The λ-calculus

Thanks to Petar Maksimović

Imperial College London

λ-CALCULUS: THE SIMPLEST PROGRAMMING LANGUAGE

$$M \ ::= \quad x \quad | \quad \lambda x.\, M \quad | \quad M\ M$$

Variable

Abstraction

(single-parameter function)

Application

λ-ABSTRACTION IN PROGRAMMING LANGUAGES

`\x -> 2*x + 1` (Haskell)

`x -> 2*x + 1` (Java)

`x => 2*x + 1` (JavaScript, C#, Scala)

`fun x -> 2*x + 1` (OCaml, F#)

`{x in 2*x+1}` (Swift)

$$\lambda x.(2x + 1)$$

`-> $x{2*$x+1}` (Perl)

`[](int x) { return 2*x+1; }` (C++)

`lambda x: 2*x + 1` (Python)

`^(int x) { return 2*x+1; }` (Objective-C)

λ-CALCULUS: THE SIMPLEST PROGRAMMING LANGUAGE

$$M ::= x \quad | \quad \lambda x.\, M \quad | \quad M\ M$$

Variable | Abstraction (single-parameter function) | Application

WHAT WILL WE COVER IN THESE LECTURES?

SYNTAX
Free/bound variables
α-equivalence
Substitution

SEMANTICS
β-reduction
Confluence/normal forms
Reduction strategies

APPLICATIONS
Expressivity
Arithmetic, Data structures
Recursion

λ-CALCULUS: THE SIMPLEST PROGRAMMING LANGUAGE

$$M ::= x \mid \lambda x.M \mid M\ M$$

Variable | Abstraction (single-parameter function) | Application

WHAT WILL WE COVER IN THESE LECTURES?

SYNTAX
Free/bound variables
α-equivalence
Substitution

SEMANTICS
β-reduction
Confluence/normal forms
Reduction strategies

APPLICATIONS
Expressivity
Arithmetic, Data structures
Recursion

FREE AND BOUND VARIABLES

$$M ::= x \mid \lambda x. M \mid M\ M$$

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

Bound variables are in **red**, free variables are in **light blue**

Free and Bound Variables

$$M ::= x \mid \lambda x. M \mid M\ M$$

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

Contraction

Bound variables are in **red**, free variables are in **light blue**

Free and Bound Variables

$$M ::= x \mid \lambda x. M \mid M\ M$$

$\lambda x. x$

$\lambda x. y$

$\lambda xyz. x\ y$

Closed term – no free variables

Bound variables are in **red**, free variables are in **light blue**

Free and Bound Variables

$$M ::= x \mid \lambda x.\, M \mid M\ M$$

$\lambda x.\, x$ $\lambda x.\, y$ $\lambda xyz.\, x\ y$

$$((\lambda x.\, x\ y)(\lambda y.\, x\ y))(\lambda xy.\, x\ y\ z)$$

Application is left-associative

Bound variables are in **red**, free variables are in **light blue**

Free and Bound Variables

$$M ::= x \mid \lambda x. M \mid M\ M$$

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

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

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

Bound variables are in **red**, free variables are in **light blue**

Free variables, formally

$$M \ ::= x \mid \lambda x.\, M \ \mid \ M\ M$$

$$\mathrm{FV}(x) = \{x\}$$
$$\mathrm{FV}(\lambda x.\, M) = \mathrm{FV}(M) \setminus \{x\}$$
$$\mathrm{FV}(M N) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$$

Example: $\mathrm{FV}\big((\lambda x.\,(\lambda y.\, x\ y) y)(\lambda z.\, z\ x)\big) = \{x, y\}$

Tutorial: Free and Bound Variables

1. (**Free and bound variables.**)

   (a) i. Circle all the binding occurrences of variables in this λ-term:

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

   ii. Circle all the bound occurrences of variables in this λ-term:

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

   iii. Circle all the free occurrences of variables in this λ-term:

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

   (b) Give the set of free variables for:

   i. $(\lambda x.\ xy)(x\lambda y.\ yx)(\lambda yz.\ zy)$

   ii. $(\lambda z.\ z(\lambda y.\ yzx)y)(\lambda xz.\ (\lambda\ y.\ zxy)x)$

Answer: Free and Bound Variables

(Free and bound variables.)

(a) i. The binding occurrences of variables in this λ-term are:

$$(\lambda\, x\, y\, .\; y\; (\lambda\, x\, .\; x\; y)\; z)(x\; (\lambda\, z\, x\, .\; x\; z\; y))$$

ii. The bound occurrences of variables in this λ-term are:

$$(\lambda\, x\; y\; .\; y\, (\lambda\, x\; .\; x\, y)\; z)(x\; (\lambda\; z\; x\; .\; x\, z\, y))$$

iii. The free occurrences of variables in this λ-term are:

$$(\lambda\, x\; y\; .\; y\; (\lambda\; x\; .\; x\; y)\; z)(x\, (\lambda\; z\; x\; .\; x\; z\; y))$$

(b) i. For $(\lambda x.\ xy)(x\lambda y.\ yx)(\lambda yz.\ zy)$: $FV = \{x, y\}$.
Notice that the x that is appearing twice in the second parentheses is free, as it is outside of the scope of the binding λx from the first parentheses.

ii. For $(\lambda z.\ z(\lambda y.\ yzx)y)(\lambda xz.\ (\lambda\ y.\ zxy)x)$: $FV = \{x, y\}$.
You might be tempted to think that the z in $\lambda y.\ yzx$ is free, but it is, in fact, bound to the outside λz.

RENAMING BOUND VARIABLES: α-EQUIVALENCE

In programming languages: Bound variables $\longrightarrow$ **function parameters**.

```
function (x, y) { return x + y; }
function (a, b) { return a + b; }
```

$\lambda xy.\, x\, y$

$\lambda ab.\, a\, b$

$$\lambda xy.\, x\, y \;=_\alpha\; \lambda ab.\, a\, b$$

INTUITION: $M =_\alpha N$ if and only if one can be obtained from the other by renaming the bound variables

HINT: If $M =_\alpha N$, then they must have **the same set of free variables**

Tutorial: α-Equivalence

2. (α**-Equivalence.**)

   (a) Which of the following λ-terms is α-equivalent to $(\lambda xy.\, y(\lambda x.\, xy)z)$?

   i. $(\lambda xy.\, a(\lambda x.\, xa)a)$
   ii. $(\lambda zy.\, y(\lambda x.\, xy)z)$
   iii. $(\lambda xy.\, y(\lambda z.\, zy)z)$
   iv. $(\lambda xy.\, y(\lambda z.\, zy)a)$
   v. $(\lambda xa.\, a(\lambda a.\, aa)z)$
   vi. $(\lambda xa.\, a(\lambda x.\, xa)a)$
   vii. $(\lambda xa.\, a(\lambda z.\, za)z)$
   viii. $(\lambda za.\, a(\lambda z.\, za)z)$

   (b) Write down three λ-terms which are α-equivalent to $(\lambda y.(\lambda x.\, xy)zxy)$.

