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.

Notation for <u>function definitions</u> in mathematical discourse:

NAMED

" let f be the function $f(x) = x^2 + x + 1$ .... $[f]$ ... "

ANONYMOUS

" the function $x \mapsto x^2 + x + 1$ ... "

" the function $\underbrace{\lambda x .\ x^2 + x + 1}$ ... "

LAMBDA NOTATION

λ-Terms, M

are built up from a given, countable collection of

- variables $x, y, z, \ldots$

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:

- $(\lambda x_1\, x_2 \ldots x_n.M)$ means $(\lambda x_1.(\lambda x_2 \ldots (\lambda x_n.M) \ldots))$
- $(M_1\, M_2 \ldots M_n)$ means $(\ldots (M_1\, M_2) \ldots M_n)$ (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:

$$
\begin{aligned}
FV(x) &= \{x\}\\
FV(\lambda x.M) &= FV(M) - \{x\}\\
FV(M\,N) &= FV(M) \cup FV(N)\\[4pt]
BV(x) &= \emptyset\\
BV(\lambda x.M) &= BV(M) \cup \{x\}\\
BV(M\,N) &= BV(M) \cup BV(N)
\end{aligned}
$$

E.g. $FV((\lambda x.\, y x) x) = \{x, y\}$

$BV((\lambda x.\, y x) x) = \{x\}$

Free and bound variables

Sets of free and bound variables:

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

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

E.g. $FV(\lambda y.\lambda x.(\lambda x.y\,x)\,x) = \emptyset$

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:

$$\frac{}{x =_\alpha x} \qquad \frac{z \mathrel{\#} (M\,N) \qquad M\{z/x\} =_\alpha N\{z/y\}}{\lambda x.M =_\alpha \lambda y.N}$$

$$\frac{M =_\alpha M' \qquad N =_\alpha N'}{M\,N =_\alpha M'\,N'}$$

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

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

α-Equivalence $M =_\alpha M'$

For example:

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

because $\quad (\lambda z\, x'.z) x' =_\alpha (\lambda x\, x'.x) x'$

because $\quad \lambda z\, x'.z =_\alpha \lambda x\, x'.x$ and $x' =_\alpha x'$

because

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 $\lambda z\, x'.z =_\alpha \lambda x\, x'.x$ and $x' =_\alpha x'$

because $\lambda x'.u =_\alpha \lambda x'.u$ and $x' =_\alpha x'$

because

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 $\lambda z\, x'.z =_\alpha \lambda x\, x'.x$ and $x' =_\alpha x'$

because $\lambda x'.u =_\alpha \lambda x'.u$ and $x' =_\alpha x'$

because $u =_\alpha u$ 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, …).

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 $\qquad (\lambda x.M)N$

to the corresponding

β-reduct $\qquad M[N/x]$

Substitution $N[M/x]$

$$
\begin{array}{rcll}
x[M/x] & = & M & \\
y[M/x] & = & y & \text{if } y \neq x \\
(\lambda y.N)[M/x] & = & \lambda y.N[M/x] & \text{if } y \# (M\ x) \\
(N_1\ N_2)[M/x] & = & N_1[M/x]\ N_2[M/x] &
\end{array}
$$

$N[M/x]$ = result of replacing all <u>free</u> occurrences of x in N with M, avoiding "capture" of free variables in M by λ-binders in N

Substitution $N[M/x]$

$$
\begin{array}{rcll}
x[M/x] & = & M & \\
y[M/x] & = & y & \text{if } y \neq x \\
(\lambda y.N)[M/x] & = & \lambda y.N[M/x] & \text{if } y \# (M\,x) \\
(N_1\,N_2)[M/x] & = & N_1[M/x]\,N_2[M/x] &
\end{array}
$$

Side-condition $y \# (M\,x)$ (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 $N[M/x]$

$$
\begin{array}{rcll}
x[M/x] & = & M & \\
y[M/x] & = & y & \text{if } y \neq x \\
(\lambda y.N)[M/x] & = & \lambda y.N[M/x] & \text{if } y \# (M\ x) \\
(N_1\ N_2)[M/x] & = & N_1[M/x]\ N_2[M/x] &
\end{array}
$$

Can always satisfy this up to α-equivalence

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

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

In fact $N \mapsto N[M/x]$ induces a totally defined function from the set of α-equivalence classes of λ-terms to itself.

