid x) \ z)\]
Example 1: Normal Order

\[(\lambda x \mid (\lambda y \mid x \mid x) \mid z)) \mid (\lambda x \mid x \mid y)\]

First step? Identify *leftmost* applicable function

\[(\lambda x \mid (\lambda y \mid x \mid x) \mid z)) \mid (\lambda x \mid x \mid y)\]

\[\text{leftmost}\]

\[(\lambda x \mid (\lambda y \mid x \mid x) \mid z)) \mid (\lambda x \mid x \mid y)\]

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

\[(\lambda x \mid (\lambda y \mid (\lambda x \mid x) \mid z))) \mid (\lambda x \mid x \mid y)\]

\[\beta\]

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

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

No free instances of \(x\) within \((\lambda y \mid (\lambda x \mid x) \mid z))\)

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

\[(\lambda y \mid (\lambda x \mid x) \mid z)\]
Example 1: Normal Order

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

First step? Identify leftmost applicable function

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

leftmost

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

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

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

\(\beta\) → \([ (\lambda x \mid x \ y) \ / \ x ] \ (\lambda y \mid (\lambda x \mid x) \mid z)\)

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

No free instances of \(x\) within \((\lambda y \mid (\lambda x \mid x) \mid z)\)

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

\((\lambda y \mid (\lambda x \mid x) \mid z)\) \(\eta\) → \((\lambda x \mid x)\)
Example 1 : Applicative

\[(\lambda x \mid (\lambda y \mid x) \mid x) \ z) \ (\lambda x \mid x \ y)\]
Example 1: Applicative

\[(\lambda x \; | \; (\lambda y \; x \; | \; x) \; z) \; (\lambda x \; | \; x \; y)\]

First step?
Example 1: Applicative

\[(\lambda x \ | \ (\lambda y \ x \ | \ x) \ z) \ (\lambda x \ | \ x \ y)\]

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

\[(\lambda x \mid (\lambda y \ x \mid x) \ z) \ (\lambda x \mid x \ y)\]

*innermost*
Example 1: Applicative

\[(\lambda x \mid (\lambda y \ x \mid x) \ z) \ (\lambda x \mid x \ y)\]

First step? Identify *innermost* applicable function

\[(\lambda x \mid (\lambda y \ x \mid x) \ z) \ (\lambda x \mid x \ y)\]

innermost

Recall: \((\lambda y \ x \mid x)\) means \((\lambda y \mid (\lambda x \mid x))\)
Example 1: Applicative

\[(\lambda x \mid (\lambda y \ x \mid x) \ z) \ (\lambda x \mid x \ y)\]

First step? Identify *innermost* applicable function

\[(\lambda x \mid (\lambda y \ x \mid x)\ z) \ (\lambda x \mid x \ y)\]

innermost

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

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

First step? Identify *innermost* applicable function

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

innermost

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

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

\(\beta \rightarrow (\lambda x \mid [z / y] \mid (\lambda x \mid x) \mid (\lambda x \mid x \mid y))\)
Example 1: Applicative

\[(\lambda x \mid (\lambda y \mid x \mid x) \mid x) \mid z)\] \[(\lambda x \mid x \mid y)\]

First step? Identify innermost applicable function

\[(\lambda x \mid (\lambda y \mid x \mid x) \mid z)\] \[(\lambda x \mid x \mid y)\]

innermost

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

\[(\lambda x \mid (\lambda y \mid (\lambda x \mid x)) \mid z)\] \[(\lambda x \mid x \mid y)\]

\[\beta \rightarrow (\lambda x \mid [z / y] \mid (\lambda x \mid x) \mid (\lambda x \mid x \mid y))\]

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

\[(\lambda x \mid (\lambda y \ x \mid x) \ z) \ (\lambda x \mid x \ y)\]

\textit{innermost}

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

\[(\lambda x \mid (\lambda y \ (\lambda x \mid x)) \ z) \ (\lambda x \mid x \ y)\]

\[\beta \rightarrow (\lambda x \mid [z / y ](\lambda x \mid x) \ (\lambda x \mid x \ y)\]

