CENG 2010 - Programming Language Concepts

Week 7 and Week 9: (Untyped) λ-Calculus

Burak Ekici

April 17 and May 8, 2023

Outline

Church's Thesis

A function is said to be effectively computable if it could be computed in a finite amount of time using finite resources

Church's Thesis

A function is said to be effectively computable if it could be computed in a finite amount of time using finite resources

- Effectively computable functions are just those could be computed by a Turing Machine

  $\Longleftrightarrow$

  Effectively computable functions are just those definable in the lambda calculus

Church’s Thesis

A function is said to be effectively computable if it could be computed in a finite amount of time using finite resources

- Effectively computable functions are just those could be computed by a Turing Machine

  $$\Longleftrightarrow$$

  Effectively computable functions are just those definable in the lambda calculus

- Effectively computable is an intuitive notion not a mathematical one: Church’s thesis cannot be proven
- Only refutable – by counterexample: give a function that could be computed with some model but not with a Turing machine

Uncomputability

A problem that cannot be solved by any Turing machine in finite time (or any equivalent formalism) is called uncomputable

Uncomputability

A problem that cannot be solved by any Turing machine in finite time (or any equivalent formalism) is called uncomputable

- The Halting Problem: given an arbitrary Turing machine and its input tape, will the machine eventually halt?

Uncomputability

A problem that cannot be solved by any Turing machine in finite time (or any equivalent formalism) is called uncomputable

- The Halting Problem: given an arbitrary Turing machine and its input tape, will the machine eventually halt?
- The Halting Problem is provably uncomputable – which means that it cannot be solved in practice.

What is a Function? – Extensional View

- Functions as graphs

What is a Function? – Extensional View

- Functions as graphs
  - each function f has a fixed domain X and a co-domain Y
  - each function $f: X \to Y$ is a set of pairs $f \subseteq X \times Y$ such that for each $x \in X$, there exists exactly one $y \in Y$ such that $(x, y) \in f$

What is a Function? – Extensional View

- Functions as graphs
  - each function f has a fixed domain X and a co-domain Y
  - each function $f : X \to Y$ is a set of pairs $f \subseteq X \times Y$ such that for each $x \in X$, there exists exactly one $y \in Y$ such that $(x, y) \in f$
- Equality of functions

What is a Function? – Extensional View

- Functions as graphs
  - each function f has a fixed domain X and a co-domain Y
  - each function $f : X \to Y$ is a set of pairs $f \subseteq X \times Y$ such that for each $x \in X$, there exists exactly one $y \in Y$ such that $(x, y) \in f$
- Equality of functions
  - two functions are said to be equal, for each input they yield the same output:

What is a Function? – Extensional View

- Functions as graphs
  - each function f has a fixed domain X and a co-domain Y
  - each function $f: X \to Y$ is a set of pairs $f \subseteq X \times Y$ such that for each $x \in X$, there exists exactly one $y \in Y$ such that $(x, y) \in f$
- Equality of functions
  - two functions are said to be equal, for each input they yield the same output:

$$f, g: X \to Y, \quad f = g \iff \forall x \in X, f(x) = g(x)$$

What is a Function? – Intensional View

- A function $f : A \to B$ is an abstraction $\lambda x.e$, where x is a variable name, and e is an expression, such that when a value $a \in A$ is substituted for x in e, then this expression (i.e., $f(a)$) evaluates to some (unique) value $b \in B$

What is a Function? – Intensional View

- A function $f: A \rightarrow B$ is an abstraction $\lambda x.e$, where x is a variable name, and e is an expression, such that when a value $a \in A$ is substituted for x in e, then this expression (i.e., $f(a)$) evaluates to some (unique) value $b \in B$
- Equality of functions

What is a Function? – Intensional View

- A function $f : A \rightarrow B$ is an abstraction $\lambda x.e$, where x is a variable name, and e is an expression, such that when a value $a \in A$ is substituted for x in e, then this expression (i.e., $f(a)$) evaluates to some (unique) value $b \in B$
- Equality of functions
  - two functions are equal if they are defined by (essentially) the same abstraction/formula

Observations About Functions

- functions need not be explicitly named

Observations About Functions

- functions need not be explicitly named
  - i.e., identity functions $f(x) = g(x) = x$ could be expressed by $x \mapsto x$ having no name

Observations About Functions

- functions need not be explicitly named
  - i.e., identity functions $f(x) = g(x) = x$ could be expressed by $x \mapsto x$ having no name
- specific choice of argument names are irrelevant

Observations About Functions

- functions need not be explicitly named
  - i.e., identity functions $f(x) = g(x) = x$ could be expressed by $x \mapsto x$ having no name
- specific choice of argument names are irrelevant
  - i.e., $(x, y) \mapsto x - y$ and $(u, v) \mapsto u - v$ are the same

Observations About Functions

- functions need not be explicitly named
  - i.e., identity functions $f(x) = g(x) = x$ could be expressed by $x \mapsto x$ having no name
- specific choice of argument names are irrelevant
  - i.e., $(x, y) \mapsto x - y$ and $(u, v) \mapsto u - v$ are the same
- functions can be written in a way to accept a single input (currification)

Observations About Functions

- functions need not be explicitly named
  - i.e., identity functions $f(x) = g(x) = x$ could be expressed by $x \mapsto x$ having no name
- specific choice of argument names are irrelevant
  - i.e., $(x, y) \mapsto x - y$ and $(u, v) \mapsto u - v$ are the same
- functions can be written in a way to accept a single input (currification)
  - i.e., $(x, y) \mapsto x - y$ could be rewritten as $x \mapsto (y \mapsto x - y)$

λ-Calculus

- λ-calculus: theory of functions as formulas (based on aforementioned observations) – mathematical formalism to express computations as functions

λ-Calculus

- λ-calculus: theory of functions as formulas (based on aforementioned observations) – mathematical formalism to express computations as functions
- Easier manipulation of functions using expressions

λ-Calculus

- λ-calculus: theory of functions as formulas (based on aforementioned observations) – mathematical formalism to express computations as functions
- Easier manipulation of functions using expressions
- Examples of λ-notation (expressions):

λ-Calculus

- λ-calculus: theory of functions as formulas (based on aforementioned observations) – mathematical formalism to express computations as functions
- Easier manipulation of functions using expressions
- Examples of λ-notation (expressions):
  - The identity function $f(x) = x$ is denoted as $\lambda x.x$

λ-Calculus

- λ-calculus: theory of functions as formulas (based on aforementioned observations) – mathematical formalism to express computations as functions
- Easier manipulation of functions using expressions
- Examples of λ-notation (expressions):
  - The identity function $f(x) = x$ is denoted as $\lambda x.x$
  - $\lambda x.x$ is the same as $\lambda y.y$ (called α-equivalence)

λ-Calculus

- λ-calculus: theory of functions as formulas (based on aforementioned observations) – mathematical formalism to express computations as functions
- Easier manipulation of functions using expressions
- Examples of λ-notation (expressions):
  - The identity function $f(x) = x$ is denoted as $\lambda x.x$
  - $\lambda x.x$ is the same as $\lambda y.y$ (called α-equivalence)
  - Function defined as $f := x \mapsto x^2$ is written as $\lambda x.x^2$

λ-Calculus

- λ-calculus: theory of functions as formulas (based on aforementioned observations) – mathematical formalism to express computations as functions
- Easier manipulation of functions using expressions
- Examples of λ-notation (expressions):
  - The identity function $f(x) = x$ is denoted as $\lambda x.x$
  - $\lambda x.x$ is the same as $\lambda y.y$ (called α-equivalence)
  - Function defined as $f := x \mapsto x^2$ is written as $\lambda x.x^2$
  - $f(5)$ is $(\lambda x.x^2)(5)$, and evaluates to 25 (called β-reduction)

Definition (λ-terms)

$$
\begin{array}{lll}
\text{Terms } s, t, r := & & \\
\quad | & x & \text{variable (countable many)} \\
\quad | & \lambda x.\, t & \text{function abstraction} \\
\quad | & s\ t & \text{function application}
\end{array}
$$

Definition (λ-equations)

1. β-equivalence – to get there, we first need to define α-equivalence and substitution

Definition (Free Variables)

- An occurrence of variable x is said to be bound when it occurs in the body t of an abstraction $\lambda x.t$

