

Lambda-Calculus

Notions of computability

- ▶ Church (1936): λ -calculus
- ▶ Turing (1936): Turing machines.

Turing showed that the two very different approaches determine the same class of computable functions.

Hence:

Church-Turing Thesis. Every algorithm [in intuitive sense of Lect. 1] can be realized as a Turing machine.

λ -Terms, M

are built up from a given, countable collection of

- ▶ variables x, y, z, \dots

by two operations for forming λ -terms:

- ▶ λ -abstraction: $(\lambda x.M)$
(where x is a variable and M is a λ -term)
- ▶ application: $(M M')$
(where M and M' are λ -terms).

λ -Terms, M

are built up from a given, countable collection of

- ▶ variables x, y, z, \dots

by two operations for forming λ -terms:

- ▶ λ -abstraction: $(\lambda x.M)$
(where x is a variable and M is a λ -term)
- ▶ application: $(M M')$
(where M and M' are λ -terms).

Some random examples of λ -terms:

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

λ -Terms, M

Notational conventions:

- ▶ $(\lambda x_1 x_2 \dots x_n. M)$ means $(\lambda x_1. (\lambda x_2 \dots (\lambda x_n. M) \dots))$
- ▶ $(M_1 M_2 \dots M_n)$ means $(\dots (M_1 M_2) \dots M_n)$
(i.e. application is left-associative)
- ▶ drop outermost parentheses and those enclosing the body of a λ -abstraction. E.g. write $(\lambda x. (x(\lambda y. (y x))))$ as $\lambda x. x(\lambda y. y x)$.
- ▶ $x \# M$ means that the variable x does not occur anywhere in the λ -term M .

Free and bound variables

In $\lambda x.M$, we call x the **bound variable** and M the **body** of the λ -abstraction.

An occurrence of x in a λ -term M is called

- ▶ **binding** if in between λ and $.$
(e.g. $(\lambda x.y x) x$)
- ▶ **bound** if in the body of a binding occurrence of x
(e.g. $(\lambda x.y x) x$)
- ▶ **free** if neither binding nor bound
(e.g. $(\lambda x.y x)x$).

Free and bound variables

Sets of **free** and **bound** variables:

$$\begin{aligned}FV(x) &= \{x\} \\FV(\lambda x.M) &= FV(M) - \{x\} \\FV(MN) &= FV(M) \cup FV(N) \\BV(x) &= \emptyset \\BV(\lambda x.M) &= BV(M) \cup \{x\} \\BV(MN) &= BV(M) \cup BV(N)\end{aligned}$$

If $FV(M) = \emptyset$, M is called a **closed term**, or **combinator**.

α -Equivalence $M =_{\alpha} M'$

$\lambda x.M$ is intended to represent the function f such that

$$f(x) = M \text{ for all } x.$$

So the name of the bound variable is immaterial: if $M' = M\{x'/x\}$ is the result of taking M and changing all occurrences of x to some variable $x' \# M$, then $\lambda x.M$ and $\lambda x'.M'$ both represent the same function.

For example, $\lambda x.x$ and $\lambda y.y$ represent the same function (the identity function).

α -Equivalence $M =_{\alpha} M'$

is the binary relation inductively generated by the rules:

$$\frac{}{x =_{\alpha} x} \quad \frac{z \# (M N) \quad M\{z/x\} =_{\alpha} N\{z/y\}}{\lambda x.M =_{\alpha} \lambda y.N}$$
$$\frac{M =_{\alpha} M' \quad N =_{\alpha} N'}{M N =_{\alpha} M' N'}$$

where $M\{z/x\}$ is M with all occurrences of x replaced by z .

α -Equivalence $M =_{\alpha} M'$

For example:

$$\lambda x. (\lambda x x'. x) x' =_{\alpha} \lambda y. (\lambda x x'. x) x'$$

because

$$(\lambda z x'. z) x' =_{\alpha} (\lambda x x'. x) x'$$

because

$$\lambda z x'. z =_{\alpha} \lambda x x'. x \text{ and } x' =_{\alpha} x'$$

because

$$\lambda x'. u =_{\alpha} \lambda x'. u \text{ and } x' =_{\alpha} x'$$

because

$$u =_{\alpha} u \text{ and } x' =_{\alpha} x'.$$

α -Equivalence $M =_{\alpha} M'$

Fact: $=_{\alpha}$ is an equivalence relation (reflexive, symmetric and transitive).

We do not care about the particular names of bound variables, just about the distinctions between them. So α -equivalence classes of λ -terms are more important than λ -terms themselves.

- ▶ Textbooks (and these lectures) suppress any notation for α -equivalence classes and refer to an equivalence class via a representative λ -term (look for phrases like “we identify terms up to α -equivalence” or “we work up to α -equivalence”).
- ▶ For implementations and computer-assisted reasoning, there are various devices for picking canonical representatives of α -equivalence classes (e.g. de Bruijn indexes, graphical representations, ...).

Substitution $N[M/x]$

$$x[M/x] = M$$

$$y[M/x] = y \quad \text{if } y \neq x$$

$$(\lambda y.N)[M/x] = \lambda y.N[M/x] \quad \text{if } y \# (M x)$$

$$(N_1 N_2)[M/x] = N_1[M/x] N_2[M/x]$$

Substitution $N[M/x]$

$$x[M/x] = M$$

$$y[M/x] = y \quad \text{if } y \neq x$$

$$(\lambda y.N)[M/x] = \lambda y.N[M/x] \quad \text{if } y \# (Mx)$$

$$(N_1 N_2)[M/x] = N_1[M/x] N_2[M/x]$$

Side-condition $y \# (Mx)$ (y does not occur in M and $y \neq x$) makes substitution “capture-avoiding”.