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

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

First step? Identify innermost applicable function

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

innermost

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

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

\(\beta\) → \((\lambda x \mid [z / y ] \ (\lambda x \mid x) \ (\lambda x \mid x \ y)\)

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

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

\((\lambda x \mid (\lambda x \mid x)) \ (\lambda x \mid x \ y)\)
Example 1: Applicative

\[(\lambda x \mid (\lambda y \mid x) \mid x) \mid z) \ (\lambda x \mid x \mid y)\]

First step? Identify nearest applicable function

\[(\lambda x \mid (\lambda y \mid x) \mid x) \mid z) \ (\lambda x \mid x \mid y)\]

innermost

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

\[(\lambda x \mid (\lambda y \mid (\lambda x \mid x)) \mid z) \ (\lambda x \mid x \mid y)\]

\[\beta \rightarrow (\lambda x \mid [z / y \mid ] \ (\lambda x \mid x) \ (\lambda x \mid x \mid y)\]

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

\[\equiv (\lambda x \mid (\lambda x \mid x)) \ (\lambda x \mid x \mid y)\]

\[(\lambda x \mid (\lambda x \mid x)) \ (\lambda x \mid x \mid y)\]

\[\eta \rightarrow (\lambda x \mid x)\]
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

\[\underbrace{(\lambda x \mid (\lambda y \mid x))((\lambda x \mid y)x)}_{\text{leftmost}}\]
Example 2: Normal Order I

\[(\lambda x \ | \ (\lambda y \ | \ x)) \ ((\lambda x \ | \ y) \ x)\]

First task: find leftmost applicable function

\[
(\lambda x \ | \ (\lambda y \ | \ x)) \ ((\lambda x \ | \ y) \ x) \]

leftmost

\[
(\lambda x \ | \ (\lambda y \ | \ x)) \ ((\lambda x \ | \ y) \ x)
\]
Example 2: Normal Order I

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

First task: find leftmost applicable function

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

leftmost

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

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

$$\beta \rightarrow [((\lambda x \mid y) \ x) / x] \ (\lambda y \mid x)$$
Example 2: Normal Order I

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

First task: find leftmost applicable function

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

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

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

\[(\lambda x \mid (\lambda y \mid x)) \ ((\lambda x \mid y) \ x)\]

leftmost

\[(\lambda x \mid (\lambda y \mid x)) \ ((\lambda x \mid y) \ x)\]

\[\beta \rightarrow [((\lambda x \mid y) \ x) / x] \ (\lambda y \mid x)\]

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

\[\not\rightarrow (\lambda y \mid ((\lambda x \mid y) \ x))\]
Example 2: Normal Order I

\[(\lambda x \mid (\lambda y \mid x)) \ ((\lambda x \mid y) \ x)\]

First task: find leftmost applicable function
\[(\lambda x \mid (\lambda y \mid x)) \ ((\lambda x \mid y) \ x)\]

leftmost

\[(\lambda x \mid (\lambda y \mid x)) \ ((\lambda x \mid y) \ x)\]

\[\beta \rightarrow [((\lambda x \mid y) \ x) \ / \ x] \ (\lambda y \mid x)\]

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

\[\not\rightarrow (\lambda y \mid ((\lambda x \mid y) \ x))\]

Use \(\alpha\) rule.
Example 2: Normal Order I

\[(\lambda x \mid (\lambda y \mid x)) (\lambda x \mid y) x\]

First task: find leftmost applicable function

\[(\lambda x \mid (\lambda y \mid x)) (\lambda x \mid y) x\]

leftmost

\[(\lambda x \mid (\lambda y \mid x)) (\lambda x \mid y) x\]

\[\beta \rightarrow [((\lambda x \mid y) x) / x] (\lambda y \mid x)\]

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

\[\not\rightarrow (\lambda y \mid ((\lambda x \mid y) x))\]

Use \(\alpha\) rule.

\[\alpha \rightarrow [(\lambda x \mid y) x) / x] [z/y] (\lambda y \mid x)\]
Example 2: Normal Order I

\[(\lambda x \mid (\lambda y \mid x)) \ ((\lambda x \mid y) \ x)\]