Definition (Free Variables)

- An occurrence of variable x is said to be bound when it occurs in the body t of an abstraction $\lambda x.t$
- An occurrence of x is free if it appears in a position where it is not bound by an enclosing abstraction of x

Definition (Free Variables)

- An occurrence of variable x is said to be bound when it occurs in the body t of an abstraction $\lambda x.t$
- An occurrence of x is free if it appears in a position where it is not bound by an enclosing abstraction of x
- The set of free variables of a term is

$$FV(x) \quad = \quad \{x\}$$

Definition (Free Variables)

- An occurrence of variable x is said to be bound when it occurs in the body t of an abstraction $\lambda x.t$
- An occurrence of x is free if it appears in a position where it is not bound by an enclosing abstraction of x
- The set of free variables of a term is

$$\begin{array}{lcl} FV(x) & = & \{x\} \\ FV(\lambda x.e) & = & FV(e)\backslash\{x\} \end{array}$$

Definition (Free Variables)

- An occurrence of variable x is said to be bound when it occurs in the body t of an abstraction $\lambda x.t$
- An occurrence of x is free if it appears in a position where it is not bound by an enclosing abstraction of x
- The set of free variables of a term is

$$\begin{array}{lcl} FV(x) & = & \{x\} \\ FV(\lambda x.e) & = & FV(e) \setminus \{x\} \\ FV(e_1 e_2) & = & FV(e_1) \cup FV(e_2) \end{array}$$

Definition (Free Variables)

- An occurrence of variable x is said to be bound when it occurs in the body t of an abstraction $\lambda x.t$
- An occurrence of x is free if it appears in a position where it is not bound by an enclosing abstraction of x
- The set of free variables of a term is

$$\begin{array}{lcl} FV(x) & = & \{x\} \\ FV(\lambda x.e) & = & FV(e)\backslash\{x\} \\ FV(e_1 e_2) & = & FV(e_1) \cup FV(e_2) \end{array}$$

- A term e is called closed if $FV(e) = \emptyset$

Example (Free and Bound Variables)

- Let M be the following lambda term: $\lambda x.\lambda y.\Big(\big(\lambda z.\lambda v.z\,(z\,v)\big)\,(x\,y)\,(z\,u)\Big)$

Example (Free and Bound Variables)

- Let M be the following lambda term: $\lambda x.\lambda y.\Big((\lambda z.\lambda v.z\,(z\,v))\,(x\,y)\,(z\,u)\Big)$

$$FV(M) \quad = \quad FV\big((\lambda z.\lambda v.z\,(z\,v))\,(x\,y)\,(z\,u)\big)\backslash\{x, y\}$$

Example (Free and Bound Variables)

- Let M be the following lambda term: $\lambda x.\lambda y.\Big((\lambda z.\lambda v.z\,(z\,v))\,(x\,y)\,(z\,u)\Big)$

$$\begin{aligned} FV(M) &= FV\big((\lambda z.\lambda v.z\,(z\,v))\,(x\,y)\,(z\,u)\big)\backslash\{x, y\} \\ &= \big(FV(\lambda z.\lambda v.z\,(z\,v)) \cup FV(x\,y) \cup FV(z\,u)\big)\backslash\{x, y\} \end{aligned}$$

Example (Free and Bound Variables)

- Let M be the following lambda term: $\lambda x.\lambda y.\Big((\lambda z.\lambda v.z\,(z\,v))\,(x\,y)\,(z\,u)\Big)$

$$
\begin{aligned}
FV(M) &= FV\big((\lambda z.\lambda v.z\,(z\,v))\,(x\,y)\,(z\,u)\big)\backslash\{x, y\} \\
&= \big(FV(\lambda z.\lambda v.z\,(z\,v)) \cup FV(x\,y) \cup FV(z\,u)\big)\backslash\{x, y\} \\
&= \big((\{z, v\}\backslash\{z, v\}) \cup \{x, y\} \cup \{z, u\}\big)\backslash\{x, y\}
\end{aligned}
$$

Example (Free and Bound Variables)

- Let M be the following lambda term: $\lambda x.\lambda y.\Big((\lambda z.\lambda v.z\,(z\,v))\,(x\,y)\,(z\,u)\Big)$

$$
\begin{aligned}
FV(M) &= FV\big((\lambda z.\lambda v.z\,(z\,v))\,(x\,y)\,(z\,u)\big)\backslash\{x, y\} \\
&= \big(FV(\lambda z.\lambda v.z\,(z\,v)) \cup FV(x\,y) \cup FV(z\,u)\big)\backslash\{x, y\} \\
&= \big((\{z, v\}\backslash\{z, v\}) \cup \{x, y\} \cup \{z, u\}\big)\backslash\{x, y\} \\
&= \{z, u\}
\end{aligned}
$$

Definition (α-equivalence)

- names of λ-bound variables should not affect meaning

Definition (α-equivalence)

- names of λ-bound variables should not affect meaning
- e.g., $\lambda f.\, \lambda x.\, f\ x$ should have the same meaning as $\lambda x.\, \lambda y.\, x\ y$

Definition (α-equivalence)

- names of λ-bound variables should not affect meaning
- e.g., $\lambda f. \lambda x. f\ x$ should have the same meaning as $\lambda x. \lambda y. x\ y$
- this issue is best dealt with at the level of syntax rather than semantics

Definition (α-equivalence)

- names of λ-bound variables should not affect meaning
- e.g., $\lambda f.\, \lambda x.\, f\ x$ should have the same meaning as $\lambda x.\, \lambda y.\, x\ y$
- this issue is best dealt with at the level of syntax rather than semantics
- from now on we re-define λ term to mean not an abstract syntax tree but rather an equivalence class of such trees with respect to α-equivalence $s =_\alpha t$:

$$\frac{}{x =_\alpha x}$$

$$\frac{s =_\alpha s' \qquad t =_\alpha t'}{s\ t =_\alpha s'\ t'}$$

$$\frac{t \cdot (y\ x) =_\alpha t' \cdot (y\ x') \qquad y \text{ does not occur in } \{x,x',t,t'\}}{\lambda x.\, t =_\alpha \lambda x'.\, t'}$$

where $t \cdot (y\ x)$ denotes the result of replacing all occurrences of x with y in t

Example (α-equivalence)

$$
\begin{array}{rclcl}
\lambda x.\, x\, x & =_\alpha & \lambda y.\, y\, y & \neq_\alpha & \lambda x.\, x\, y \\
(\lambda y.\, y)\, x & =_\alpha & (\lambda x.\, x)\, x & \neq_\alpha & (\lambda x.\, x)\, y
\end{array}
$$

Definition (substitution)

- substitution $t[s/x]$ denotes the result of replacing all free occurrences of variable x in term t (i.e. those not occurring within the scope of a $\lambda x.\_$ binder) by the term s

Definition (substitution)

- substitution $t[s/x]$ denotes the result of replacing all free occurrences of variable x in term t (i.e. those not occurring within the scope of a $\lambda x.\_$ binder) by the term s
- alpha-converting λ-bound variables in t to avoid them "capturing" any free variables of t

Definition (substitution)

- substitution $t[s/x]$ denotes the result of replacing all free occurrences of variable x in term t (i.e. those not occurring within the scope of a $\lambda x.\,\_$ binder) by the term s
- alpha-converting λ-bound variables in t to avoid them "capturing" any free variables of t
- e.g., $(\lambda y.\,(y,x))[y/x]$ is $\lambda z.\,(z,y)$ and is not $\lambda y.\,(y,y)$

Definition (substitution)

- substitution $t[s/x]$ denotes the result of replacing all free occurrences of variable x in term t (i.e. those not occurring within the scope of a $\lambda x.\,\_$ binder) by the term s
- alpha-converting λ-bound variables in t to avoid them "capturing" any free variables of t
- e.g., $(\lambda y.\,(y, x))[y/x]$ is $\lambda z.\,(z, y)$ and is not $\lambda y.\,(y, y)$
- the relation $t[s/x] = t'$ can be inductively defined by the following rules:

$$\frac{}{x[s/x] = s} \qquad \frac{y \neq x}{y[s/x] = y}$$

$$\frac{t[s/x] = t' \qquad y \neq x \text{ and } y \text{ does not freely occur in } s}{(\lambda y.\,t)[s/x] = \lambda y.\,t'}$$

$$\frac{t_1[s/x] = t_1' \qquad t_2[s/x] = t_2'}{(t_1\ t_2)[s/x] = t_1'\ t_2'}$$

Example (Substitution)

$$(\lambda x.\, \lambda y.\, x\ y\ z)[w/z] \quad = \quad \lambda x.\, \lambda y.\, x\ y\ w$$