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

$$=$$

$\lambda x.(\lambda z.z)\,y\,x\,[\lambda z.y/y]$ no possible capture

$=$

$$\lambda x.\,(\lambda z.z)\,y\,x\left[\lambda z.y/y\right]$$

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

$$\lambda x.\,(\lambda u.u)\,x\,y\left[\lambda y.x/y\right]$$

$$=$$

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

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

$$\lambda x. (\lambda u. u) x y [\lambda y. x / y]$$ possible capture

$$=$$

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

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

$$\lambda x.\,(\lambda u.u)\,x\,y\,[\,\lambda y.x\,/\,y\,]$$ possible capture...

$$=_\alpha \lambda z.\,(\lambda u.u)\,z\,y\,[\,\lambda y.x\,/\,y\,]$$ ...α-convert to avoid

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

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

$$\lambda x.(\lambda u.u)\,x\,y\,[\lambda y.x/y]$$

possible capture...

$$=_\alpha \lambda z.(\lambda u.u)\,z\,y\,[\lambda y.x/y]$$

...α-convert to avoid

$$= \lambda z.(\lambda u.u)\,z(\lambda y.x)$$

β-Reduction

One-step β-reduction, $M \to M'$:

$$\frac{}{(\lambda x.M)N \to M[N/x]} \qquad \frac{M \to M'}{\lambda x.M \to \lambda x.M'}$$

$$\frac{M \to M'}{M\,N \to M'\,N} \qquad \frac{M \to M'}{N\,M \to N\,M'}$$

$$\frac{N =_\alpha M \qquad M \to M' \qquad M' =_\alpha N'}{N \to N'}$$

β-Reduction

E.g.

$$\begin{tikzcd}[column sep=small]
 & ((\lambda y.\lambda z.z)u)y \arrow[dr] & & \\
(\lambda x.x\,y)((\lambda y.\lambda z.z)u) \arrow[ur] \arrow[dr] & & (\lambda z.z)y \arrow[r] & y \\
 & (\lambda x.x\,y)(\lambda z.z) \arrow[ur] & &
\end{tikzcd}$$

β-Reduction

E.g.

$$\begin{tikzcd}[column sep=small]
 & ((\lambda y.\lambda z.z)u)y \arrow[dr] & & \\
(\lambda x.x\,y)((\lambda y.\lambda z.z)u) \arrow[ur] \arrow[dr] & & (\lambda z.z)y \arrow[r] & y \\
 & (\lambda x.x\,y)(\lambda z.z) \arrow[ur] & &
\end{tikzcd}$$

β-Reduction

E.g.

$$\begin{tikzcd}
 & ((\lambda y.\lambda z.z)\,u)\,y \arrow[dr] & & \\
(\lambda x.x\,y)((\lambda y.\lambda z.z)\,u) \arrow[ur] \arrow[dr] & & (\lambda z.z)\,y \arrow[r] & y \\
 & (\lambda x.x\,y)(\lambda z.z) \arrow[ur] & &
\end{tikzcd}$$

β-Reduction

E.g.

$$\begin{tikzcd}[column sep=small]
 & ((\lambda y.\lambda z.z)u)y \arrow[dr] & & \\
(\lambda x.x\,y)((\lambda y.\lambda z.z)u) \arrow[ur] \arrow[dr] & & (\lambda z.z)y \arrow[r] & y \\
 & (\lambda x.x\,y)(\lambda z.z) \arrow[ur] & &
\end{tikzcd}$$

β-Reduction

E.g.

$$\begin{tikzcd}[column sep=small]
 & ((\lambda y.\lambda z.z)u)y \arrow[dr] & & \\
(\lambda x.x\,y)((\lambda y.\lambda z.z)u) \arrow[ur] \arrow[dr] & & (\lambda z.z)y \arrow[r] & y \\
 & (\lambda x.x\,y)(\lambda z.z) \arrow[ur] & &
\end{tikzcd}$$

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

$$\frac{M =_\alpha M'}{M \twoheadrightarrow M'} \quad \left[\frac{M \to M'}{M \twoheadrightarrow M'}\right] \quad \frac{M \twoheadrightarrow M' \qquad M' \to M''}{M \twoheadrightarrow M''}$$

(no steps) (1 step) (1 more step)

E.g.