   (c) For each of the three λ-terms you gave in part b, write down the set of free variables.

SOLUTION: α-EQUIVALENCE (PART)

(α-**Equivalence.**)

(a) Which of the following λ-terms is α-equivalent to $(\lambda xy.\, y(\lambda x.\, xy)z)$?

i. $(\lambda xy.\, a(\lambda x.\, xa)a)$ ✗

ii. $(\lambda zy.\, y(\lambda x.\, xy)z)$ ✗

iii. $(\lambda xy.\, y(\lambda z.\, zy)z)$ ✓

iv. $(\lambda xy.\, y(\lambda z.\, zy)a)$ ✗

v. $(\lambda xa.\, a(\lambda a.\, aa)z)$ ✗

vi. $(\lambda xa.\, a(\lambda x.\, xa)a)$ ✗

vii. $(\lambda xa.\, a(\lambda z.\, za)z)$ ✓

viii. $(\lambda za.\, a(\lambda z.\, za)z)$ ✗

α-EQUIVALENCE: EXAMPLES

STRATEGY: Are the terms of the same structure?
Do all of the free variables match?
Can you rename the bound variables so that they match?

$$\lambda x. x \;=_\alpha\; \lambda y. x\, y \quad ✗$$

$$\lambda x. x\, (\lambda z. x\, z) y \;=_\alpha\; \lambda y. y (\lambda t. y\, t) x \quad ✗$$

$$\lambda x. x\, (\lambda z. x\, z) y \;=_\alpha\; \lambda y. y (\lambda t. y\, t) y \quad ✗$$

$$\lambda x. x\, (\lambda z. x\, z) y \;=_\alpha\; \lambda w. w (\lambda t. w\, t) y \quad ✓$$

$$(\lambda xy. z\, x\, (\lambda t. t\, y)) x \;=_\alpha\; (\lambda yw. z\, y\, (\lambda z. z\, w)) x \quad ✓$$

SUBSTITUTION BY EXAMPLE

$M[N / x]$

$x\ [y / x]$	equals	y
$z\ [y / x]$		
$(x\ y)(y\ z)\ [y / x]$		
$(\lambda z.xz)\ [y / x]$		
$(\lambda x.xy)\ [y / x]$		
$(\lambda y.xy)\ [y / x]$		

Substitution by example

$$M[N / x]$$

$x\ [y / x]$	equals	y
$z\ [y / x]$	equals	z
$(x\ y)(y\ z)\ [y / x]$		
$(\lambda z. xz)\ [y / x]$		
$(\lambda x. xy)\ [y / x]$		
$(\lambda y. xy)\ [y / x]$		

No x to substitute for

SUBSTITUTION BY EXAMPLE

$M[N / x]$

$x\,[y / x]$	equals	y
$z\,[y / x]$	equals	z
$(x\ y)(y\ z)\,[y / x]$	equals	$(y\ y)(y\ z)$
$(\lambda z.xz)\,[y / x]$	equals	$\lambda z.yz$
$(\lambda x.xy)\,[y / x]$		
$(\lambda y.xy)\,[y / x]$		

Substitution by example

$M[N / x]$

$x\,[y / x]$	equals	y
$z\,[y / x]$	equals	z
$(x\ y)(y\ z)\,[y / x]$	equals	$(y\ y)(y\ z)$
$(\lambda z.\,xz)\,[y / x]$	equals	$\lambda z.\,yz$
$(\lambda x.\,xy)\,[y / x]$	equals	$\lambda y.\,yy$
$(\lambda y.\,xy)\,[y / x]$	equals	

Can't substitute for a bound variable!

SUBSTITUTION BY EXAMPLE

$M[N / x]$

$x\,[y / x]$	equals	y
$z\,[y / x]$	equals	z
$(x\ y)(y\ z)\,[y / x]$	equals	$(y\ y)(y\ z)$
$(\lambda z.xz)\,[y / x]$	equals	$\lambda z.yz$
$(\lambda x.xy)\,[y / x]$	equals	$\lambda x.xy$
$(\lambda y.xy)\,[y / x]$	equals	

SUBSTITUTION BY EXAMPLE

$$M[N / x]$$

$x\ [y / x]$	equals	y
$z\ [y / x]$	equals	z
$(x\ y)(y\ z)\ [y / x]$	equals	$(y\ y)(y\ z)$
$(\lambda z.xz)\ [y / x]$	equals	$\lambda z.yz$
$(\lambda x.xy)\ [y / x]$	equals	$\lambda x.xy$
$(\lambda y.xy)\ [y / x]$	equals	$\lambda y.yy$

Can't substitute and bind a variable!

SUBSTITUTION BY EXAMPLE $M[N / x]$

$x\,[y / x]$	equals	y
$z\,[y / x]$	equals	z
$(x\ y)(y\ z)\,[y / x]$	equals	$(y\ y)(y\ z)$
$(\lambda z.xz)\,[y / x]$	equals	$\lambda z.yz$
$(\lambda x.xy)\,[y / x]$	equals	$\lambda x.xy$
$(\lambda y.xy)\,[y / x]$	equals	$\lambda z.yz$

SUBSTITUTION, FORMALLY

$$M ::= x \mid \lambda x. M \mid M\ M$$

$$x[M/y] = \begin{cases} M & x = y \\ x & x \neq y \end{cases}$$

$$(\lambda x. N)[M/y] = \begin{cases} \lambda x. N & x = y \\ \lambda z. N[z/x][M/y] & x \neq y \end{cases}$$

where $z \notin (FV(N) \setminus \{x\}), z \notin FV(M), z \neq y$

$$(M_1\ M_2)[M/y] = (M_1[M/y])(M_2[M/y])$$

TUTORIAL: SUBSTITUTION

3. (**Expression substitution.**) Give the result of each of the following λ-term substitutions:

(a) $(xy)[z/x]$

(b) $(xy)[\lambda x.\ xx/x]$

(c) $(\lambda x.\ xy)[z/y]$

(d) $(\lambda x.\ xy)[z/x]$

(e) $(\lambda x.\ xy)[x/y]$

(f) $(\lambda x.\ xx)[\lambda x.\ xx/x]$

(g) $(\lambda x.\ xy)[\lambda x.\ xy/y]$

(h) $(\lambda x.\ xy)[x(\lambda x.\ xy)/y]$

ANSWER: SUBSTITUTION

(**Expression substitution.**)
The results of the substitutions are as follows:

(a) $(xy)[z/x] = zy$ — Simple substitution.

(b) $(xy)[\lambda x.\ xx/x] = (\lambda x.\ xx)y$ — Simple substitution.

(c) $(\lambda x.\ xy)[z/y] = \lambda x.\ xz$ — Substitution under binder, no capture threat.

(d) $(\lambda x.\ xy)[z/x] = \lambda x.\ xy$ — No free x to be substituted, no effect.

(e) $(\lambda x.\ xy)[x/y] = \lambda z.\ zx$ — Must rename binding x to avoid capture.