Example (Substitution)

$$
\begin{array}{lcl}
(\lambda x.\, \lambda y.\, x\ y\ z)[w/z] & = & \lambda x.\, \lambda y.\, x\ y\ w \\
(\lambda y.\, y\ x)[y/x] & = & \lambda z.\, z\ y
\end{array}
$$

Example (Substitution)

$$
\begin{aligned}
(\lambda x.\lambda y.\, x\ y\ z)[w/z] &= \lambda x.\lambda y.\, x\ y\ w \\
(\lambda y.\, y\ x)[y/x] &= \lambda z.\, z\ y \\
(\lambda x.\lambda y.\, x\ y\ z)[y/z] &= \lambda x.\lambda a.\, x\ a\ y
\end{aligned}
$$

Example (Substitution)

$$
\begin{array}{lcl}
(\lambda x.\lambda y.\, x\ y\ z)[w/z] & = & \lambda x.\lambda y.\, x\ y\ w \\
(\lambda y.\, y\ x)[y/x] & = & \lambda z.\, z\ y \\
(\lambda x.\lambda y.\, x\ y\ z)[y/z] & = & \lambda x.\lambda a.\, x\ a\ y \\
(\lambda x.\lambda y.\, x\ y\ z)[(\lambda x.\, x\ x)/y] & = & \lambda x.\lambda y.\, x\ y\ z
\end{array}
$$

Example (Substitution)

$$
\begin{array}{lcl}
(\lambda x.\lambda y.x\ y\ z)[w/z] & = & \lambda x.\lambda y.x\ y\ w \\
(\lambda y.y\ x)[y/x] & = & \lambda z.z\ y \\
(\lambda x.\lambda y.x\ y\ z)[y/z] & = & \lambda x.\lambda a.x\ a\ y \\
(\lambda x.\lambda y.x\ y\ z)[(\lambda x.x\ x)/y] & = & \lambda x.\lambda y.x\ y\ z \\
(\lambda x.\lambda y.x\ y\ z)[(\lambda x.x\ y)/z] & = & \lambda x.\lambda a.x\ a\ (\lambda x.x\ y)
\end{array}
$$

Definition (β-equivalence (or β-reduction))

the relation $s =_\beta t$ (where s and t over terms) is inductively defined by the following rules:

- β-conversion

$$\frac{}{(\lambda x.\, t)\ s =_\beta t[s/x]}$$

Definition (β-equivalence (or β-reduction))

the relation $s =_\beta t$ (where s and t over terms) is inductively defined by the following rules:

- β-conversion

$$\frac{}{(\lambda x.\, t)\ s =_\beta t[s/x]}$$

- congruence rules

$$\frac{t =_\beta t'}{\lambda x.\, t =_\beta \lambda x.\, t'} \qquad \frac{s =_\beta s' \qquad t =_\beta t'}{s\ t =_\beta s'\ t'}$$

Definition (β-equivalence (or β-reduction))

the relation $s =_\beta t$ (where s and t over terms) is inductively defined by the following rules:

- β-conversion

$$\frac{}{(\lambda x.\, t)\ s =_\beta t[s/x]}$$

- congruence rules

$$\frac{t =_\beta t'}{\lambda x.\, t =_\beta \lambda x.\, t'} \qquad \frac{s =_\beta s' \qquad t =_\beta t'}{s\ t =_\beta s'\ t'}$$

- $=_\beta$ is reflexive, symmetric and transitive

$$\frac{}{t =_\beta t} \qquad \frac{s =_\beta t}{t =_\beta s} \qquad \frac{r =_\beta s \qquad s =_\beta t}{r =_\beta t}$$

Example (β-reduction)

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

Example (β-reduction)

$$\underline{(\lambda x.\, x)\ (\lambda x.\, x)} \ \rightarrow_\beta \ x[x := \lambda x.\, x]$$

Example (β-reduction)

$$\underline{(\lambda x.\, x)\ (\lambda x.\, x)} \rightarrow_\beta x[x := \lambda x.\, x] \qquad = \lambda x.\, x$$

Example (β-reduction)

$$\underline{(\lambda x.\, x)\ (\lambda x.\, x)} \rightarrow_\beta x[x := \lambda x.\, x] \quad = \quad \lambda x.\, x$$

Example (β-reduction)

$$\begin{aligned} \underline{(\lambda x.\, x)\ (\lambda x.\, x)} &\to_\beta x[x := \lambda x.\, x] && = \lambda x.\, x \\ \underline{(\lambda xy.\, y)\ (\lambda x.\, x)} &\to_\beta (\lambda y.\, y)[x := \lambda x.\, x] \end{aligned}$$

Example (β-reduction)

$$
\begin{aligned}
\underline{(\lambda x.\, x)\ (\lambda x.\, x)} \ \rightarrow_\beta\ x[x := \lambda x.\, x] &= \lambda x.\, x \\
\underline{(\lambda xy.\, y)\ (\lambda x.\, x)} \ \rightarrow_\beta\ (\lambda y.\, y)[x := \lambda x.\, x] &= \lambda y.\, y
\end{aligned}
$$

Example (β-reduction)

$$\begin{aligned}
\underline{(\lambda x.\, x)\ (\lambda x.\, x)} &\rightarrow_\beta x[x := \lambda x.\, x] &&= \lambda x.\, x \\
\underline{(\lambda xy.\, y)\ (\lambda x.\, x)} &\rightarrow_\beta (\lambda y.\, y)[x := \lambda x.\, x] &&= \lambda y.\, y
\end{aligned}$$

Example (β-reduction)

$$
\begin{aligned}
\underline{(\lambda x.\, x)\ (\lambda x.\, x)} &\to_\beta x[x := \lambda x.\, x] &&= \lambda x.\, x \\
\underline{(\lambda xy.\, y)\ (\lambda x.\, x)} &\to_\beta (\lambda y.\, y)[x := \lambda x.\, x] &&= \lambda y.\, y \\
\underline{(\lambda xyz.\, x\, z\, (y\, z))\ (\lambda x.\, x)} &\to_\beta \lambda yz.\, \underline{(\lambda x.\, x)\, z}\, (y\, z)
\end{aligned}
$$

Example (β-reduction)

$$
\begin{aligned}
\underline{(\lambda x.\, x)\ (\lambda x.\, x)} &\to_\beta x[x := \lambda x.\, x] &&= \lambda x.\, x \\
\underline{(\lambda xy.\, y)\ (\lambda x.\, x)} &\to_\beta (\lambda y.\, y)[x := \lambda x.\, x] &&= \lambda y.\, y \\
\underline{(\lambda xyz.\, x\, z\ (y\, z))\ (\lambda x.\, x)} &\to_\beta \lambda yz.\, \underline{(\lambda x.\, x)\ z}\ (y\, z) &&\to_\beta \lambda yz.\, z\ (y\, z)
\end{aligned}
$$

Example (β-reduction)

$$
\begin{array}{rcll}
\underline{(\lambda x.\, x)\ (\lambda x.\, x)} & \rightarrow_\beta & x[x := \lambda x.\, x] & = \ \lambda x.\, x \\
\underline{(\lambda xy.\, y)\ (\lambda x.\, x)} & \rightarrow_\beta & (\lambda y.\, y)[x := \lambda x.\, x] & = \ \lambda y.\, y \\
\underline{(\lambda xyz.\, x\, z\, (y\, z))\ (\lambda x.\, x)} & \rightarrow_\beta & \lambda yz.\, \underline{(\lambda x.\, x)\, z}\, (y\, z) & \rightarrow_\beta \ \lambda yz.\, z\, (y\, z)
\end{array}
$$