First task: find leftmost applicable function

\[(\lambda x \mid (\lambda y \mid x)) \ ((\lambda x \mid y) \ x)\]

leftmost

\[(\lambda x \mid (\lambda y \mid x)) \ ((\lambda x \mid y) \ x)\]

\[\beta \rightarrow [((\lambda x \mid y) \ x) / x] \ (\lambda y \mid x)\]

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

\[\not\rightarrow (\lambda y \mid ((\lambda x \mid y) \ x))\]

Use \(\alpha\) rule.

\[\alpha \rightarrow [((\lambda x \mid y) \ x) / x] \ [z/y] \ (\lambda y \mid x)\]

\[\equiv [((\lambda x \mid y) \ x) / x] \ (\lambda z \mid x)\]
Example 2: Normal Order I

\[(\lambda x \mid (\lambda y \mid x)) \ (\ (\lambda x \mid y) \ x)\]

First task: find leftmost applicable function

\[\underbrace{\ (\lambda x \mid (\lambda y \mid x)) \ (\ (\lambda x \mid y) \ x)}\]

leftmost

\[(\lambda x \mid (\lambda y \mid x)) \ (\ (\lambda x \mid y) \ x)\]

\[\beta \rightarrow [((\lambda x \mid y) \ x) \ / \ x] \ (\lambda y \mid x)\]

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

\[\not\beta (\lambda y \mid ((\lambda x \mid y) \ x))\]

Use \(\alpha\) rule.

\[\alpha \rightarrow [((\lambda x \mid y) \ x) \ / \ x] \ [z/y] \ (\lambda y \mid x)\]

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

\[(\lambda z \mid ((\lambda x \mid y) \ x))\]
Example 2: Normal Order II

\((\lambda z \mid (\lambda x \mid y) x)\)
Example 2: Normal Order II

\[
(\lambda z \ | \ (\lambda x \ | \ y) \ x )
\]

leftmost

\[
(\lambda z \ | \ (\lambda x | y) \ x)
\]
Example 2: Normal Order II

\[(\lambda z \mid (\lambda x \mid y) \; x)\]
\[(\lambda z \mid (\lambda x \mid y) \; x)\]
\[\underbrace{\text{leftmost}}\]
\[(\lambda z \mid (\lambda x \mid y) \; x)\]
Example 2: Normal Order II

\[(\lambda z \mid (\lambda x \mid y) \ x)\]
\[(\lambda z \mid (\lambda x \mid y) \ x)\]
\[
\underbrace{\text{leftmost}}(\lambda z \mid ((\lambda x \mid y) \ x))
\]
\[\eta \rightarrow (\lambda z \mid y)\]
Example 2: Applicative I

\[(\lambda x \mid (\lambda y \mid x)) ((\lambda x \mid y) x)\]
Example 2: Applicative I

\[(\lambda x \mid (\lambda y \mid x)) \((\lambda x \mid y) \ x)\]

First step?
Example 2: Applicative I

\[(\lambda x \ | \ (\lambda y \ | \ x)) \ ((\lambda x \ | \ y) \ x)\]

First step? Find innermost application
Example 2: Applicative I

\[(\lambda x \ | \ (\lambda y \ | \ x)) \ ((\lambda x \ | \ y) \ x)\]

First step? Find innermost application

\[(\lambda x \ | \ (\lambda y \ | \ x)) \underbrace{((\lambda x \ | \ y) \ x)}_{\text{innermost}}\]
Example 2: Applicative I

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

First step? Find innermost application

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

innermost

$$(\lambda x \mid (\lambda y \mid x)) \ ((\lambda x \mid y) \ x)$$
Example 2: Applicative I

\[(\lambda x \mid (\lambda y \mid x)) \ ((\lambda x \mid y) \ x)\]