(f) $(\lambda x.\ xx)[\lambda x.\ xx/x] = \lambda x.\ xx$ — No free x to be substituted, no effect.

(g) $(\lambda x.\ xy)[\lambda x.\ xy/y] = \lambda x.\ x(\lambda x.\ xy)$ — Substitution under binder, no capture threat.

(h) $(\lambda x.\ xy)[x(\lambda x.\ xy)/y] = \lambda z.\ z(x(\lambda x.\ xy))$ — Must rename binding x to avoid capture.

λ-CALCULUS: THE SIMPLEST PROGRAMMING LANGUAGE

$$M ::= x \quad | \quad \lambda x.M \quad | \quad M\ M$$

Variable | Abstraction (single-parameter function) | Application

WHAT WILL WE COVER IN THESE LECTURES?

SYNTAX
Free/bound variables
α-equivalence
Substitution

SEMANTICS
β-reduction
Confluence/normal forms
Reduction strategies

APPLICATIONS
Expressivity
Arithmetic, Data structures
Recursion

COMPUTATION: β-REDUCTION

Redex

$$\frac{}{\underline{(\lambda x. M)N} \longrightarrow_\beta M[N/x]}$$

$$\frac{M \longrightarrow_\beta M'}{\lambda x. M \longrightarrow_\beta \lambda x. M'} \qquad \frac{M \longrightarrow_\beta M'}{MN \longrightarrow_\beta M'N} \qquad \frac{N \longrightarrow_\beta N'}{MN \longrightarrow_\beta MN'}$$

$$\frac{M =_\alpha M' \quad M' \longrightarrow_\beta N' \quad N' =_\alpha N}{M \longrightarrow_\beta N}$$

β-reduction: Example

$$(\lambda x.\, x\ x)\big((\lambda x.\, y)z\big)$$

β-reduction: Example

$$(\lambda x.\, x\ x)\big((\lambda x.\, y)z\big)$$

β-reduction: Example

$$(\lambda x.\, x\ x)\big((\lambda x.\, y)z\big)$$

$$\xrightarrow{\beta}$$

$$\big((\lambda x.\, y)z\big)\ \big((\lambda x.\, y)z\big)$$

β-reduction: Example

$$(\lambda x.\, x\ x)\big((\lambda x.\, y)z\big)$$

$$\xrightarrow{\beta}$$

$$\big((\lambda x.\, y)z\big)\,\big((\lambda x.\, y)z\big)$$

β-REDUCTION: EXAMPLE

$$(\lambda x.\, x\ x)\big((\lambda x.\, y)z\big)$$

$$\xrightarrow{\beta}$$

$$\big((\lambda x.\, y)z\big)\ \big((\lambda x.\, y)\underline{z}\big)$$

$$\xrightarrow{\beta}$$

$$\big((\lambda x.\, y)z\big)\ y$$

β-REDUCTION: EXAMPLE

$$(\lambda x.\, x\ x)\big((\lambda x.\, y)z\big)$$

$$\xrightarrow{\beta}$$

$$\big((\lambda x.\, y)z\big)\ \big((\lambda x.\, y)z\big)$$

$$\xrightarrow{\beta}$$

$$\big((\lambda x.\, y)\underline{z}\big)\ y$$

β-Reduction: Example

$$(\lambda x.\, x\ x)\big((\lambda x.\, y)z\big)$$

$$\xrightarrow{\beta}$$

$$\big((\lambda x.\, y)z\big)\ \big((\lambda x.\, y)z\big)$$

$$\xrightarrow{\beta}$$

$$\big((\lambda x.\, y)\underline{z}\big)\ y$$

$$\xrightarrow{\beta}$$

$$yy$$

β-reduction: Example

$$(\lambda x.\,x\;x)\big((\lambda x.\,y)z\big)$$

$$\xrightarrow{\beta} \big((\lambda x.\,y)\underline{z}\big)\;\big((\lambda x.\,y)z\big)$$

$$\xrightarrow{\beta} \big((\lambda x.\,y)z\big)\;y$$

$$\xrightarrow{\beta} yy$$

β-REDUCTION: EXAMPLE

$$
\begin{tikzcd}
 & & (\lambda x.\, x\, x)\big((\lambda x.\, y)z\big) \arrow[dl, "\beta"'] \\
 & \big((\lambda x.\, y)\underline{z}\big)\,\big((\lambda x.\, y)z\big) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
y\big((\lambda x.\, y)z\big) & & \big((\lambda x.\, y)z\big)\, y \arrow[d, "\beta"] \\
 & & yy
\end{tikzcd}
$$

β-REDUCTION: EXAMPLE

$$
\begin{tikzcd}
& & (\lambda x.\,x\ x)\big((\lambda x.\,y)z\big) \arrow[dl, "\beta"'] \\
& \big((\lambda x.\,y)z\big)\ \big((\lambda x.\,y)z\big) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
y\big((\lambda x.\,y)\underline{z}\big) & & \big((\lambda x.\,y)z\big)\ y \arrow[d, "\beta"] \\
& & yy
\end{tikzcd}
$$

β-REDUCTION: EXAMPLE

$$
\begin{tikzcd}
 & & (\lambda x.\, x\, x)\big((\lambda x.\, y)z\big) \arrow[dl, "\beta"'] \\
 & \big((\lambda x.\, y)z\big)\,\big((\lambda x.\, y)z\big) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
 y\big((\lambda x.\, y)z\big) \arrow[drr, "\beta"] & & \big((\lambda x.\, y)z\big)\, y \arrow[d, "\beta"] \\
 & & yy
\end{tikzcd}
$$

β-reduction: Example

$$
\begin{tikzcd}[column sep=small]
 & (\lambda x.\, x\ x)\big((\lambda x.\, y)\underline{z}\big) \arrow[d, "\beta"'] & \\
 & \big((\lambda x.\, y)z\big)\,\big((\lambda x.\, y)z\big) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
y\big((\lambda x.\, y)z\big) \arrow[drr, "\beta"] & & \big((\lambda x.\, y)z\big)\, y \arrow[d, "\beta"] \\
 & & yy
\end{tikzcd}
$$

β-REDUCTION: EXAMPLE

$$
\begin{tikzcd}
 & (\lambda x.\, x\, x)\big((\lambda x.\, y)\underline{z}\big) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
\big((\lambda x.\, y)z\big)\,\big((\lambda x.\, y)z\big) \arrow[d, "\beta"'] \arrow[dr, "\beta"] & & (\lambda x.\, x\, x)\; y \\
y\big((\lambda x.\, y)z\big) \arrow[dr, "\beta"'] & \big((\lambda x.\, y)z\big)\; y \arrow[d, "\beta"] & \\
 & yy &
\end{tikzcd}
$$

β-Reduction: Example

$$
\begin{tikzcd}[column sep=small]
 & & (\lambda x.\,x\;x)\big((\lambda x.\,y)z\big) \arrow[dl, "\beta"'] \arrow[ddr, "\beta"] & \\
 & \big((\lambda x.\,y)z\big)\,\big((\lambda x.\,y)z\big) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & & \\
y\big((\lambda x.\,y)z\big) \arrow[drr, "\beta"] & & \big((\lambda x.\,y)z\big)\,y \arrow[d, "\beta"] & (\lambda x.\,x\;x)\;y \\
 & & yy &
\end{tikzcd}
$$