Example (β-reduction)

$$
\begin{array}{rcll}
\underline{(\lambda x.\, x)\ (\lambda x.\, x)} & \to_\beta & x[x := \lambda x.\, x] & = \ \lambda x.\, x \\
\underline{(\lambda xy.\, y)\ (\lambda x.\, x)} & \to_\beta & (\lambda y.\, y)[x := \lambda x.\, x] & = \ \lambda y.\, y \\
\underline{(\lambda xyz.\, x\, z\, (y\, z))\ (\lambda x.\, x)} & \to_\beta & \lambda yz.\, \underline{(\lambda x.\, x)\, z}\, (y\, z) & \to_\beta \ \lambda yz.\, z\, (y\, z) \\
\underline{(\lambda x.\, x\, x)\ (\lambda x.\, x\, x)} & \to_\beta & \underline{(\lambda x.\, x\, x)\ (\lambda x.\, x\, x)} &
\end{array}
$$

Example (β-reduction)

$$
\begin{aligned}
\underline{(\lambda x.\, x)\ (\lambda x.\, x)} &\rightarrow_\beta x[x := \lambda x.\, x] &&= \lambda x.\, x \\
\underline{(\lambda xy.\, y)\ (\lambda x.\, x)} &\rightarrow_\beta (\lambda y.\, y)[x := \lambda x.\, x] &&= \lambda y.\, y \\
\underline{(\lambda xyz.\, x\, z\, (y\, z))\ (\lambda x.\, x)} &\rightarrow_\beta \lambda yz.\, \underline{(\lambda x.\, x)\, z}\, (y\, z) &&\rightarrow_\beta \lambda yz.\, z\, (y\, z) \\
\underline{(\lambda x.\, x\, x)\ (\lambda x.\, x\, x)} &\rightarrow_\beta \underline{(\lambda x.\, x\, x)\ (\lambda x.\, x\, x)} &&\rightarrow_\beta \cdots
\end{aligned}
$$

Outline

1. λ-Calculus

2. Programming In λ-Calculus

Programming in λ-Calculus

- Recall Church's thesis: Turing machines $\iff$ λ-Calculus

Programming in λ-Calculus

- Recall Church's thesis: Turing machines $\iff$ λ-Calculus
- We shall see how different types of data and related operations can be programmed in λ-calculus.

Programming in λ-Calculus

- Recall Church's thesis: Turing machines $\iff$ λ-Calculus
- We shall see how different types of data and related operations can be programmed in λ-calculus.
- Functions with many arguments: currification

Representing Booleans

`true`	:=	$\lambda a.\lambda b.a$
`false`	:=	$\lambda a.\lambda b.b$
`not`	:=	$\lambda a.\lambda b.\lambda c.a\,c\,b$
`and`	:=	$\lambda a.\lambda b.a\,b\,a$
`or`	:=	$\lambda a.\lambda b.a\,a\,b$
`if` a `then` b `else` c	:=	$\lambda a.\lambda b.\lambda c.\,a\,b\,c$

Representing Booleans

$$
\begin{array}{lcl}
\texttt{true} & := & \lambda a.\lambda b.a \\
\texttt{false} & := & \lambda a.\lambda b.b \\
\texttt{not} & := & \lambda a.\lambda b.\lambda c.a\,c\,b \\
\texttt{and} & := & \lambda a.\lambda b.a\,b\,a \\
\texttt{or} & := & \lambda a.\lambda b.a\,a\,b \\
\texttt{if}\ a\ \texttt{then}\ b\ \texttt{else}\ c & := & \lambda a.\lambda b.\lambda c.\,a\,b\,c
\end{array}
$$

For example:
$(\lambda a.\,\lambda b.\,\lambda c.\,a\,b\,c)(\lambda x.\lambda y.x) =_\beta \lambda b.\,\lambda c.\,(\lambda x.\lambda y.x)\;(b\,c) =_\beta \lambda b.\,\lambda c.\,(\lambda y.b)\;(c) =_\beta \lambda b.\,\lambda c.\,b$

Representing Booleans

$$
\begin{array}{lcl}
\texttt{true} & := & \lambda a.\lambda b.a \\
\texttt{false} & := & \lambda a.\lambda b.b \\
\texttt{not} & := & \lambda a.\lambda b.\lambda c.a\,c\,b \\
\texttt{and} & := & \lambda a.\lambda b.a\,b\,a \\
\texttt{or} & := & \lambda a.\lambda b.a\,a\,b \\
\texttt{if}\ a\ \texttt{then}\ b\ \texttt{else}\ c & := & \lambda a.\lambda b.\lambda c.\,a\,b\,c
\end{array}
$$

For example:
$(\lambda a.\,\lambda b.\,\lambda c.\,a\,b\,c)(\lambda x.\lambda y.x) =_\beta \lambda b.\,\lambda c.\,(\lambda x.\lambda y.x)\ (b\,c) =_\beta \lambda b.\,\lambda c.\,(\lambda y.b)\ (c) =_\beta \lambda b.\,\lambda c.\,b$
$(\lambda a.\,\lambda b.\,\lambda c.\,a\,b\,c)(\lambda x.\lambda y.y) =_\beta \lambda b.\,\lambda c.\,(\lambda x.\lambda y.y)\ (b\,c) =_\beta \lambda b.\,\lambda c.\,(\lambda y.y)\ (c) =_\beta \lambda b.\,\lambda c.\,c$

Representing Booleans

$$
\begin{array}{lcl}
\texttt{true} & := & \lambda a.\lambda b.a \\
\texttt{false} & := & \lambda a.\lambda b.b \\
\texttt{not} & := & \lambda a.\lambda b.\lambda c.a\,c\,b \\
\texttt{and} & := & \lambda a.\lambda b.a\,b\,a \\
\texttt{or} & := & \lambda a.\lambda b.a\,a\,b \\
\texttt{if}\ a\ \texttt{then}\ b\ \texttt{else}\ c & := & \lambda a.\lambda b.\lambda c.\,a\,b\,c
\end{array}
$$

For example:
$(\lambda a.\,\lambda b.\,\lambda c.\,a\,b\,c)(\lambda x.\lambda y.x) =_\beta \lambda b.\,\lambda c.\,(\lambda x.\lambda y.x)\ (b\,c) =_\beta \lambda b.\,\lambda c.\,(\lambda y.b)\ (c) =_\beta \lambda b.\,\lambda c.\,b$
$(\lambda a.\,\lambda b.\,\lambda c.\,a\,b\,c)(\lambda x.\lambda y.y) =_\beta \lambda b.\,\lambda c.\,(\lambda x.\lambda y.y)\ (b\,c) =_\beta \lambda b.\,\lambda c.\,(\lambda y.y)\ (c) =_\beta \lambda b.\,\lambda c.\,c$

$$
\begin{array}{lcl}
\texttt{not true} & := & (\lambda a.\lambda b.\lambda c.a\,c\,b)\,(\lambda a.\lambda b.a) \\
 & =_\beta & (\lambda b.\lambda c.(\lambda a.\lambda b.a)\,c\,b) \\
 & =_\beta & (\lambda b.\lambda c.c)
\end{array}
$$

Representing Natural Numbers

- Church numerals:

$$\mathtt{0} \quad := \quad \lambda s.\lambda z.z$$

Representing Natural Numbers

- Church numerals:

$$\begin{aligned} \texttt{0} &:= \lambda s.\lambda z.z \\ \texttt{1} &:= \lambda s.\lambda z.s\,z \end{aligned}$$

Representing Natural Numbers

- Church numerals:

$$
\begin{aligned}
\mathtt{0} &:= \lambda s.\lambda z.z \\
\mathtt{1} &:= \lambda s.\lambda z.s\,z \\
\mathtt{2} &:= \lambda s.\lambda z.s\,(s\,z)
\end{aligned}
$$

Representing Natural Numbers

- Church numerals:

$$
\begin{aligned}
\mathtt{0} &:= \lambda s.\lambda z.z \\
\mathtt{1} &:= \lambda s.\lambda z.s\,z \\
\mathtt{2} &:= \lambda s.\lambda z.s\,(s\,z) \\
\mathtt{3} &:= \lambda s.\lambda z.s\,(s\,(s\,z))
\end{aligned}
$$

Representing Natural Numbers

- Church numerals:

$$
\begin{aligned}
\mathtt{0} &:= \lambda s.\lambda z.z \\
\mathtt{1} &:= \lambda s.\lambda z.s\,z \\
\mathtt{2} &:= \lambda s.\lambda z.s\,(s\,z) \\
\mathtt{3} &:= \lambda s.\lambda z.s\,(s\,(s\,z)) \\
\mathtt{n} &:= \lambda s.\lambda z.s^n\,z
\end{aligned}
$$