First step? Find innermost application
\[(\lambda x \mid (\lambda y \mid x)) \ ((\lambda x \mid y) \ x)\]

\[\overset{\text{innermost}}{\rightarrow} \ (\lambda x \mid (\lambda y \mid x)) \ y\]
Example 2: Applicative II

\[(\lambda x \mid (\lambda y \mid x)) \, y\]
Example 2: Applicative II

\[(\lambda x \mid (\lambda y \mid x)) \ y\]
\[(\lambda x\mid(\lambda y\mid x)) \ y\]
\[\text{innermost}\]
Example 2: Applicative II

\[(\lambda x \mid (\lambda y \mid x)) \, y\]
\[\underbrace{(\lambda x \mid (\lambda y \mid x)) \, y}_\text{innermost}\]
\[\xrightarrow{\beta} [y/x] \, (\lambda y \mid x)\]
Example 2: Applicative II

\[(\lambda x \mid (\lambda y \mid x)) \ y\]
\[(\lambda x \mid (\lambda y \mid x)) \ y\]
innermost
\[\beta\]
\[\rightarrow [y/x] \ (\lambda y \mid x)\]

Free vars get bound?
Example 2: Applicative II

\[(\lambda x \mid (\lambda y \mid x)) \ y\]
\[(\lambda x \mid (\lambda y \mid x)) \ y\]
\text{innermost}
\[\beta\]
\[\rightarrow [y/x] \ (\lambda y \mid x)\]
Free vars get bound? Yes
Example 2: Applicative II

\[(\lambda x \mid (\lambda y \mid x))\ y\]
\[(\lambda x\mid(\lambda y\mid x))\ y\]
\[\text{innermost}\]
\[\beta\to[y/x] (\lambda y \mid x)\]
Free vars get bound? Yes
\[\alpha\to[y/x][z/y](\lambda y \mid x)\]
Example 2: Applicative II

\((\lambda x \mid (\lambda y \mid x)) \ y\)
\[\underbrace{\left(\lambda x \mid (\lambda y | x)\right)}_{\text{innermost}}\ y\]
\[\beta \rightarrow [y/x] (\lambda y \mid x)\]

Free vars get bound? Yes
\[\alpha \rightarrow [y/x][z/y](\lambda y \mid x)\]
\[\equiv [y/x](\lambda z \mid x)\]
Example 2: Applicative II

\[(\lambda x \mid (\lambda y \mid x)) \ y\]
\[\underbrace{(\lambda x \mid (\lambda y \mid x))} \ y\]
innermost
\[\beta \rightarrow [y/x] \ (\lambda y \mid x)\]

Free vars get bound? Yes
\[\alpha \rightarrow [y/x][z/y](\lambda y \mid x)\]
\[\equiv [y/x](\lambda z \mid x)\]
\[\equiv (\lambda z \mid y)\]
Example 3: Normal I

\[
\left( (\lambda x \; y \mid y) \; \left( (\lambda x \mid x \; x) \; (\lambda x \mid x \; x) \right) \right) \; a
\]
Example 3: Normal I

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

First step?
Example 3: Normal I

\[ ((\lambda x \ y \ | \ y) \ ((\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x))) \ a \]

First step? Find leftmost application
Example 3: Normal I

\[ ((\lambda x \ y \ y) ((\lambda x \ x \ x) (\lambda x \ x \ x))) \ a \]

First step? Find leftmost application

\[ ((\lambda x \ y \ y) ((\lambda x \ x \ x) (\lambda x \ x \ x))) \ a \]

leftmost
Example 3: Normal I

\[(\lambda x \ y \ | \ y) \ ((\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x)) \] a

First step? Find leftmost application

\[(\lambda x \ y \ | \ y) \ ((\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x)) \] a

Re-call:

\[(\lambda x \ y \ | \ y) \equiv (\lambda x \ | \ (\lambda y \ | \ y))\]
Example 3: Normal I

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

First step? Find leftmost application

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

leftmost

Re-

call: (\lambda x \ y \ | \ y) \equiv (\lambda x \ | (\lambda y \ | \ y))

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

leftmost
Example 3: Normal I

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

First step? Find leftmost application

\((\lambda x \ y \ | \ y) \ ((\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x))\) \ a \ Re-

leftmost

call: \((\lambda x \ y \ | \ y) \equiv (\lambda x \ | \ (\lambda y \ | \ y))\)