β-REDUCTION: EXAMPLE

$$
\begin{tikzcd}
 & & (\lambda x.\, x\ x)\big((\lambda x.\, y)z\big) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
 & \big((\lambda x.\, y)z\big)\,\big((\lambda x.\, y)z\big) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & & (\lambda x.\, x\ x)\ \underline{y} \arrow[ddl, "\beta"'] \\
y\big((\lambda x.\, y)z\big) \arrow[drr, "\beta"] & & \big((\lambda x.\, y)z\big)\ y \arrow[d, "\beta"] & \\
 & & yy &
\end{tikzcd}
$$

β-reduction: Example

$$
\begin{tikzcd}
 & & (\lambda x.\, x\ x)\big((\lambda x.\, y)z\big) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
 & \big((\lambda x.\, y)z\big)\,\big((\lambda x.\, y)z\big) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & & (\lambda x.\, x\ x)\ y \arrow[ddl, "\beta"'] \\
y\big((\lambda x.\, y)z\big) \arrow[drr, "\beta"] & & \big((\lambda x.\, y)z\big)\ y \arrow[d, "\beta"] & \\
 & & yy &
\end{tikzcd}
$$

TUTORIAL: β-REDUCTION

4. (β**-reduction.**)

   (a) For each of the following λ-terms, perform a single β-reduction step and give the entire derivation three for this step.

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

   ii. $(\lambda x.\, \lambda y.\, xy)y$

   iii. $(\lambda x.\, \lambda y.\, xy)z$

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

   (b) Find distinct λ-terms M, N such that M and N are *not* α-equivalent and:

$$((\lambda x.\, x)(\lambda x.\, xx))((\lambda x.\, x)(\lambda x.\, xx)) \to M$$

$$((\lambda x.\, x)(\lambda x.\, xx))((\lambda x.\, x)(\lambda x.\, xx)) \to N$$

   (c) What happens if you continue reducing M and N?

   (d) Let $T \triangleq \lambda x.xxx$. Perform some β-reduction steps on TT. There is no need to give full derivation trees. What do you observe?

Solution: β-reduction (part)

iv.

$$\dfrac{\dfrac{\overline{(\lambda x.\, x)y \to y}}{x((\lambda x.\, x)y) \to xy}}{\lambda x.\, x((\lambda x.\, x)y) \to \lambda x.\, xy}$$

MULTI-STEP β-REDUCTION: $\longrightarrow_\beta^*$

Reflexive-transitive closure of β-reduction under α-conversion

Reflexivity, α-conversion: $$\frac{M =_\alpha M'}{M \longrightarrow_\beta^* M'}$$

Transitivity: $$\frac{M \longrightarrow_\beta M'' \quad M'' \longrightarrow_\beta^* M'}{M \longrightarrow_\beta^* M'}$$

Confluence

Theorem (Church-Rosser)

$\forall M, M_1, M_2.$

$M \longrightarrow_\beta^* M_1 \ \wedge \ M \longrightarrow_\beta^* M_2 \Longrightarrow$

$\exists M'.\ M_1 \longrightarrow_\beta^* M' \ \wedge \ M_2 \longrightarrow_\beta^* M'$

$$\begin{tikzcd}
 & M \arrow[dl] \arrow[dr] & \\
M_1 \arrow[dr, dashed] & & M_2 \arrow[dl, dashed] \\
 & M' &
\end{tikzcd}$$

β-NORMAL FORMS

λ-terms are in β-normal form if they contain no redexes.

$$\text{is\_in\_nf}(M) \stackrel{\text{def}}{=} \forall M'.\ M \not\rightarrow_\beta M'$$

$$\text{has\_nf}(M) \stackrel{\text{def}}{=} \exists M'.\ M \longrightarrow_\beta^* M' \wedge \text{is\_in\_nf}(M')$$

THEOREM (UNIQUENESS OF β-NORMAL FORMS)

$$\forall M, N_1, N_2.\ M \longrightarrow_\beta^* N_1 \wedge M \longrightarrow_\beta^* N_2 \wedge$$
$$\text{is\_in\_nf}(N_1) \wedge \text{is\_in\_nf}(N_2) \Longrightarrow N_1 =_\alpha N_2$$

β-NORMAL FORMS

THEOREM (UNIQUENESS OF β-NORMAL FORMS)

$$\forall M, N_1, N_2.\, M \longrightarrow_\beta^* N_1 \ \wedge \ M \longrightarrow_\beta^* N_2 \ \wedge$$
$$\text{is\_in\_nf}(N_1) \ \wedge \ \text{is\_in\_nf}(N_2) \Longrightarrow N_1 =_\alpha N_2$$

PROOF. From Church-Rosser, obtain N, such that $N_1 \longrightarrow_\beta^* N$ and $N_2 \longrightarrow_\beta^* N$. However, since N_1 and N_2 are in normal form, we have that $N_1 =_\alpha N =_\alpha N_2$.

TUTORIAL: β-NORMAL FORMS

5. (**β-normal-forms.**) For each of the following λ-terms, find its normal form, if it exists.

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

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

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

   (d) $(\lambda x.\ xx)(\lambda x.\ xx)$

   (e) $(\lambda x.\ xx)(\lambda x.\ x)$

   (f) $(\lambda x.\ x)(\lambda x.\ xx)$

Solution: β-normal forms

(β**-normal-forms.**)

(a) $(\lambda x.\ x)y$ has nf y: $(\lambda x.\ x)y \rightarrow_\beta y$, no further reduction possible (nfrp).

(b) $y(\lambda x.\ x)$ is already in nf, as no reductions can be done at all.

(c) $(\lambda x.\ x)(\lambda y.\ y)$ has nf $\lambda y.\ y$: $(\lambda x.\ x)(\lambda y.\ y) \rightarrow_\beta \lambda y.\ y$, nfrp.

(d) $(\lambda x.\ xx)(\lambda x.\ xx)$ has no normal form, as it keeps reducing to itself.

(e) $(\lambda x.\ xx)(\lambda x.\ x)$ has nf $(\lambda x.\ x)$: $(\lambda x.\ xx)(\lambda x.\ x) \rightarrow_\beta (\lambda x.\ x)(\lambda x.\ x) \rightarrow_\beta (\lambda x.\ x)$, nfrp.

(f) $(\lambda x.\ x)(\lambda x.\ xx)$ has nf $(\lambda x.\ xx)$: $(\lambda x.\ x)(\lambda x.\ xx) \rightarrow_\beta (\lambda x.\ xx)$, nfrp.