Representing Natural Numbers

- Church numerals:

$$\begin{array}{lcl} \mathtt{0} & := & \lambda s.\lambda z.z \\ \mathtt{1} & := & \lambda s.\lambda z.s\,z \\ \mathtt{2} & := & \lambda s.\lambda z.s\,(s\,z) \\ \mathtt{3} & := & \lambda s.\lambda z.s\,(s\,(s\,z)) \\ \mathtt{n} & := & \lambda s.\lambda z.s^n\,z \end{array}$$

- Some operations:

$$\mathtt{add} \quad := \quad \lambda M.\lambda N.\lambda s.\lambda z.N\,s\,(M\,s\,z)$$

Representing Natural Numbers

- Church numerals:

$$
\begin{array}{lcl}
\mathtt{0} & := & \lambda s.\lambda z.z \\
\mathtt{1} & := & \lambda s.\lambda z.s\,z \\
\mathtt{2} & := & \lambda s.\lambda z.s\,(s\,z) \\
\mathtt{3} & := & \lambda s.\lambda z.s\,(s\,(s\,z)) \\
\mathtt{n} & := & \lambda s.\lambda z.s^n\,z
\end{array}
$$

- Some operations:

$$
\begin{array}{lcl}
\mathtt{add} & := & \lambda M.\lambda N.\lambda s.\lambda z.N\,s\,(M\,s\,z) \\
\mathtt{mult} & := & \lambda M.\lambda N.\lambda s.\lambda z.N\,(M\,s)\,z
\end{array}
$$

Representing Natural Numbers

- Church numerals:

$$
\begin{aligned}
\mathtt{0} &:= \lambda s.\lambda z.z \\
\mathtt{1} &:= \lambda s.\lambda z.s\,z \\
\mathtt{2} &:= \lambda s.\lambda z.s\,(s\,z) \\
\mathtt{3} &:= \lambda s.\lambda z.s\,(s\,(s\,z)) \\
\mathtt{n} &:= \lambda s.\lambda z.s^n\,z
\end{aligned}
$$

- Some operations:

$$
\begin{aligned}
\mathtt{add} &:= \lambda M.\lambda N.\lambda s.\lambda z.N\,s\,(M\,s\,z) \\
\mathtt{mult} &:= \lambda M.\lambda N.\lambda s.\lambda z.N\,(M\,s)\,z \\
\mathtt{pred} &:= \lambda n.\lambda s.\lambda z.n\,(\lambda g.\lambda h.h\,(g\,s))(\lambda u.z)(\lambda u.u)
\end{aligned}
$$

Representing Natural Numbers

- Church numerals:

$$
\begin{aligned}
\mathtt{0} &:= \lambda s.\lambda z.z \\
\mathtt{1} &:= \lambda s.\lambda z.s\,z \\
\mathtt{2} &:= \lambda s.\lambda z.s\,(s\,z) \\
\mathtt{3} &:= \lambda s.\lambda z.s\,(s\,(s\,z)) \\
\mathtt{n} &:= \lambda s.\lambda z.s^{n}\,z
\end{aligned}
$$

- Some operations:

$$
\begin{aligned}
\mathtt{add} &:= \lambda M.\lambda N.\lambda s.\lambda z.N\,s\,(M\,s\,z) \\
\mathtt{mult} &:= \lambda M.\lambda N.\lambda s.\lambda z.N\,(M\,s)\,z \\
\mathtt{pred} &:= \lambda n.\lambda s.\lambda z.n\,(\lambda g.\lambda h.h\,(g\,s))(\lambda u.z)(\lambda u.u) \\
\mathtt{subtr} &:= \lambda m.\lambda n.n\,\mathrm{pred}\,m
\end{aligned}
$$

Representing Natural Numbers

- Church numerals:

$$
\begin{array}{lcl}
\mathtt{0} & := & \lambda s.\lambda z.z \\
\mathtt{1} & := & \lambda s.\lambda z.s\,z \\
\mathtt{2} & := & \lambda s.\lambda z.s\,(s\,z) \\
\mathtt{3} & := & \lambda s.\lambda z.s\,(s\,(s\,z)) \\
\mathtt{n} & := & \lambda s.\lambda z.s^{n}\,z
\end{array}
$$

- Some operations:

$$
\begin{array}{lcl}
\mathtt{add} & := & \lambda M.\lambda N.\lambda s.\lambda z.N\,s\,(M\,s\,z) \\
\mathtt{mult} & := & \lambda M.\lambda N.\lambda s.\lambda z.N\,(M\,s)\,z \\
\mathtt{pred} & := & \lambda n.\lambda s.\lambda z.n\,(\lambda g.\lambda h.h\,(g\,s))(\lambda u.z)(\lambda u.u) \\
\mathtt{subtr} & := & \lambda m.\lambda n.n\,\mathrm{pred}\,m \\
\mathtt{isZero} & := & \lambda n.n(\lambda x.\mathrm{false})\,\mathrm{true}
\end{array}
$$

Representing Natural Numbers

- Church numerals:

$$\begin{array}{lcl} \mathtt{0} & := & \lambda s.\lambda z.z \\ \mathtt{1} & := & \lambda s.\lambda z.s\,z \\ \mathtt{2} & := & \lambda s.\lambda z.s\,(s\,z) \\ \mathtt{3} & := & \lambda s.\lambda z.s\,(s\,(s\,z)) \\ \mathtt{n} & := & \lambda s.\lambda z.s^n\,z \end{array}$$

- Some operations:

$$\begin{array}{lcl} \mathtt{add} & := & \lambda M.\lambda N.\lambda s.\lambda z.N\,s\,(M\,s\,z) \\ \mathtt{mult} & := & \lambda M.\lambda N.\lambda s.\lambda z.N\,(M\,s)\,z \\ \mathtt{pred} & := & \lambda n.\lambda s.\lambda z.n\,(\lambda g.\lambda h.h\,(g\,s))(\lambda u.z)(\lambda u.u) \\ \mathtt{subtr} & := & \lambda m.\lambda n.n\,\mathrm{pred}\,m \\ \mathtt{isZero} & := & \lambda n.n(\lambda x.\mathrm{false})\,\mathrm{true} \\ \mathtt{leq} & := & \lambda m.\lambda n.\mathtt{isZero}\,(\mathtt{subtr}\;m\;n) \end{array}$$

Representing Natural Numbers

- Church numerals:

$$
\begin{array}{lcl}
\mathtt{0} & := & \lambda s.\lambda z.z \\
\mathtt{1} & := & \lambda s.\lambda z.s\,z \\
\mathtt{2} & := & \lambda s.\lambda z.s\,(s\,z) \\
\mathtt{3} & := & \lambda s.\lambda z.s\,(s\,(s\,z)) \\
\mathtt{n} & := & \lambda s.\lambda z.s^n\,z
\end{array}
$$

- Some operations:

$$
\begin{array}{lcl}
\mathtt{add} & := & \lambda M.\lambda N.\lambda s.\lambda z.N\,s\,(M\,s\,z) \\
\mathtt{mult} & := & \lambda M.\lambda N.\lambda s.\lambda z.N\,(M\,s)\,z \\
\mathtt{pred} & := & \lambda n.\lambda s.\lambda z.n\,(\lambda g.\lambda h.h\,(g\,s))(\lambda u.z)(\lambda u.u) \\
\mathtt{subtr} & := & \lambda m.\lambda n.n\,\mathrm{pred}\,m \\
\mathtt{isZero} & := & \lambda n.n(\lambda x.\mathrm{false})\,\mathrm{true} \\
\mathtt{leq} & := & \lambda m.\lambda n.\mathtt{isZero}\,(\mathtt{subtr}\,m\,n) \\
\mathtt{eq} & := & \lambda m.\lambda n.\mathrm{and}\,(\mathtt{leq}\,m\,n)\,(\mathtt{leq}\,n\,m)
\end{array}
$$

Example (addition)

$$
\begin{aligned}
\texttt{add 2 3} &:= \lambda s.\lambda z.(\lambda s.\lambda z.s\,s\,s\,z)\,s\,((\lambda s.\lambda z.s\,s\,z)\,s\,z) \\
&=_\beta \lambda s.\lambda z.(\lambda z.s\,s\,s\,z)((\lambda z.s\,s\,z)\,z) \\
&=_\beta \lambda s.\lambda z.s\,s\,s\,((\lambda z.s\,s\,z)\,z) \\
&=_\beta \lambda s.\lambda z.s\,s\,s\,s\,s\,z
\end{aligned}
$$