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

leftmost

\(\eta\) \((\lambda y \ | \ y)\) \ a
Example 3: Normal I

\[
((λx \ y \ | \ y) \ ((λx \ | \ x \ x) \ (λx \ | \ x \ x))) \ a
\]

First step? Find leftmost application

\[
(λx \ y \ | \ y) \ ((λx \ | \ x \ x) \ (λx \ | \ x \ x)) \ a
\]

leftmost

\[
(λx \ y \ | \ y) \equiv (λx \ | \ (λy \ | \ y))
\]

leftmost

\[
((λx \ |(λy \ | \ y)) \ ((λx \ | \ x \ x) \ (λx \ | \ x \ x))) \ a
\]

\eta

\[
(λy \ | \ y) \ a
\]

\beta

\[
[a/y] \ y \equiv \ a
\]
Example 3: Applicative I

\[(\lambda x \ y \ | \ y) \ ((\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x)) \ a\]
Example 3: Applicative I

\[(\lambda x\ y \ |\ y)\ ((\lambda x\ |\ x\ x)\ (\lambda x\ |\ x\ x))\ a\]

First step? Find innermost application.
Example 3: Applicative I

\[(\lambda x \ y \ | \ y) \ ((\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x)) \ a\]

First step? Find innermost application.

\[(\lambda x \ y \ | \ y) \ ((\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x)) \ a\]

innermost
Example 3: Applicative I

\[ (\lambda x \ y \ | \ y) \ ( (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x)) \ a \]

First step? Find innermost application.

\[ (\lambda x \ y \ | \ y) \ ( (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x)) \ a \]

innermost

\[ \xrightarrow{\beta} \ ((\lambda x \ y \ | \ y) \ [ (\lambda x \ | \ x \ x) / x] \ (x \ x)) \ a \]
Example 3: Applicative I

\[(\lambda x \ y \ | \ y) \ ((\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x)) \ a\]

First step? Find innermost application.

\[(\lambda x \ y \ | \ y) \ (\underbrace{\(\lambda x \ | \ x \ x\)}_{\text{innermost}}) \ (\lambda x \ | \ x \ x)) \ a\]

\[\beta \rightarrow ((\lambda x \ y \ | \ y) \ [ (\lambda x \ | \ x \ x)/ x] \ (x \ x)) \ a\]

Will free vars in get \((\lambda x \ | \ x \ x)\) bound?
Example 3: Applicative I

\[ (\lambda x \ y \ | \ y) \ (\ (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ ) \ a \]

First step? Find innermost application.

\[ (\lambda x \ y \ | \ y) \ (\underbrace{\ (\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x) \ ) \ a} \]

innermost

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

Will free vars in get \((\lambda x \ | \ x \ x)\) bound? No free vars!
Example 3: Applicative I

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

First step? Find innermost application.
\((\lambda x \ y \ | \ y) \ ((\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x)) \ a\)

innermost

\[\beta\] ((\lambda x \ y \ | \ y) \ [ (\lambda x \ | \ x \ x)/ x] \ (x \ x)) \ a

Will free vars in get (\(\lambda x \ | \ x \ x\)) bound? No free vars!
\[\equiv\] (\(\lambda x \ y \ | \ y\) ((\(\lambda x \ | \ x \ x\))(\(\lambda x \ | \ x \ x\))) \ a

\(\lambda\ x\ y\ \ |\ y\) ((\(\lambda x\ |\ x\ x\))(\(\lambda x\ |\ x\ x\))) \ a
Example 3: Applicative I

\[(\lambda x \ y \ | \ y) \ ((\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x)) \ a\]

First step? Find innermost application.

\[(\lambda x \ y \ | \ y) \ ((\lambda x \ | \ x \ x) \ (\lambda x \ | \ x \ x)) \ a\]

innermost

\[\beta \rightarrow ((\lambda x \ y \ | \ y) \ [\ (\lambda x \ | \ x \ x)/ \ x \ ] \ (x \ x)) \ a\]

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

\[\equiv (\lambda x \ y \ | \ y) \ ((\lambda x \ | \ x \ x)(\lambda x \ | \ x \ x)) \ a\]