β-EQUIVALENCE: $=_\beta$

- Smallest equivalence relation containing $\longrightarrow_\beta$; or

- $\longrightarrow_\beta^*$ + symmetry; or

- $M_1 =_\beta M_2 \Leftrightarrow \exists M'.\ M_1 \longrightarrow_\beta^* M' \wedge M_2 \longrightarrow_\beta^* M'$

WHICH OF THESE SIX λ-TERMS ARE β-EQUIVALENT?

$$\begin{tikzcd}
 & & (\lambda x.\, x\ x)\big((\lambda x.\, y)z\big) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
 & \big((\lambda x.\, y)z\big)\,\big((\lambda x.\, y)z\big) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & & (\lambda x.\, x\ x)\ y \arrow[ddl, "\beta"'] \\
y\big((\lambda x.\, y)z\big) \arrow[drr, "\beta"] & & \big((\lambda x.\, y)z\big)\ y \arrow[d, "\beta"] & \\
 & & yy &
\end{tikzcd}$$

Any one to any other.

β-REDUCTION: NORMALISATION

$$(\lambda x.\, x\ x)(\lambda x.\, x\ x)$$

Must all λ-terms necessarily have a normal form?

No, they needn't.

β-REDUCTION: NORMALISATION

$$(\lambda x.\, x\ x)(\underline{\lambda x.\, x\ x})$$

Must all λ-terms necessarily have a normal form?

No, they needn't.

β-REDUCTION: NORMALISATION

$$(\lambda x.\, x\ x)(\lambda x.\, \underline{x\ x})$$

$$\xrightarrow{\beta}$$

$$(\lambda x.\, x\ x)(\lambda x.\, x\ x)$$

Must all λ-terms necessarily have a normal form?

No, they needn't.

β-REDUCTION: NORMALISATION

Must all λ-terms necessarily have a normal form?

No, they needn't.

$$(\lambda x.\, x\; x)(\lambda x.\, x\; x)$$

$$\xrightarrow{\beta}$$

$$(\lambda x.\, x\; x)(\underline{\lambda x.\, x\; x})$$

$$\xrightarrow{\beta}$$

$$(\lambda x.\, x\; x)(\lambda x.\, x\; x)$$

β-REDUCTION: NORMALISATION

Must all λ-terms necessarily have a normal form?

No, they needn't.

$$(\lambda x.\, x\ x)(\lambda x.\, x\ x) \xrightarrow{\beta} (\lambda x.\, x\ x)(\lambda x.\, x\ x) \xrightarrow{\beta} (\lambda x.\, x\ x)(\lambda x.\, x\ x) \xrightarrow{\beta}$$

…and so on…

DOES THE ORDER OF REDUCTION MATTER?

It matters...

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

Does the order of reduction matter?

It matters...

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

$$\xrightarrow{\beta} y$$

DOES THE ORDER OF REDUCTION MATTER?

It matters...

$$(\lambda x.\, y)\big((\lambda x.\, x\ x)(\underline{\lambda x.\, x\ x})\big)$$

$$y \xleftarrow{\beta} (\lambda x.\, y)\big((\lambda x.\, x\ x)(\lambda x.\, x\ x)\big) \xrightarrow{\beta} (\lambda x.\, y)\big((\lambda x.\, x\ x)(\lambda x.\, x\ x)\big)$$

DOES THE ORDER OF REDUCTION MATTER?

It matters...

$$(\lambda x.\, y)\big((\lambda x.\, x\ x)(\lambda x.\, x\ x)\big) \xrightarrow{\beta} y$$

$$(\lambda x.\, y)\big((\lambda x.\, x\ x)(\lambda x.\, x\ x)\big) \xrightarrow{\beta} (\lambda x.\, y)\big((\lambda x.\, x\ x)(\lambda x.\, x\ x)\big)$$

$$(\lambda x.\, y)\big((\lambda x.\, x\ x)(\lambda x.\, x\ x)\big) \xrightarrow{\beta} y$$

$$(\lambda x.\, y)\big((\lambda x.\, x\ x)(\lambda x.\, x\ x)\big) \xrightarrow{\beta} (\lambda x.\, y)\big((\lambda x.\, x\ x)(\lambda x.\, x\ x)\big)$$

DOES THE ORDER OF REDUCTION MATTER?

It matters...

$$(\lambda x. y)\big((\lambda x. x\ x)(\lambda x. x\ x)\big) \xrightarrow{\beta} y$$

$$(\lambda x. y)\big((\lambda x. x\ x)(\lambda x. x\ x)\big) \xrightarrow{\beta} (\lambda x. y)\big((\lambda x. x\ x)(\lambda x. x\ x)\big)$$

$$(\lambda x. y)\big((\lambda x. x\ x)(\lambda x. x\ x)\big) \xrightarrow{\beta} y$$

$$(\lambda x. y)\big((\lambda x. x\ x)(\lambda x. x\ x)\big) \xrightarrow{\beta} (\lambda x. y)\big((\lambda x. x\ x)(\lambda x. x\ x)\big) \xrightarrow{\beta}$$

…and so on…

Innermost and outermost redexes

Take a redex: $E = (\lambda x. M)\ N$

Any redex that is in M or in N is **inside** the redex E

The redex E is **outside** any redex that is in M or in N

A redex is **outermost** if there are no redexes outside it.

A redex is **innermost** if there are no redexes inside it.

$$\big((\lambda x\ y.\ x\ y\ x)\ t\ u\big)\Big(\Big(\lambda xyz.x\ \big((\lambda x.x\ x)\ y\big)\Big)\ v\ \big((\lambda x.x\ y)\ w\big)\Big)$$

(leftmost) outermost **outermost**

(leftmost) innermost **(rightmost) outermost**

Reduction strategies

Normal order

Reduces the **leftmost outermost** redex first

Always reduces a term to its normal form (if the normal form exists)

Call by name

Reduces the **leftmost outermost** redex first

Does not reduce inside λ-abstractions

Does not always reduce a term to its normal form

Call by value

Reduces the **leftmost innermost** redex first

Does not reduce inside λ-abstractions

Does not always reduce a term to its normal form

Reduction strategies

Normal order

Can perform computations in unevaluated function bodies

Is not used by any programming language

Call by name

Passes the function parameters unevaluated into the function body

Evaluates the passed function parameter on each use

Is used, with some variations, by, for example, Algol60, Haskell, R, and LaTeX

Call by value

Evaluates the function parameters before passing them into the function body

Terminates less often than call by name, but evaluates parameters only once

Is used, with some variations, by, for example, C, Scheme, and OCaml

Reduction strategies

$$((\lambda x\, y.\ x\ y\ x)\ t\ u)\Big(\big(\lambda xyz.x\ ((\lambda x.x\ x)\ y)\big)\ v\ ((\lambda x.x\ y)\ w)\Big)$$

Normal order:	first	second	third	fourth (twice)
Call-by-name:	first	second	never	never
Call-by-value:	first	second	third	never