Example (addition)

$$
\begin{aligned}
\texttt{add 2 3} &:= \lambda s.\lambda z.(\lambda s.\lambda z.sssz)\,s\,((\lambda s.\lambda z.ssz)\,s\,z) \\
&=_\beta \lambda s.\lambda z.(\lambda z.sssz)((\lambda z.ssz)\,z) \\
&=_\beta \lambda s.\lambda z.sss\,((\lambda z.ssz)\,z) \\
&=_\beta \lambda s.\lambda z.sssssz
\end{aligned}
$$

$$
\begin{aligned}
\texttt{mult 3 2} &:= \lambda s.\lambda z.(\lambda s.\lambda z.ssz)\,((\lambda s.\lambda z.sssz)\,s)\,z \\
&=_\beta \lambda s.\lambda z.(\lambda z.((\lambda s.\lambda z.sssz)s)\,((\lambda s.\lambda z.sssz)\,s)\,z)\,z \\
&=_\beta \lambda s.\lambda z.((\lambda s.\lambda z.sssz)s)\,((\lambda s.\lambda z.sssz)\,s)\,z \\
&=_\beta \lambda s.\lambda z.(\lambda z.sssz)\,(\lambda z.sssz)\,z \\
&=_\beta \lambda s.\lambda z.sss\,(\lambda z.sssz)\,z \\
&=_\beta \lambda s.\lambda z.ssssssz
\end{aligned}
$$

Representing Pairs & Tuples

- Pairs

$$\texttt{pair} \quad := \quad \lambda e_1.\lambda e_2.\lambda p.p\; e_1\; e_2$$

Representing Pairs & Tuples

- Pairs

$$\texttt{pair} \quad := \quad \lambda e_1.\lambda e_2.\lambda p.p\; e_1\; e_2$$

- Projections

$$\begin{aligned} \texttt{proj}_1 &\quad := \quad \lambda u.u\; \texttt{true} \\ \texttt{proj}_2 &\quad := \quad \lambda u.u\; \texttt{false} \end{aligned}$$

Representing Pairs & Tuples

- Pairs

$$\texttt{pair} \quad := \quad \lambda e_1.\lambda e_2.\lambda p.p\, e_1\, e_2$$

- Projections

$$\texttt{proj}_1 \quad := \quad \lambda u.u\ \texttt{true}$$
$$\texttt{proj}_2 \quad := \quad \lambda u.u\ \texttt{false}$$

- Tuples

$$\texttt{tuple} \quad := \quad \lambda e_1.\cdots.\lambda e_n.\lambda p.p\, e_1\, \cdots\, e_n$$

Representing Pairs & Tuples

- Pairs

$$\texttt{pair} \quad := \quad \lambda e_1.\lambda e_2.\lambda p.p\ e_1\ e_2$$

- Projections

$$\begin{aligned} \texttt{proj}_1 &\quad := \quad \lambda u.u\ \texttt{true} \\ \texttt{proj}_2 &\quad := \quad \lambda u.u\ \texttt{false} \end{aligned}$$

- Tuples

$$\texttt{tuple} \quad := \quad \lambda e_1.\cdots.\lambda e_n.\lambda p.p\ e_1\ \cdots\ e_n$$

- i^{th} projection

$$\texttt{proj}_i \quad := \quad \lambda u.u(\lambda x_1.\cdots.\lambda x_n.x_i)$$

Representing Lists

- constructors: cons and nil

$$\texttt{cons} \quad := \quad \lambda t_1.\lambda t_2.\texttt{pair}\ \texttt{false}\ (\texttt{pair}\ t_1\ t_2)$$

Representing Lists

- constructors: cons and nil

$$\begin{aligned} \texttt{cons} &:= \lambda t_1.\lambda t_2.\texttt{pair}\ \texttt{false}\ (\texttt{pair}\ t_1\ t_2) \\ \texttt{nil} &:= \lambda l.l \end{aligned}$$

Representing Lists

- constructors: cons and nil

$$\begin{aligned} \texttt{cons} &:= \lambda t_1.\lambda t_2.\texttt{pair false}\,(\texttt{pair}\ t_1\ t_2) \\ \texttt{nil} &:= \lambda l.l \end{aligned}$$

- head, tail and nullary check

$$\texttt{head} := \lambda l.\texttt{proj}_1\,(\texttt{proj}_2\ l)$$

Representing Lists

- constructors: cons and nil

$$\begin{array}{lcl} \texttt{cons} & := & \lambda t_1.\lambda t_2.\texttt{pair}\ \texttt{false}\ (\texttt{pair}\ t_1\ t_2) \\ \texttt{nil} & := & \lambda l.l \end{array}$$

- head, tail and nullary check

$$\begin{array}{lcl} \texttt{head} & := & \lambda l.\texttt{proj}_1\ (\texttt{proj}_2\ l) \\ \texttt{tail} & := & \lambda l.\texttt{proj}_2\ (\texttt{proj}_2\ l) \end{array}$$

Representing Lists

- constructors: cons and nil

$$\begin{aligned} \texttt{cons} &:= \lambda t_1.\lambda t_2.\texttt{pair}\ \texttt{false}\ (\texttt{pair}\ t_1\ t_2) \\ \texttt{nil} &:= \lambda l.l \end{aligned}$$

- head, tail and nullary check

$$\begin{aligned} \texttt{head} &:= \lambda l.\texttt{proj}_1\ (\texttt{proj}_2\ l) \\ \texttt{tail} &:= \lambda l.\texttt{proj}_2\ (\texttt{proj}_2\ l) \\ \texttt{isNil} &:= \texttt{proj}_1 \end{aligned}$$

Encoding Recursion: the $\mathcal{Y}$ Combinator

- to encode recursion, we are looking for a combinator that, given an argument some function F, would not only reproduce itself but also pass F on itself.

Encoding Recursion: the $\mathcal{Y}$ Combinator

- to encode recursion, we are looking for a combinator that, given an argument some function F, would not only reproduce itself but also pass F on itself.
- we can then define:

$$\mathcal{Y} := \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

Encoding Recursion: the $\mathcal{Y}$ Combinator

- to encode recursion, we are looking for a combinator that, given an argument some function F, would not only reproduce itself but also pass F on itself.
- we can then define:

$$\mathcal{Y} := \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

- let us observe what happens when we pass a function F to the $\mathcal{Y}$ combinator:

$$\mathcal{Y}\,F \quad := \quad \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))\,F$$

Encoding Recursion: the $\mathcal{Y}$ Combinator

- to encode recursion, we are looking for a combinator that, given an argument some function F, would not only reproduce itself but also pass F on itself.

- we can then define:

$$\mathcal{Y} := \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

- let us observe what happens when we pass a function F to the $\mathcal{Y}$ combinator:

$$\begin{aligned} \mathcal{Y}\,F \quad &:= \quad \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))\,F \\ &=_\beta \quad (\lambda x.F\,(x\,x))\,(\lambda x.F\,(x\,x)) \end{aligned}$$

Encoding Recursion: the $\mathcal{Y}$ Combinator

- to encode recursion, we are looking for a combinator that, given an argument some function F, would not only reproduce itself but also pass F on itself.
- we can then define:

$$\mathcal{Y} \;:=\; \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

- let us observe what happens when we pass a function F to the $\mathcal{Y}$ combinator:

$$\begin{aligned}
\mathcal{Y}\,F \quad &:= \quad \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))\,F \\
&=_\beta \quad (\lambda x.F\,(x\,x))\,(\lambda x.F\,(x\,x)) \\
&=_\beta \quad F\big(\underbrace{(\lambda x.F\,(x\,x))\,(\lambda x.F\,(x\,x))}_{\mathcal{Y}\,F}\big) = F\,(\mathcal{Y}\,F)
\end{aligned}$$

Encoding Recursion: the $\mathcal{Y}$ Combinator

- to encode recursion, we are looking for a combinator that, given an argument some function F, would not only reproduce itself but also pass F on itself.
- we can then define:

$$\mathcal{Y} := \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

- let us observe what happens when we pass a function F to the $\mathcal{Y}$ combinator:

$$\begin{aligned}
\mathcal{Y}\,F &:= \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))\,F \\
&=_\beta (\lambda x.F\,(x\,x))\,(\lambda x.F\,(x\,x)) \\
&=_\beta F\Big(\underbrace{(\lambda x.F\,(x\,x))\,(\lambda x.F\,(x\,x))}_{\mathcal{Y}\,F}\Big) = F\,(\mathcal{Y}\,F) \\
&=_\beta F\,(F\,(\mathcal{Y}\,F))
\end{aligned}$$

Encoding Recursion: the $\mathcal{Y}$ Combinator

- to encode recursion, we are looking for a combinator that, given an argument some function F, would not only reproduce itself but also pass F on itself.

