Famous examples of rewriting systems

1. Lambda calculus
2. Combinatory logic

Lambda terms

- a variable $(x, y, z, ...)$ is a lambda term,
- if M is a lambda term, then $(\lambda x.M)$ is a lambda term (*functional abstraction*)
- if n and M are lambda terms then (MN) is a lambda term (*functional application*)

Notational conventions:

- Left-associativity of application

  $((xy)z) \quad \Rightarrow \quad xyz$
- Skipping multiple λs

  $(\lambda x.(\lambda y.(M))) \quad \Rightarrow \quad \lambda xy.M$

$$\lambda xy.xy(xy)$$

means

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

Intuition of rewriting

- $\lambda x.M$ is a function with parameter x: x *is bound in* M, non-bound variables are *free*.
- $(\lambda x.M)N$ is a "call": *replace* x in M by N.

$$(\lambda x.x)y \;\to\; y$$

$$(\lambda x.xx)(\lambda x.x)y \to (\lambda x.x)(\lambda x.x)y \to (\lambda x.x)y \to y$$

But how to handle:

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

Replacement **must not** bind free variables.

The replacement operation

$$[x \mapsto N]M$$

(i) $[x \mapsto N]x$ is N,

(ii) $[x \mapsto N]y$ is y for any variable y different from x,

(iii) $[x \mapsto N](PQ)$ is $([x \mapsto N]P \; [x \mapsto N]Q)$,

(iv) $[x \mapsto N]\lambda x.P$ is $\lambda x.P$,

(v) $[x \mapsto N]\lambda y.P$ is $\lambda y.[x \mapsto N]P$ if y is different from x and has no free occurrence in N, or if x has no free occurrence in P,

(vi) $[x \mapsto N]\lambda y.P$ is $\lambda z[x \mapsto N]([y \mapsto z]P)$ if y is different from x and has a free occurrence in N, and x has a free occurrence in P. z is a variable that does not occur free in N nor in P.

$$(\lambda xy.xy)y \to [x \mapsto y](\lambda y.xy) =$$
$$\lambda z[x \mapsto y]([y \mapsto z]\lambda y.xy) =$$
$$\lambda z[x \mapsto y](\lambda z.xz) = (\lambda z.yz)$$

The α-conversion

A rewrite relation $\to_\alpha$ on λ-terms:

(α) If y is not free in M, then
$\lambda x.M \quad \to_\alpha \quad \lambda y.[x \mapsto y]M$.

- Intuitively $\to_\alpha$ is a renaming;
$$\lambda x.xy \to_\alpha \lambda z.zy$$
- Restriction on $\to_\alpha$: "fresh" variables are to be used:
$$\lambda x.xy \not\to_\alpha \lambda y.yy$$
- $\twoheadrightarrow_\alpha$ is symmetric, e.g.:
$$\lambda xy.xy \twoheadrightarrow_\alpha \lambda zw.zw$$
$$\lambda zw.zw \twoheadrightarrow_\alpha \lambda xy.xy$$
If $s \twoheadrightarrow_\alpha t$ then s and t are called *α-congruent*.

The β-reduction

$$(\lambda x.M)N \quad \to_\beta \quad [x \mapsto N]M$$

- Intuition of $\to_\beta$: function call.
- Computation by β-reduction may or may not terminate, e.g.
$$(\lambda y.yy)(\lambda y.yy) \to (\lambda y.yy)(\lambda y.yy)$$
- Church-Rosser theorem: if $P \twoheadrightarrow_\beta M$ and $P \twoheadrightarrow_\beta N$ then there exist α-congruent terms T_1 and T_2 such that $M \twoheadrightarrow_\beta T_1$ and $N \twoheadrightarrow_\beta T_2$.
- Hence, every lambda term has a unique (up to α-conversion) normal form, or no normal form at all.

Some rewrite strategies

- The *normal-order reduction* (lazy evaluation):
reduce the left-most redex,e.g.:
$$\begin{aligned}&(\lambda x.xx)((\lambda y.y)(\lambda y.y)) \to\\&((\lambda y.y)(\lambda y.y))((\lambda y.y)(\lambda y.y)) \to\\&(\lambda y.y)((\lambda y.y)(\lambda y.y)) \to\\&((\lambda y.y)(\lambda y.y)) \to\\&\lambda y.y\end{aligned}$$
- The *applicative-order reduction* (call by value):
reduce the left-most innermost redex, e.g.:
$$\begin{aligned}&(\lambda x.xx)((\lambda y.y)(\lambda y.y)) \to\\&(\lambda x.xx)(\lambda y.y) \to\\&((\lambda y.y)(\lambda y.y)) \to\\&\lambda y.y\end{aligned}$$

The rewrite strategies and termination

- If the normal form exists, the normal-order reduction will compute it.
- The applicative order reduction may not terminate even if there is a normal form, e.g.:
$$(\lambda xz.z)((\lambda y.yy)(\lambda y.yy)) \to$$
$$(\lambda xz.z)((\lambda y.yy)(\lambda y.yy))$$
using the applicative-order reduction. But:
$$(\lambda xz.z)((\lambda y.yy)(\lambda y.yy)) \to (\lambda z.z)$$
using the normal-order reduction.