Reduction strategies

$$\begin{tikzcd}
 & (\lambda x.y)((\lambda x.x\,x)(\lambda x.x\,x)) \arrow[dl, "\beta\mathrm{N}"'] \arrow[dr, "\beta\mathrm{V}"] & & & \\
y & & (\lambda x.y)((\lambda x.x\,x)(\lambda x.x\,x)) \arrow[dl, "\beta\mathrm{N}"'] \arrow[dr, "\beta\mathrm{V}"] & & \\
 & y & & (\lambda x.y)((\lambda x.x\,x)(\lambda x.x\,x)) \arrow[dr, "\beta\mathrm{V}"] & \\
 & & & & \text{...and so on...}
\end{tikzcd}$$

Reduction strategies

$$
\begin{tikzcd}
 & & (\lambda x.\, x\ x)\big((\lambda x.\, y)z\big) \arrow[dl, "\beta N"'] \arrow[ddrr, "\beta V"] & & \\
 & \big((\lambda x.\, y)z\big)\ \big((\lambda x.\, y)z\big) \arrow[dl, "\beta N"'] \arrow[dr, "\beta"] & & & \\
y\big((\lambda x.\, y)z\big) \arrow[drr, "\beta N"'] & & \big((\lambda x.\, y)z\big)\ y \arrow[d, "\beta"] & & (\lambda x.\, x\ x)\ y \arrow[dll, "\beta V"] \\
 & & yy & &
\end{tikzcd}
$$

TUTORIAL: REDUCTION STRATEGIES

6. (**Reduction strategies.**) Consider the λ-term:

$$((\lambda xy.\ x\ y\ x)\ t\ u)((\lambda xyz.x\ ((\lambda x.\ x\ x)\ y))\ v\ ((\lambda x.\ x\ y)\ w))$$

Perform as many as possible reduction steps for this term, ignoring α-conversion and using:

(a) the normal order reduction strategy;
(b) the call by name reduction strategy;
(c) the call by value reduction strategy;

For each step, underline the redex that is to be reduced in the next step. Comment on the differences that you observed.

SOLUTION: REDUCTION STRATEGIES

(b) Call by value: leftmost innermost redex first, no reduction under λ

$$
\begin{gathered}
\underline{((\lambda xy.\ x\ y\ x)\ t}\ u)((\lambda xyz.\ x\ ((\lambda x.\ x\ x)\ y))\ v\ ((\lambda x.\ x\ y)\ w)) \rightarrow_\beta \\
\underline{((\lambda y.\ t\ y\ t)\ u)}((\lambda xyz.\ x\ ((\lambda x.\ x\ x)\ y))\ v\ ((\lambda x.\ x\ y)\ w)) \rightarrow_\beta \\
(t\ u\ t)(\underline{(\lambda xyz.\ x\ ((\lambda x.\ x\ x)\ y))\ v}\ ((\lambda x.\ x\ y)\ w)) \rightarrow_\beta \\
(t\ u\ t)((\lambda yz.\ v\ ((\lambda x.\ x\ x)\ y))\ \underline{((\lambda x.\ x\ y)\ w)}) \rightarrow_\beta \\
(t\ u\ t)\underline{((\lambda yz.\ v\ ((\lambda x.\ x\ x)\ y))\ (w\ y))} \rightarrow_\beta \\
(t\ u\ t)(\lambda z.\ v\ ((\lambda x.\ x\ x)\ (w\ y))).
\end{gathered}
$$

BONUS

EXTENSIONALITY: η-EQUIVALENCE

$(\lambda x. f\ x) \neq_\beta f$ but $(\lambda x. f\ x)\ M =_\beta f\ M$

η-equivalence: $$\frac{x \notin \mathrm{FV}(M)}{(\lambda x. M\ x) =_\eta M}$$

More general (but infinitary) rule: $$\frac{M\ N =_{\eta^+} M' N, \text{ for all } N}{M =_{\eta^+} M'}$$

$=_{\beta\eta}$ and $=_{\beta\eta^+}$ capture "equality" better than just $=_\beta$

λ-CALCULUS: THE SIMPLEST PROGRAMMING LANGUAGE

$$M \; ::= \quad x \quad | \quad \lambda x.\, M \quad | \quad M \; M$$

Variable — Abstraction (single-parameter function) — Application

WHAT WILL WE COVER IN THESE LECTURES?

SYNTAX
Free/bound variables
α-equivalence
Substitution

SEMANTICS
β-reduction
Confluence/normal forms
Reduction strategies

APPLICATIONS
Expressivity
Arithmetic, Data structures
Recursion

Register machines, Turing machines: Computability

RM-computability. A partial function $f: \mathbb{N}^n \rightharpoonup \mathbb{N}$ is RM-computable *iff* there exists a register machine M with at least $n+1$ registers, $R_0, \dots, R_n$, with the following property: starting M from the state in which $R_0 = 0$, $R_i = x_i \mid_{i=1}^{n}$, M halts *iff* $f(x_1, \dots, x_n) \downarrow$, and in that case $R_0 = y$, where $y = f(x_1, \dots, x_n)$.

$\updownarrow$

Turing-computability. A partial function $f: \mathbb{N}^n \rightharpoonup \mathbb{N}$ is Turing-computable *iff* there exists a Turing machine M with the following property: starting M from its initial state, with tape head on the leftmost 0 of a tape coding $[x_1, \dots, x_n]$, M halts *iff* $f(x_1, \dots, x_n) \downarrow$, and in that case the final tape codes a list whose first element is y, where $y = f(x_1, \dots, x_n)$.

λ-CALCULUS: DEFINABILITY

A partial function $f\colon \mathbb{N}^n \rightharpoonup \mathbb{N}$ is **λ-definable** *iff* there exists a closed λ-term M with the following property:

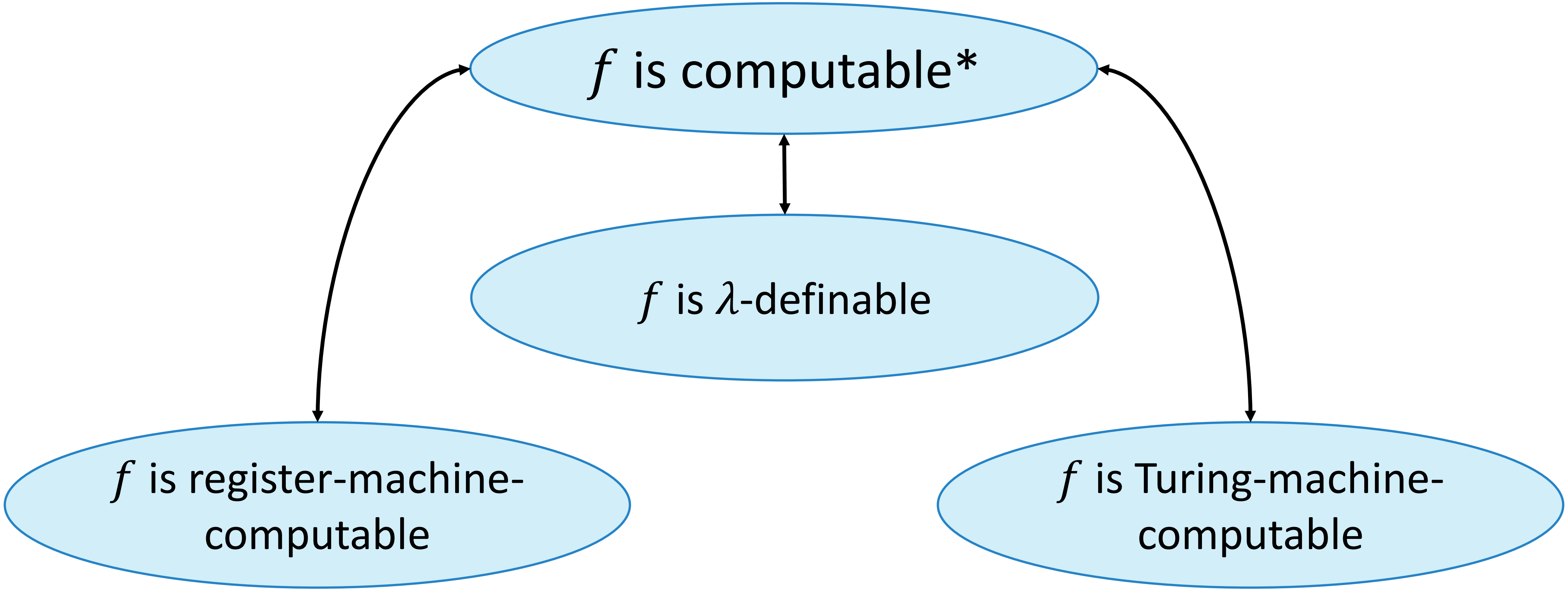
$$f(x_1, \dots, x_n) = y \quad \textit{iff} \quad M\ \underline{x_1}\ \underline{x_2}\ \dots\ \underline{x_n} =_\beta \underline{y}$$

and

$$f(x_1, \dots, x_n)\uparrow \quad \textit{iff} \quad M\ \underline{x_1}\ \underline{x_2}\ \dots\ \underline{x_n} \text{ has no normal form}$$

where $\underline{n}$ denotes the encoding of the natural number n in the λ-calculus.

The Church-Turing Thesis



* by a human following an algorithm, ignoring resource limitations

λ-CALCULUS: ENCODING NATURAL NUMBERS

Church numerals: $\underline{n} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{n}\ x)\ldots)$

Informally, a Church numeral n means:
"to do something n times"

$$\underline{0} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ x$$
$$\underline{1} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ f\ x$$
$$\underline{2} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ f\ (f\ x)$$
$$\underline{3} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ f\ (f\ (f\ x))$$
$$\ldots$$

ENCODING ADDITION

We have: $\underline{m} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{m}\ x)\ldots)$ and $\underline{n} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{n}\ x)\ldots)$

We need: $\text{plus}\ \underline{m}\ \underline{n} =_\beta \lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{m+n}\ x)\ldots)$

Encoding addition

We have: $\underline{m} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ \underbrace{f(\dots(f}_{m}\ x)\dots)$ and $\underline{n} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ \underbrace{f(\dots(f}_{n}\ x)\dots)$

$$\text{plus}\ \left(\lambda f.\lambda x.\ \underbrace{f(\dots(f}_{m}\ x)\dots)\right)\left(\lambda f.\lambda x.\ \boxed{\underbrace{f(\dots(f}_{n}\ x)\dots)}\right) =_\beta \lambda f.\lambda x.\ \underbrace{f(\dots(f}_{m+n}\ x)\dots)$$

✓

Encoding addition

We have: $\underline{m} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{m}\ x)\ldots)$ and $\underline{n} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{n}\ x)\ldots)$

$$\text{plus}\ \left(\lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{m}\ x)\ldots)\right)\left(\lambda f.\lambda x.\ \boxed{\underbrace{f(\ldots(f}_{n}\ x)\ldots)}\right) =_\beta \lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{m+n}\ x)\ldots)$$

- Obtain the body of $\underline{n}$: $\underline{n}\ f\ x =_\beta \underbrace{f(\ldots(f}_{n}\ x)\ldots)$
- Put this in the body of $\underline{m}$: $\underline{m}\ f\ (\underline{n}\ f\ x) =_\beta \underbrace{f(\ldots(f}_{m+n}\ x)\ldots)$
- Make this a Church numeral: $\lambda f.\lambda x.\underline{m}\ f\ (\underline{n}\ f\ x) =_\beta \underline{m+n}$
- Make this a function that accepts m and n: $\lambda m.\lambda n.\lambda f.\lambda x.m\ f\ (n\ f\ x)$

ENCODING ADDITION

We have: $\underline{m} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{m}\ x)\ldots)$ and $\underline{n} \stackrel{\text{def}}{=} \lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{n}\ x)\ldots)$

$$\text{plus}\ \left(\lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{m}\ x)\ldots)\right)\left(\lambda f.\lambda x.\ \boxed{\underbrace{f(\ldots(f}_{n}\ x)\ldots)}\right) =_\beta \lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{m+n}\ x)\ldots)$$

$$\text{plus} \equiv \lambda m.\lambda n.\lambda f.\lambda x.m\ f\ (n\ f\ x)$$

Exercise: Evaluate "plus $\underline{2}$ $\underline{3}$".

Tutorial: Encoding multiplication

(Encoding Multiplication in the λ-calculus.) Recall the Church numerals:

$$\underline{n} = (\lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{n}\ x)\ldots)).$$

Also, Recall the way in which we encoded addition:

$$\lambda m.\lambda n.\lambda f.\lambda x.\ m\ f\ (n\ f\ x).$$

Design a λ-term `mult` that encodes multiplication and test it by computing `mult` $\underline{2}$ $\underline{3}$. Explain your thinking process in the design phase.

SOLUTION: ENCODING MULTIPLICATION (PART 1)

We have $\underline{m} = \lambda f.\lambda x.\ f^m\ x$ and $\underline{n} = \lambda f.\lambda x.\ f^n\ x$. From these two, we need to construct $f^{m \cdot n}\ x$. Informally, we can do that by replacing each of the m occurrences of f in m with $\underline{n \text{ applications of } f}$. We could try in the same way as for the addition, taking $(n\ f\ x)$ and going for $m\ (n\ f\ x)$, since now we have to substitute for f, not for x. However, this produces:

$$\begin{aligned} m\ (n\ f\ x) = (\lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{m}\ x)\ldots))\ (n\ f\ x) \quad &= \quad (\lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{m}\ x)\ldots))\ (f^n\ x) \\ &\rightarrow_\beta \quad (\lambda x.\ \underbrace{((f^n\ x)\ldots((f^n\ x)}_{m}\ x)\ldots)) \end{aligned}$$

but we cannot β-reduce this further. We need to be able to propagate the inside x.