- we can then define:

$$\mathcal{Y} := \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

- let us observe what happens when we pass a function F to the $\mathcal{Y}$ combinator:

$$\begin{aligned}
\mathcal{Y}\,F \quad &:= \quad \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))\,F \\
&=_\beta \quad (\lambda x.F\,(x\,x))\,(\lambda x.F\,(x\,x)) \\
&=_\beta \quad F\Big(\underbrace{(\lambda x.F\,(x\,x))\,(\lambda x.F\,(x\,x))}_{\mathcal{Y}\,F}\Big) = F\,(\mathcal{Y}\,F) \\
&=_\beta \quad F\,(F\,(\mathcal{Y}\,F)) \\
&=_\beta \quad F\,(F\,(F\,(\mathcal{Y}\,F)))
\end{aligned}$$

Encoding Recursion: the $\mathcal{Y}$ Combinator

- to encode recursion, we are looking for a combinator that, given an argument some function F, would not only reproduce itself but also pass F on itself.

- we can then define:

$$\mathcal{Y} := \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

- let us observe what happens when we pass a function F to the $\mathcal{Y}$ combinator:

$$\begin{aligned}
\mathcal{Y}\,F &:= \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))\,F \\
&=_\beta (\lambda x.F\,(x\,x))\,(\lambda x.F\,(x\,x)) \\
&=_\beta F\Big(\underbrace{(\lambda x.F\,(x\,x))\,(\lambda x.F\,(x\,x))}_{\mathcal{Y}\,F}\Big) = F\,(\mathcal{Y}\,F) \\
&=_\beta F\,(F\,(\mathcal{Y}\,F)) \\
&=_\beta F\,(F\,(F\,(\mathcal{Y}\,F))) \\
&=_\beta \ldots
\end{aligned}$$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

$$\mathcal{Y} F\, 3 \quad =_\beta^+ \quad F\,(\mathcal{Y} F)\, 3$$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

$$\begin{aligned}
\mathcal{Y}\,F\,3 \quad &=^{+}_{\beta} \quad F\,(\mathcal{Y}\,F)\,3 \\
&:= \quad \lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}\,F)\,3
\end{aligned}$$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

$$\begin{aligned}
\mathcal{Y}F\,3 \quad &=^{+}_{\beta} \quad F\,(\mathcal{Y}F)\,3 \\
&:= \quad \lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}F)\,3 \\
&=_{\beta} \quad \lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}F)\,(x-1))\,3
\end{aligned}$$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

$$
\begin{aligned}
\mathcal{Y}F\,3 \quad &=_\beta^+ \quad F\,(\mathcal{Y}F)\,3 \\
&:= \quad \lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}F)\,3 \\
&=_\beta \quad \lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}F)\,(x-1))\,3 \\
&=_\beta \quad \text{if } 3 == 0 \text{ then } 1 \text{ else } 3 * (\mathcal{Y}F)\,(3-1)
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

$$
\begin{aligned}
\mathcal{Y}\,F\,3 &=_\beta^+ F\,(\mathcal{Y}\,F)\,3 \\
&:= \lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}\,F)\,3 \\
&=_\beta \lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}\,F)\,(x-1))\,3 \\
&=_\beta \text{if } 3 == 0 \text{ then } 1 \text{ else } 3 * (\mathcal{Y}\,F)\,(3-1) \\
&=_\beta 3 * (\mathcal{Y}\,F)\,2
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

$$
\begin{aligned}
\mathcal{Y} F\, 3 \quad & =_\beta^+ \quad F\,(\mathcal{Y} F)\, 3 \\
& := \quad \lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y} F)\, 3 \\
& =_\beta \quad \lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y} F)\,(x-1))\, 3 \\
& =_\beta \quad \text{if } 3 == 0 \text{ then } 1 \text{ else } 3 * (\mathcal{Y} F)\,(3-1) \\
& =_\beta \quad 3 * (\mathcal{Y} F)\, 2 \\
& =_\beta^+ \quad 3 * F\,(\mathcal{Y} F)\, 2
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

$$
\begin{array}{lll}
\mathcal{Y}\,F\,3 & =_\beta^+ & F\,(\mathcal{Y}\,F)\,3 \\
 & := & \lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}\,F)\,3 \\
 & =_\beta & \lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}\,F)\,(x-1))\,3 \\
 & =_\beta & \text{if } 3 == 0 \text{ then } 1 \text{ else } 3 * (\mathcal{Y}\,F)\,(3-1) \\
 & =_\beta & 3 * (\mathcal{Y}\,F)\,2 \\
 & =_\beta^+ & 3 * F\,(\mathcal{Y}\,F)\,2 \\
 & =_\beta & 3 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}\,F)\,2)
\end{array}
$$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

$$
\begin{aligned}
\mathcal{Y}F\,3 &=_\beta^+ & & F\,(\mathcal{Y}F)\,3 \\
&:= & & \lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}F)\,3 \\
&=_\beta & & \lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}F)\,(x-1))\,3 \\
&=_\beta & & \text{if } 3 == 0 \text{ then } 1 \text{ else } 3 * (\mathcal{Y}F)\,(3-1) \\
&=_\beta & & 3 * (\mathcal{Y}F)\,2 \\
&=_\beta^+ & & 3 * F\,(\mathcal{Y}F)\,2 \\
&=_\beta & & 3 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}F)\,2) \\
&=_\beta & & 3 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}F)\,(x-1))\,2)
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

$$
\begin{aligned}
\mathcal{Y} F\, 3 &=_\beta^+ F\,(\mathcal{Y} F)\, 3 \\
&:= \lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y} F)\, 3 \\
&=_\beta \lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y} F)\,(x-1))\, 3 \\
&=_\beta \text{if } 3 == 0 \text{ then } 1 \text{ else } 3 * (\mathcal{Y} F)\,(3-1) \\
&=_\beta 3 * (\mathcal{Y} F)\, 2 \\
&=_\beta^+ 3 * F\,(\mathcal{Y} F)\, 2 \\
&=_\beta 3 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y} F)\, 2) \\
&=_\beta 3 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y} F)\,(x-1))\, 2) \\
&=_\beta 3 * (\text{if } 2 == 0 \text{ then } 1 \text{ else } 2 * (\mathcal{Y} F)\,(2-1))
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

$$
\begin{aligned}
\mathcal{Y} F\, 3 \quad &=_\beta^+ \quad F\,(\mathcal{Y} F)\, 3 \\
&:= \quad \lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y} F)\, 3 \\
&=_\beta \quad \lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y} F)\,(x-1))\, 3 \\
&=_\beta \quad \text{if } 3 == 0 \text{ then } 1 \text{ else } 3 * (\mathcal{Y} F)\,(3-1) \\
&=_\beta \quad 3 * (\mathcal{Y} F)\, 2 \\
&=_\beta^+ \quad 3 * F\,(\mathcal{Y} F)\, 2 \\
&=_\beta \quad 3 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y} F)\, 2) \\
&=_\beta \quad 3 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y} F)\,(x-1))\, 2) \\
&=_\beta \quad 3 * (\text{if } 2 == 0 \text{ then } 1 \text{ else } 2 * (\mathcal{Y} F)\,(2-1)) \\
&=_\beta \quad 3 * 2 * (\mathcal{Y} F)\, 1
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

$$\begin{aligned}
\mathcal{Y}\,F\,3 &=_\beta^+ F\,(\mathcal{Y}\,F)\,3 \\
&:= \lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}\,F)\,3 \\
&=_\beta \lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}\,F)\,(x-1))\,3 \\
&=_\beta \text{if } 3 == 0 \text{ then } 1 \text{ else } 3 * (\mathcal{Y}\,F)\,(3-1) \\
&=_\beta 3 * (\mathcal{Y}\,F)\,2 \\
&=_\beta^+ 3 * F\,(\mathcal{Y}\,F)\,2 \\
&=_\beta 3 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}\,F)\,2) \\
&=_\beta 3 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}\,F)\,(x-1))\,2) \\
&=_\beta 3 * (\text{if } 2 == 0 \text{ then } 1 \text{ else } 2 * (\mathcal{Y}\,F)\,(2-1)) \\
&=_\beta 3 * 2 * (\mathcal{Y}\,F)\,1 \\
&=_\beta 6 * (\mathcal{Y}\,F)\,1
\end{aligned}$$