We get the original expression back again!
Shortcuts for Multi-argument $\lambda$’s

$$(\lambda x \ y \ z \ | \ 〈E〉) \langle A \rangle \langle B \rangle \langle C \rangle$$
Shortcuts for Multi-argument λ’s

\[
(\lambda x \ y \ z \mid \langle E \rangle) \langle A \rangle \langle B \rangle \langle C \rangle \\
\equiv (\lambda x \mid (\lambda y \mid (\lambda z \mid \langle E \rangle)) \langle A \rangle \langle B \rangle \langle C \rangle)
\]
Shortcuts for Multi-argument λ’s

\[
(\lambda x \ y \ z \ | \ \langle E \rangle ) \ \langle A \rangle \ \langle B \rangle \ \langle C \rangle \\
\equiv (\lambda x \ | \ (\lambda y \ | \ (\lambda z \ | \ \langle E \rangle )) \ \langle A \rangle \ \langle B \rangle \ \langle C \rangle \\
\xrightarrow{\beta} [\langle A \rangle / x] \ (\lambda y \ | \ (\lambda z \ | \ \langle E \rangle )) \ \langle B \rangle \ \langle C \rangle
\]
Shortcuts for Multi-argument λ’s

\[
\left(\lambda x \ y \ z \mid \langle E \rangle \right) \langle A \rangle \langle B \rangle \langle C \rangle \\
\equiv \left(\lambda x \mid (\lambda y \mid (\lambda z \mid \langle E \rangle)) \langle A \rangle \langle B \rangle \langle C \rangle \right)
\]

\[
\beta \rightarrow \left[\langle A \rangle/x\right] \left(\lambda y \mid (\lambda z \mid \langle E \rangle) \langle B \rangle \langle C \rangle \right)
\]
If \( \langle A \rangle \) has free \( y \) or \( z \), must rename \( (\lambda y \mid (\lambda z \mid \langle E \rangle)) \)
Shortcuts for Multi-argument λ’s

\[(\lambda x \ y \ z \ | \ \langle E \rangle) \ \langle A \rangle \ \langle B \rangle \ \langle C \rangle\]
\[\equiv (\lambda x \ | \ (\lambda y \ | \ (\lambda z \ | \ \langle E \rangle)) \ \langle A \rangle \ \langle B \rangle \ \langle C \rangle)\]

\[\beta \rightarrow [\langle A \rangle/x] (\lambda y \ | \ (\lambda z \ | \ \langle E \rangle) \ \langle B \rangle \ \langle C \rangle)\]
If \(\langle A \rangle\) has free \(y\) or \(z\), must rename \((\lambda y \ | \ (\lambda z \ | \ \langle E \rangle)\)

\[\beta \rightarrow [\langle B \rangle/y] (\lambda z \ | \ \langle E \rangle)\]
Shortcuts for Multi-argument $\lambda$’s

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

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

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

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

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

If $\langle B \rangle$ has free $z$, must rename $(\lambda z\ |\ \langle E \rangle)$
Shortcuts for Multi-argument λ’s

\[(\lambda x \ y \ z \ | \ \langle E \rangle \ ) \ \langle A \rangle \ \langle B \rangle \ \langle C \rangle\]
\[\equiv (\lambda x \ | \ (\lambda y \ | \ (\lambda z \ | \ \langle E \rangle \ ) \ ) \ \langle A \rangle \ \langle B \rangle \ \langle C \rangle\]

\[\beta \rightarrow [\langle A \rangle/x] \ (\lambda y \ | \ (\lambda z \ | \ \langle E \rangle \ ) \ ) \ \langle B \rangle \ \langle C \rangle\]

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

\[\beta \rightarrow [\langle B \rangle/y] \ (\lambda z \ | \ \langle E \rangle \ )\]