Solution: Encoding multiplication (part 2)

Let's try with $(n\ f)$ instead of $(n\ f\ x)$:

$$
\begin{aligned}
m\ (n\ f) &= (\lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{m}\ x)\ldots))\ (n\ f) \\
&= (\lambda f.\lambda x.\ \underbrace{f(\ldots(f}_{m}\ x)\ldots))\ (\lambda x.\ f^n\ x) \\
&=_\beta (\lambda x.\ \underbrace{((\lambda x.\ f^n\ x)\ldots((\lambda x.\ f^n\ x)}_{m}\ x)\ldots)) \\
&\to_\beta (\lambda x.\ \underbrace{((\lambda x.\ f^n\ x)\ldots((\lambda x.\ f^n\ x)}_{m-1}(f^n\ x))\ldots)) \\
&\to_\beta (\lambda x.\ \underbrace{((\lambda x.\ f^n\ x)\ldots((\lambda x.\ f^n\ x)}_{m-2}(f^{2\cdot n}\ x))\ldots)) \\
&\to_\beta^* (\lambda x.\ f^{m\cdot n}\ x)
\end{aligned}
$$

What remains is to wrap $m\ (n\ f)$ so that it is a function that takes two Church numerals (an additional $\lambda m.\lambda n$ and returns a Church numeral (an additional λf, like we did for the addition:

$$\texttt{mult} \equiv \lambda m.\lambda n.\lambda f.\ m\ (n\ f).$$

Some more encodings with natural numbers

Exponentiation: $\underline{m^n} \stackrel{\text{def}}{=} \lambda m.\lambda n.n\ m$

Conditional: if (m = 0) then x_1 else x_2

$\text{ifz} \stackrel{\text{def}}{=} \lambda m.\lambda x_1.\lambda x_2.m\ (\lambda z.x_2)\ x_1$

Exercise: define the successor and predecessor functions!

Tutorial: Encoding pairs

3. (**Pairs.**)

   Given two λ-terms, v_1, v_2, the pair of the two terms can be expressed in the λ-calculus as $\lambda p.\, p\ v_1\ v_2$ (where p does not occur free in v_1 or v_2). Define the following functions as λ-terms:

   (a) pair, which takes two λ-terms and constructs the pair of them;

   (b) fst, which returns the first value in a pair;

   (c) snd, which returns the second value in a pair.

Solution: Encoding pairs

(a)

$$\mathsf{pair} \stackrel{\text{def}}{=} \lambda v_1\ v_2.\,(\lambda p.\,p\ v_1\ v_2)$$

(b) Given the pair $\lambda p.\,p\ v_1\ v_2$, we have

$$\begin{aligned}(\lambda p.\,p\ v_1\ v_2)(\lambda w_1\ w_2.\,w_1) &\to (\lambda w_1\ w_2.\,w_1)\ v_1\ v_2 \\ &\to (\lambda w_2.\,v_1)\ v_2 \\ &\to v_1\end{aligned}$$

(up to alpha conversion) so we want fst to be a function that applies its argument (the pair) to the term $(\lambda w_1\ w_2.\,w_1)$:

$$\mathsf{fst} \stackrel{\text{def}}{=} \lambda q.\,q(\lambda w_1\ w_2.\,w_1)$$

(c)

$$\mathsf{snd} \stackrel{\text{def}}{=} \lambda q.\,q(\lambda w_1\ w_2.\,w_2)$$

Interlude: Combinators

Combinators: closed λ-terms.

$$\mathrm{I} \stackrel{\text{def}}{=} \lambda x.x \qquad \mathrm{K} \stackrel{\text{def}}{=} \lambda xy.x \qquad \mathrm{S} \stackrel{\text{def}}{=} \lambda xyz.xz(yz)$$

$$\mathrm{T} \stackrel{\text{def}}{=} \lambda xy.yx \qquad \mathrm{C} \stackrel{\text{def}}{=} \lambda xyz.xzy \qquad \mathrm{V} \stackrel{\text{def}}{=} \lambda xyz.zxy$$

$$\mathrm{B} \stackrel{\text{def}}{=} \lambda xyz.x(yz) \qquad \mathrm{B}' \stackrel{\text{def}}{=} \lambda xyz.y(xz) \qquad \mathrm{W} \stackrel{\text{def}}{=} \lambda xy.xyy$$

SKI and application are all we need to define **any computable function**, meaning that we can even get rid of the λ-abstraction.
This is called the SKI-combinator calculus.

TUTORIAL: COMBINATORS

4. (**SKI Combinators.**)
   Let $S \stackrel{\text{def}}{=} \lambda xyz.(xz)(yz)$ and $K \stackrel{\text{def}}{=} \lambda xy.x$. Reduce SKK to normal form. (Hint: This can be messy if you are not careful. Keep the abbreviations S and K around as long as you can and replace them with their corresponding λ-terms only if you need to. This makes it much easier)

Solution: Combinators

(SKI Combinators.)

$$
\begin{aligned}
&SKK \\
&= (\lambda xyz.\ (xz)(yz))KK \\
&\to (\lambda yz.\ (Kz)(yz))K \\
&\to \lambda z.\ (Kz)(Kz) \\
&= \lambda z.\ ((\lambda xy.\ x)z)(Kz) \\
&\to \lambda z.\ (\lambda y.\ z)(Kz) \\
&\to \lambda z.\ z
\end{aligned}
$$

Notice that we shown that SKK is α-equivalent to I. Given our definitions of S and K, we could define $I \stackrel{\text{def}}{=} SKK$ (or indeed $I \stackrel{\text{def}}{=} SKS$ – can you see why?)

Recursion: Factorial

Factorial: fact n $\stackrel{\text{def}}{=}$ if (n = 0) then 1 else n*fact(n-1)

Encoding: $\text{fact} \ =_\beta \ \lambda n.\,\text{ifz}\, n\, \underline{1} \left(\text{mult}\, n\, (\text{fact}\, (\text{pred}\, n))\right)$

$$\text{fact} \ =_\beta \ \underbrace{\left(\lambda f.\,\lambda n.\,\text{ifz}\, n\, \underline{1} \left(\text{mult}\, n\, (f\ (\text{pred}\, n))\right)\right)}_{F} \text{fact}$$

This means that fact is a **fixpoint** of F

Can we define the fixpoint operator in the λ-calculus?

Recursion: the Y combinator

$$\mathrm{Y} \stackrel{\text{def}}{=} \lambda f.\left(\lambda x. f\ (x\ x)\right)\left(\lambda x. f\ (x\ x)\right)$$

After one step of β-reduction: $\mathrm{Y}\ f \rightarrow_{\beta} f\ (\mathrm{Y}\ f)$

This means that, for any f, $\mathrm{Y}\ f$ is the fixpoint of f, that is, that Y is the fixpoint operator.

Recursion: Factorial revisited

Factorial: fact n $\stackrel{\text{def}}{=}$ if (n = 0) then 1 else n*fact(n-1)

Encoding: $\text{fact} \stackrel{\text{def}}{=} \mathrm{Y}\left(\lambda f.\,\lambda n.\,\text{ifz}\; n\; \underline{1}\left(\text{mult}\; n\; \left(f\,(\text{pred}\; n)\right)\right)\right)$

Exercise: Evaluate "fact $\underline{2}$".