Example ($\mathcal{Y}$ Combinator)

Let F be $\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f(x-1))$

$$
\begin{aligned}
\mathcal{Y}F\,3 \quad &=_\beta^+ \quad F\,(\mathcal{Y}F)\,3 \\
&:= \quad \lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}F)\,3 \\
&=_\beta \quad \lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}F)\,(x-1))\,3 \\
&=_\beta \quad \text{if } 3 == 0 \text{ then } 1 \text{ else } 3 * (\mathcal{Y}F)\,(3-1) \\
&=_\beta \quad 3 * (\mathcal{Y}F)\,2 \\
&=_\beta^+ \quad 3 * F\,(\mathcal{Y}F)\,2 \\
&=_\beta \quad 3 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}F)\,2) \\
&=_\beta \quad 3 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}F)\,(x-1))\,2) \\
&=_\beta \quad 3 * (\text{if } 2 == 0 \text{ then } 1 \text{ else } 2 * (\mathcal{Y}F)\,(2-1)) \\
&=_\beta \quad 3 * 2 * (\mathcal{Y}F)\,1 \\
&=_\beta \quad 6 * (\mathcal{Y}F)\,1 \\
&=_\beta^+ \quad 6 * F\,(\mathcal{Y}F)\,1
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator (cont'd))

$$6 * F\,(\mathcal{Y}\,F)\,1$$

Example ($\mathcal{Y}$ Combinator (cont'd))

$$
\begin{aligned}
& 6 * F\,(\mathcal{Y}\,F)\,1 \\
=_\beta\quad & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}\,F)\,1)
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator (cont'd))

$$
\begin{aligned}
& 6 * F\,(\mathcal{Y}\,F)\,1 \\
=_\beta\ & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}\,F)\,1) \\
=_\beta\ & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}\,F)\,(x-1))\,1)
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator (cont'd))

$$
\begin{aligned}
& 6 * F\,(\mathcal{Y} F)\,1 \\
=_\beta\ & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y} F)\,1) \\
=_\beta\ & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y} F)\,(x-1))\,1) \\
=_\beta\ & 6 * (\text{if } 1 == 0 \text{ then } 1 \text{ else } 1 * (\mathcal{Y} F)\,(1-1))
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator (cont'd))

$$
\begin{aligned}
& 6 * F\,(\mathcal{Y}F)\,1 \\
=_\beta\quad & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}F)\,1) \\
=_\beta\quad & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}F)\,(x-1))\,1) \\
=_\beta\quad & 6 * (\text{if } 1 == 0 \text{ then } 1 \text{ else } 1 * (\mathcal{Y}F)\,(1-1)) \\
=_\beta\quad & 6 * (\mathcal{Y}F)\,0
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator (cont'd))

$$
\begin{aligned}
& 6 * F\,(\mathcal{Y}F)\,1 \\
=_\beta\quad & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}F)\,1) \\
=_\beta\quad & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}F)\,(x-1))\,1) \\
=_\beta\quad & 6 * (\text{if } 1 == 0 \text{ then } 1 \text{ else } 1 * (\mathcal{Y}F)\,(1-1)) \\
=_\beta\quad & 6 * (\mathcal{Y}F)\,0 \\
=_\beta^+\quad & 6 * F\,(\mathcal{Y}F)\,0
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator (cont'd))

$$
\begin{aligned}
& 6 * F\,(\mathcal{Y}F)\,1 \\
=_\beta\; & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}F)\,1) \\
=_\beta\; & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}F)\,(x-1))\,1) \\
=_\beta\; & 6 * (\text{if } 1 == 0 \text{ then } 1 \text{ else } 1 * (\mathcal{Y}F)\,(1-1)) \\
=_\beta\; & 6 * (\mathcal{Y}F)\,0 \\
=_\beta^+\; & 6 * F\,(\mathcal{Y}F)\,0 \\
=_\beta\; & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}F)\,0)
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator (cont'd))

$$
\begin{aligned}
& 6 * F\,(\mathcal{Y} F)\,1 \\
=_\beta\; & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y} F)\,1) \\
=_\beta\; & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y} F)\,(x-1))\,1) \\
=_\beta\; & 6 * (\text{if } 1 == 0 \text{ then } 1 \text{ else } 1 * (\mathcal{Y} F)\,(1-1)) \\
=_\beta\; & 6 * (\mathcal{Y} F)\,0 \\
=_\beta^+\; & 6 * F\,(\mathcal{Y} F)\,0 \\
=_\beta\; & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y} F)\,0) \\
=_\beta\; & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y} F)\,(x-1))\,0)
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator (cont'd))

$$
\begin{aligned}
& 6 * F\,(\mathcal{Y}F)\,1 \\
=_\beta\; & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}F)\,1) \\
=_\beta\; & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}F)\,(x-1))\,1) \\
=_\beta\; & 6 * (\text{if } 1 == 0 \text{ then } 1 \text{ else } 1 * (\mathcal{Y}F)\,(1-1)) \\
=_\beta\; & 6 * (\mathcal{Y}F)\,0 \\
=_\beta^+\; & 6 * F\,(\mathcal{Y}F)\,0 \\
=_\beta\; & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}F)\,0) \\
=_\beta\; & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}F)\,(x-1))\,0) \\
=_\beta\; & 6 * (\text{if } 0 == 0 \text{ then } 1 \text{ else } 1 * (\mathcal{Y}F)\,(0-1))
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator (cont'd))

$$
\begin{aligned}
& 6 * F\,(\mathcal{Y} F)\,1 \\
=_\beta\quad & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y} F)\,1) \\
=_\beta\quad & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y} F)\,(x-1))\,1) \\
=_\beta\quad & 6 * (\text{if } 1 == 0 \text{ then } 1 \text{ else } 1 * (\mathcal{Y} F)\,(1-1)) \\
=_\beta\quad & 6 * (\mathcal{Y} F)\,0 \\
=_\beta^+\quad & 6 * F\,(\mathcal{Y} F)\,0 \\
=_\beta\quad & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y} F)\,0) \\
=_\beta\quad & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y} F)\,(x-1))\,0) \\
=_\beta\quad & 6 * (\text{if } 0 == 0 \text{ then } 1 \text{ else } 1 * (\mathcal{Y} F)\,(0-1)) \\
=_\beta\quad & 6 * 1
\end{aligned}
$$

Example ($\mathcal{Y}$ Combinator (cont'd))

$$
\begin{array}{ll}
 & 6 * F\,(\mathcal{Y}\,F)\,1 \\
=_\beta & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}\,F)\,1) \\
=_\beta & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}\,F)\,(x-1))\,1) \\
=_\beta & 6 * (\text{if } 1 == 0 \text{ then } 1 \text{ else } 1 * (\mathcal{Y}\,F)\,(1-1)) \\
=_\beta & 6 * (\mathcal{Y}\,F)\,0 \\
=_\beta^+ & 6 * F\,(\mathcal{Y}\,F)\,0 \\
=_\beta & 6 * (\lambda f.\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * f\,(x-1))\,(\mathcal{Y}\,F)\,0) \\
=_\beta & 6 * (\lambda x.(\text{if } x == 0 \text{ then } 1 \text{ else } x * (\mathcal{Y}\,F)\,(x-1))\,0) \\
=_\beta & 6 * (\text{if } 0 == 0 \text{ then } 1 \text{ else } 1 * (\mathcal{Y}\,F)\,(0-1)) \\
=_\beta & 6 * 1 \\
=_\beta & 6
\end{array}
$$

Thanks! & Questions?