Representing numbers as lambda terms

Church numerals:
Represent a natural number n by

$$\lambda xy.x^n y$$

where $x^n y$ denotes n applications of x to y.
Thus:

- $\lambda xy.y$ abbreviated as $\mathbf{0}$, denotes 0.
- $\lambda xy.xy$ abbreviated as $\mathbf{1}$, denotes 1.
- $\lambda xy.x(xy)$ abbreviated as $\mathbf{2}$, denotes 2.
- etc. ...

Notice:

- Each Church numeral is normal form.
- It can be seen as a function on lambda terms: given lambda terms M and N:

$$\mathbf{n}MN \twoheadrightarrow M^n N$$

Defining Function on Numbers

- Lambda term $\mathbf{s}$ defining successor:

$$\lambda uxy.x(uxy)$$

$$\begin{aligned}&\mathbf{sn}= \lambda uxy.x(uxy)(\lambda zv.z^n v) \to\\ &\lambda xy.x((\lambda zv.z^n v)xy) \to\\ &\lambda xy.x((\lambda v.x^n v)y) \to\\ &\lambda xy.x(x^n y) = \mathbf{n+1}\end{aligned}$$

- Addition:
  $\mathbf{n}xy \twoheadrightarrow x^n y$,
  $x + y = s^x s^y(0)$, hence

$$\lambda xy.x\mathbf{s}(y\mathbf{s}\ \mathbf{0})$$

defines addition. Its extended form is:

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

The power of lambda calculus

ϕ : an m-ary function on Nat

ϕ is *lambda-defined* by a lambda term F: iff for arbitrary natural numbers $n_1, ..., n_m$

$$F\ \mathbf{n}_1\ ...\ \mathbf{n}_m \twoheadrightarrow \mathbf{n} \text{ iff } \phi(n_1, ..., n_m) = n.$$

ϕ is *lambda-definable* if
it is lambda-defined by some lambda term.

The class of lambda-definable functions coincides with the class of

- *partial recursive* functions,
- Turing-computable functions

Partial Recursive Functions

Primitive recursive functions:

- the constant funtion 0
- the successor funtion
- the n, k projection functions

$$p(m_1, ..., m_n) = m_k$$

- if $\psi, \chi_1, ..., \chi_p$ are prim.r. and
  $\phi(m_1, ..., m_n) = \psi(\chi_1(m_1, ..., m_n), .., \chi_p(m_1, .., m_n))$
  then ϕ is prim.r.
- if ψ, χ are prim.r., and
  $\phi(0, m_1, ..., m_n) = \psi(m_1, ..., m_n)$
  $\phi(k+1, m_1, .., m_n) = \chi(k, \phi(k, m_1, .., m_n), m_1, .., m_n)$
  then ϕ is prim.r.

ϕ is **partial recursive** if

$\phi(m_1, ..., m_n) = \psi(\mu k[\chi(m_1, ..., m_n, k) = 0])$

where ψ, χ are some prim.r. functions and
$\mu k[\chi(m_1, ..., m_n, k) = 0]$
is the least k satisfying the equation.

Combinatory logic

- CL is yet another rewrite system
- Equivalent to lambda calculus
- Does not use bound variables
- Used in implementation of functional languages
- Intuition: to represent composition of functions
- It is an example of a *term rewriting system*

Combinatory terms

(i) A variable is a combinatory term,

(ii) The basic combinators K and S are combinatory terms,

(iii) If X and Y are combinatory terms, then the *application* (XY) is a combinatory term with subterms X and Y.

e.g.: $((SK)K)$, $(((SK)K)x)$
Notation: the application assumed left-associative, e.g.
$SS(K(SKK))$ denotes $((SS)(K((SK)K)))$

Intuition:

- C-terms represent (partial) functions on C-terms computed by rewriting.
- The value of F for A, is the normal form of the term (FA).
- K allows for creation of constant functions.
- S describes a specific way of composition of functions.

Combinatory reduction

- $KXY \to X$
- $SXYZ \to XZ(YZ)$
- This extends for reduction of subterms, i.e.:
  If $X \to W$ then $(XY) \to (WY)$ and $(ZX) \to (ZW)$

$S(KS)Kxyz \to$
$(KS)x(Kx)yz \to$
$S(Kx)yz \to$
$(Kx)z(yz) \to$
$x(yz)$

- Combinatory logic is confluent.
- If a C-term X has a normal form then reduction of X under normal-order strategy terminates.