If \(\langle B \rangle\) has free \(z\), must rename \((\lambda z \ | \ \langle E \rangle \ )\)

\[\beta \rightarrow [\langle C \rangle/z] \ \langle E \rangle\]
Shortcuts for Multi-argument λ’s

\[(\lambda x \ y \ z \ | \ ⟨E⟩) \ ⟨A⟩ \ ⟨B⟩ \ ⟨C⟩\]
\[\equiv (\lambda x \ | \ (\lambda y \ | \ (\lambda z \ | \ ⟨E⟩)) \ ⟨A⟩ \ ⟨B⟩ \ ⟨C⟩)\]

\[\beta \rightarrow [⟨A⟩/x] (\lambda y \ | \ (\lambda z \ | \ ⟨E⟩)) \ ⟨B⟩ \ ⟨C⟩\]
If \(⟨A⟩\) has free \(y\) or \(z\), must rename \((\lambda y \ | \ (\lambda z \ | \ ⟨E⟩))\)

\[\beta \rightarrow [⟨B⟩/y] (\lambda z \ | \ ⟨E⟩)\]
If \(⟨B⟩\) has free \(z\), must rename \((\lambda z \ | \ ⟨E⟩)\)

\[\beta \rightarrow [⟨C⟩/z] ⟨E⟩\]
If \(⟨C⟩\) has free var bound in \(⟨E⟩\), must rename ...
Example of Multi-argument $\lambda$’s

- Our basic solution method

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

Note: we replaced $y$ with $z$, but then immediately replaced $z$ with $\langle M \rangle$. 

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

Must rename $y$ in $(\lambda y \mid x y)$. Say $z$
Example of Multi-argument $\lambda$’s

- Our basic solution method

\[
(\lambda \, x \, y \mid \, x \, y) \, (\langle N \rangle \, y) \, \langle M \rangle \\
\equiv (\lambda \, x \mid (\lambda y \mid \, x \, y)) \, (\langle N \rangle \, y) \, \langle M \rangle
\]
Example of Multi-argument λ’s

- Our basic solution method

\[
(\lambda \, x \, y \mid x \, y) \, (\langle N \rangle \, y) \, \langle M \rangle \\
\equiv (\lambda \, x \mid (\lambda y \mid x \, y)) \, (\langle N \rangle \, y) \, \langle M \rangle \\
\beta \rightarrow [\, (\langle N \rangle \, y) \, / \, x \, ] \, (\lambda y \mid x \, y) \, \langle M \rangle
\]
Example of Multi-argument λ’s

- Our basic solution method

\[
(\lambda \ x \ y \ | \ x \ y) \ (\langle N \rangle \ y) \langle M \rangle \\
\equiv (\lambda \ x \ | \ (\lambda y \ | \ x \ y)) \ (\langle N \rangle \ y)\langle M \rangle \\
\beta \rightarrow [ (\langle N \rangle \ y) / x ] \ (\lambda y \ | \ x \ y) \langle M \rangle \\
\]
Free vars in (\langle N \rangle \ y) get bound?
Example of Multi-argument $\lambda$’s

- **Our basic solution method**

\[
(\lambda \ x \ y \ | \ x \ y) \ (\langle N \rangle \ y) \langle M \rangle \\
\equiv (\lambda \ x \ | \ (\lambda y \ | \ x \ y)) \ (\langle N \rangle \ y) \langle M \rangle \\
\beta \rightarrow \ [\ (\langle N \rangle \ y) / x \ ] \ (\lambda y \ | \ x \ y) \langle M \rangle \\
\text{Free vars in } (\langle N \rangle \ y) \text{ get bound? Yes!}
\]
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 \(z\)
Example of Multi-argument λ’s

- Our basic solution method

\[(\lambda x y \mid x y) \ (\langle N \rangle \ y) \langle M \rangle\]
\[\equiv (\lambda x \mid (\lambda y \mid x y)) \ (\langle N \rangle \ y)\langle M \rangle\]
\[\beta \rightarrow [ (\langle N \rangle \ y) / x ] \ (\lambda y \mid x y) \langle M \rangle\]

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

Must rename y in (\lambda y \mid x y). Say z
\[\alpha \rightarrow [ (\langle N \rangle \ y) / x ] [z/y] \ (\lambda y \mid x y) \langle M \rangle\]
Example of Multi-argument λ’s