E.g. if $x \neq y$

$$(\lambda y.x)[y/x] \neq \lambda y.y$$

Substitution $N[M/x]$

$$x[M/x] = M$$

$$y[M/x] = y \quad \text{if } y \neq x$$

$$(\lambda y.N)[M/x] = \lambda y.N[M/x] \quad \text{if } y \# (Mx)$$

$$(N_1 N_2)[M/x] = N_1[M/x] N_2[M/x]$$

Side-condition $y \# (Mx)$ (y does not occur in M and $y \neq x$) makes substitution “capture-avoiding”.

E.g. if $x \neq y \neq z \neq x$

$$(\lambda y.x)[y/x] =_{\alpha} (\lambda z.x)[y/x] = \lambda z.y$$

$N \mapsto N[M/x]$ induces a total operation on α -equivalence classes.

β -Reduction

Recall that $\lambda x.M$ is intended to represent the function f such that $f(x) = M$ for all x . We can regard $\lambda x.M$ as a function on λ -terms via substitution: map each N to $M[N/x]$.

So the natural notion of computation for λ -terms is given by stepping from a

β -redex $(\lambda x.M)N$

to the corresponding

β -reduct $M[N/x]$

β -Reduction

One-step β -reduction, $M \rightarrow M'$:

$$\frac{}{(\lambda x.M)N \rightarrow M[N/x]}$$

$$\frac{M \rightarrow M'}{\lambda x.M \rightarrow \lambda x.M'}$$

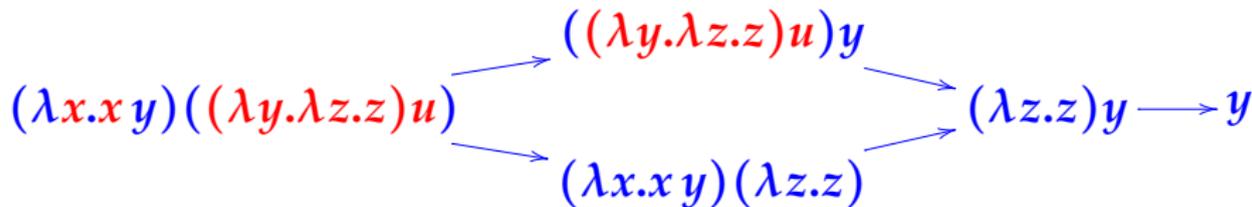
$$\frac{M \rightarrow M'}{MN \rightarrow M'N}$$

$$\frac{M \rightarrow M'}{NM \rightarrow NM'}$$

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

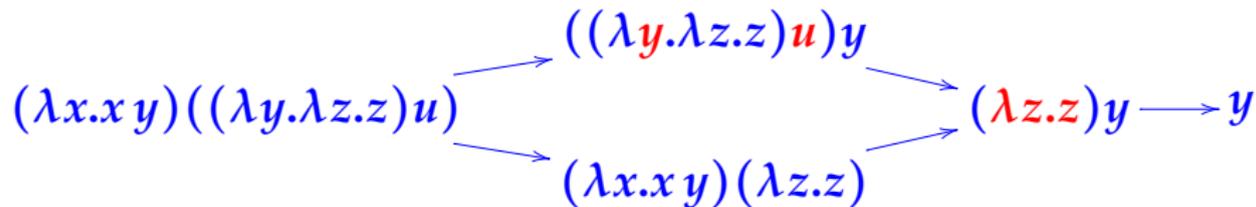
β -Reduction

E.g.



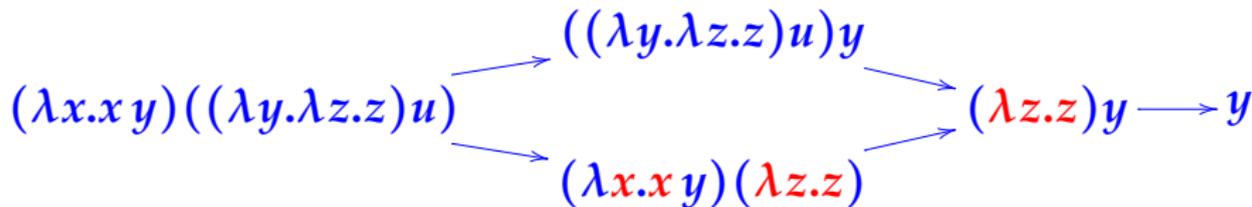
β -Reduction

E.g.



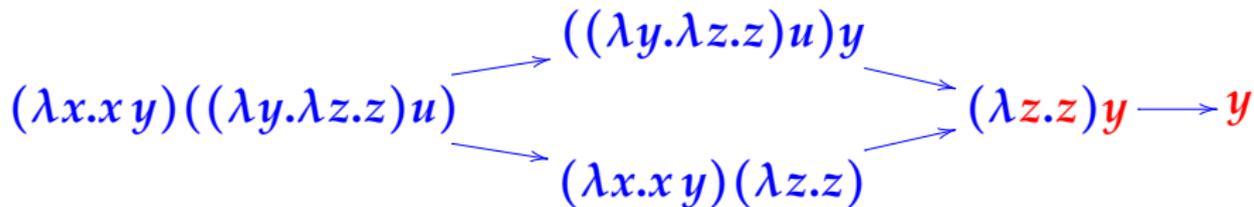
β -Reduction

E.g.



β -Reduction

E.g.



Many-step β -reduction, $M \twoheadrightarrow M'$:

$$\frac{M =_{\alpha} M'}{M \twoheadrightarrow M'}$$

(no steps)

$$\frac{M \rightarrow M'}{M \twoheadrightarrow M'}$$

(1 step)

$$\frac{M \rightarrow M' \quad M' \rightarrow M''}{M \twoheadrightarrow M''}$$

(1 more step)

E.g.

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

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