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

β-Conversion $M =_\beta N$

Informally: $M =_\beta N$ holds if N can be obtained from M by performing zero or more steps of α-equivalence, β-reduction, or β-*expansion* (= inverse of a reduction).

E.g. $u\,((\lambda x\,y.\,v\,x)y) =_\beta (\lambda x.\,u\,x)(\lambda x.\,v\,y)$

because $(\lambda x.\,u\,x)(\lambda x.\,v\,y) \to u(\lambda x.\,v\,y)$

and so we have

$$u\,((\lambda x\,y.\,v\,x)y) \quad =_\alpha \quad u\,((\lambda x\,y'.\,v\,x)y)$$

β-Conversion $M =_\beta N$

Informally: $M =_\beta N$ holds if N can be obtained from M by performing zero or more steps of α-equivalence, β-reduction, or β-*expansion* ($=$ inverse of a reduction).

E.g. $u\,((\lambda x\,y.\,v\,x)y) =_\beta (\lambda x.\,u\,x)(\lambda x.\,v\,y)$

because $(\lambda x.\,u\,x)(\lambda x.\,v\,y) \to u(\lambda x.\,v\,y)$

and so we have

$$\begin{array}{lcll} u\,((\lambda x\,y.\,v\,x)y) & =_\alpha & u\,((\lambda x\,y'.\,v\,x)y) & \\ & \to & u(\lambda y'.\,v\,y) & \text{reduction} \end{array}$$

β-Conversion $M =_\beta N$

Informally: $M =_\beta N$ holds if N can be obtained from M by performing zero or more steps of α-equivalence, β-reduction, or β-*expansion* ($=$ inverse of a reduction).

E.g. $u\,((\lambda x\,y.\,v\,x)y) =_\beta (\lambda x.\,u\,x)(\lambda x.\,v\,y)$

because $(\lambda x.\,u\,x)(\lambda x.\,v\,y) \to u(\lambda x.\,v\,y)$

and so we have

$$
\begin{array}{rcll}
u\,((\lambda x\,y.\,v\,x)y) & =_\alpha & u\,((\lambda x\,y'.\,v\,x)y) & \\
 & \to & u(\lambda y'.\,v\,y) & \text{reduction} \\
 & =_\alpha & u(\lambda x.\,v\,y) &
\end{array}
$$

β-Conversion $M =_\beta N$

Informally: $M =_\beta N$ holds if N can be obtained from M by performing zero or more steps of α-equivalence, β-reduction, or β-*expansion* ($=$ inverse of a reduction).

E.g. $u\,((\lambda x\,y.\,v\,x)y) =_\beta (\lambda x.\,u\,x)(\lambda x.\,v\,y)$

because $(\lambda x.\,u\,x)(\lambda x.\,v\,y) \to u(\lambda x.\,v\,y)$

and so we have

$$
\begin{array}{rcll}
u\,((\lambda x\,y.\,v\,x)y) & =_\alpha & u\,((\lambda x\,y'.\,v\,x)y) & \\
 & \to & u(\lambda y'.\,v\,y) & \text{reduction} \\
 & =_\alpha & u(\lambda x.\,v\,y) & \\
 & \leftarrow & (\lambda x.\,u\,x)(\lambda x.\,v\,y) & \text{expansion}
\end{array}
$$

β-Conversion $M =_\beta N$

is the binary relation inductively generated by the rules:

$$\frac{M =_\alpha M'}{M =_\beta M'} \qquad \frac{M \to M'}{M =_\beta M'} \qquad \frac{M =_\beta M'}{M' =_\beta M}$$

$$\frac{M =_\beta M' \qquad M' =_\beta M''}{M =_\beta M''} \qquad \frac{M =_\beta M'}{\lambda x.M =_\beta \lambda x.M'}$$

$$\frac{M =_\beta M' \qquad N =_\beta N'}{M\,N =_\beta M'\,N'}$$