- Our basic solution method

\[(\lambda \ x \ y \ | \ x \ y) \ (\langle N \rangle \ y) \langle M \rangle\]
\[\equiv (\lambda \ x \ | \ (\lambda y \ | \ x \ y)) \ (\langle N \rangle \ y)\langle M \rangle\]
\[\beta \rightarrow [ (\langle N \rangle \ y) / x ] \ (\lambda y \ | \ x \ y) \langle M \rangle\]

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

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

\[\alpha \rightarrow [ (\langle N \rangle \ y) / x ] [z/y] \ (\lambda y \ | \ x \ y) \langle M \rangle\]
\[\equiv [ (\langle N \rangle \ y) / x ] (\lambda z \ | \ x \ z) \langle M \rangle\]
Example of Multi-argument λ’s

- Our basic solution method

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

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

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

\[
\xrightarrow{\alpha} [ (\langle N \rangle \ y) \ / \ x ] [z/y] \ (\lambda y \ | \ x \ y) \langle M \rangle
\]
\[
\equiv [ (\langle N \rangle \ y) \ / \ x ] (\lambda z \ | \ x \ z) \langle M \rangle
\]
\[
\equiv (\lambda z \ | \ (\langle N \rangle \ y) \ z) \langle M \rangle
\]
Example of Multi-argument $\lambda$’s

- **Our basic solution method**

\[
(\lambda \ x \ y \ | \ x \ y) \ (\langle N \rangle \ y) \langle M \rangle \\
\equiv (\lambda \ x \ | (\lambda y \ | \ x \ y)) \ (\langle N \rangle \ y) \langle M \rangle \\
\beta \rightarrow [ (\langle N \rangle \ y) / x ] \ (\lambda y \ | \ x \ y) \langle M \rangle \\
\]

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

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

\[
\alpha \rightarrow [ (\langle N \rangle \ y) / x ] \ [z/y] \ (\lambda y \ | \ x \ y) \langle M \rangle \\
\equiv [ (\langle N \rangle \ y) / x ] \ (\lambda z \ | \ x \ z) \langle M \rangle \\
\equiv (\lambda z \ | (\langle N \rangle \ y) \ z) \langle M \rangle \\
\beta \rightarrow [\langle M \rangle / z] \ (\langle N \rangle \ y) \ z \equiv (\langle N \rangle \ y) \langle M \rangle 
\]
Example of Multi-argument λ’s

► Our basic solution method

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

► Note: we replaced \(y\) with \(z\),
but then immediately replace \(z\) with \(\langle M \rangle\)
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  \((λx | (λy \ldots )) \langle A \rangle \langle B \rangle,\)
  
  we do not have to check for free y’s in \langle A \rangle as \langle B \rangle will be substituted for the "(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 \((\lambda x \mid (\lambda y \ldots ) \langle A \rangle \langle B \rangle,\)
  
  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

  \((\lambda x y \mid x y) \langle N \rangle y \langle M \rangle\)
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 $\lambda$’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

\[
(\lambda x y \mid x y) \ (\langle N \rangle \ y) \langle M \rangle
\]

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

\[
\equiv (\langle N \rangle \ y) \langle M \rangle
\]
Example of Multi-argument $\lambda$’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

\[
(\lambda x \ y \mid x \ y) \ (\langle N \rangle \ y \rangle \langle M \rangle) \\
\overset{\beta}{\rightarrow} [(\langle N \rangle \ y) / x, \langle M \rangle / y] (x \ y) \\
\equiv (\langle N \rangle \ y \rangle \langle M \rangle)
\]

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

- Can represent n-ary functions as nested unary functions

\[(\lambda x \ y \ | \ E) \ a \ b \equiv (\lambda x \ (\lambda y \ E)) \ a \ b\]
Curried functions

▶ Can represent n-ary functions as nested unary functions

▶ \((\lambda x \ y \ | \ 〈E〉) \ a \ b\)

≡ \((\lambda x \ (\lambda y \ 〈E〉)) \ a \ b\)

▶ 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

\[(\lambda x \ y \ E) \ a \ b \equiv (\lambda x \ (\lambda y \ E)) \ a \ b\]

- Can treat an n-ary function as a unary function that returns an n-1-ary function

- Treating n-ary function as unary function that returns a function is called currying