Lambda Calculus.

purely mathematical language. — Alonzo Church, 1930's.

what does it mean to apply a function to an argument.

program input.

$f(t) \quad -(f\ t)$

$(\lambda x.x+x)\ 3 \quad \rightarrow$ "substitution" $\quad 3+3 \rightarrow (6)$ ← result of computation

↓ normal form — can't be reduced further.

copy & paste.

$\lambda x.(+)\ x\ x$ — constants can be defined as λ-terms. $+, 3$ will also be λ-terms.

— but not just copy & paste. must respect scope of variables.

```
int f(int x) {
  { for (int x = 0. ...) {
      (x)
    }
    return (x);
}
f(2)
```

```
int y = 1;   ← free
int f(int x) {
  int y = 2;   ← bound.
  return x + y;
}
```

3 → free var cannot be captured

$f(y) \rightarrow 3$, not 4. by bound var.

Formal Definitions

Inductive Definition of λ-terms:

- a λ-term is inductively defined as follows.
  - a variable x, y, z ⇒ a λ-term (lower case)
- if A and B are λ-terms, then so is $(A\ B)$ A applied to B (upper case)
- if A is a λ-term and x is a variable, then $\lambda x.A$ is a λ-term

  Abstraction.

— constants (not strictly necessary) $+, 3, 4$ — allow for now

$+$ is a constant - representing a function.

$\lambda x.(yx)$ is λ term, y x so $(y\ x)$ so $\lambda x.(y\ x)$.

(33) — don't worry about meaning — don't associate symbol with meaning — meaning will be defined later — think purely mechanically.

Syntactic Conventions:

1. applications associates to the left. $A\,B\,C = (A\,B)\,C \neq A\,(B\,C)$.

2. application binds tighter than abstraction:

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

— implication:

— scope of a λ extends all the way to the end of the term unless terminated by ( )'s.

$(\lambda x.\, x\,y)\,y$ ↳ must have ( ) around λ to distinguish from argument.

— $\lambda x.\,(x\,y)\,z$ ← superfluous. $\lambda x.\, x\,(y\,z)$ ← not superfluous.

$\lambda x.\,(x\,y\,z)$ ← superfluous. $(\lambda x.\, x\,y)\,z$ ← not superfluous.

$3 + (4 * 5)$

$(5-3)-2$?= 4?

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

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

Free and bound variables

$\lambda x.\, y\,x$ free

↳ bound

always relative to a term.

$\lambda y.\, x\,(\lambda x.\, x)\,y$ ← free, bound.

within body, y is free.

$\lambda x.\, x\,x$ x is bound but

inside body $x\,x$, x is free.

$\lambda x.\,[(\lambda x.\, x)\,x\,x]$ — free. inside body.

Substitution & β-reduction

$(\lambda x.\, M)\,N =_\beta M[N/x]$

when $M[N/x]$ replace all free occurrences of x in M with N

provided: the free vars of N are not bound in M.

$(\lambda x.\, M)\,N$ is called a Beta-redex — a place where β-reduction can be applied.

Examples: $(\lambda x.\,[x\,(\lambda x.\, x)])\,y \to (y\,(\lambda x.\, x))$

this is free in M ; normal form

$(\lambda x.\, xy)\ (\lambda z.\, z) \to (\lambda z.\, z)\, y \to (y)$ normal form.

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

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

$\to v\ v.$

$(\lambda x.\, y)\ z \to y$ vacuous binder

x not free in y $\quad (\lambda x.\, A)\ B$, x not free in A

$\to A.$

$(\lambda y \lambda x.\, (\lambda y.\, y + x)\ 2)\ 1 \to$

$\lambda x.\, 2 + x.$

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

$(\lambda y.\, y + y)\ 2$ — wrong.

← one of the C programs.

— later.

---

do all terms reduce to β-normal form? No.

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

Church Rosser:

$(\lambda x.\, xx)\ ((\lambda u.\, u)\ v) \to (\lambda x.\, x)\ v$

$((\lambda u.\, u)\ v)\ ((\lambda u.\, u)\ v)$

$v\ v.$

later.

Bad examples:

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

$\lambda x.\, x\, v \to$ application binds tighter than abs.

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

→ application associates to the left.

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

$\to x\ \lambda y.\, y y.$

4. $(\lambda y \lambda x.\, x\, y)\ (x x) \to \lambda x.\, x (x x)$

free x in bound in M.

α-conversion

$\lambda x.\, M =_\alpha \lambda y.\, M[y/x]$

y is fresh (does not appear in M).

← α-convert only bound vars, never free vars.

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

$(\lambda x.\,(\lambda y.\, y+x)\,2)\,y$ ← C example.

$\to (\lambda y.\, y+y)\,2 \to 2+2$ - ✗

$(\lambda x.\,(\lambda y'.\, y'+x)\,2)\,y$

$(\lambda y'.\, y'+y)\,2 \to 2+y$

α-conv. $\lambda x\,\lambda y.\, x+y$ $\lambda y\,\lambda x.\, y+x$. are α-conv.

$\lambda x\,\lambda y.\, x+y$ $\lambda x\lambda y.\, y+x$ are not α-conv.

— don't confuse α with β. α is static pure syntactic equality
β is evaluation - Running the program.

$(\lambda x.\, y)\,x$ and $(\lambda x.\, y)\,y$ are not α-convertible.

Are there always normal forms? no

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

— Church-Rosser-

$(\lambda x.\, xx)\,((\lambda u.\, u)\,v)$

$((\lambda u.\, u)\,v)\,((\lambda u.\, u)\,v)$ → $(\lambda x\,x)\,v$

A

B C

D

CBV v.s. CBN.

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

eval strategies CBV — args first - inner most.

CBN — leftmost outermost -

weak CBV → most languages. $(\lambda x.\,(\lambda y.\, y)\,x) \to$ reduce to itself - don't eval inside λ.

CBV can be more efficient, in two ways -

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

Theorem. If a term has a normal form, then it can be reached via CBN.

— DAG v.s. tree.

Basic Combinators. (pure λ-terms - no theorems)

$I = \lambda x.x$

$K = \lambda x \lambda y.x$

$S = \lambda x \lambda y \lambda z.\ xz(yz)$

$I\ A = (\lambda x.x)A \to A$ Axiom $IA = A$ for any term A.

$KAB = (\lambda x \lambda y.x)\ A\ B$ Case 1: y is not free in A.

$\to (\lambda y.A)\ B \to A$.

Case 2: y is free in A

$(\lambda x.\lambda y.x)\ A\ B \to (\lambda y.A)\ B \to A$

$\therefore$ Axiom $KAB = A$.

$KI = (\lambda x \lambda y.x)(\lambda u.u) \to \lambda y \lambda u.u \overset{\alpha}{=} \lambda x \lambda y.y$.

$KIAB = B$.

because $(KIA) = I$ and $IB = B$. Axiom $KIAB = B$.

$(\lambda x \lambda y.y)\ A\ B \to (\lambda y.y)\ B = B$.

SKI

$(\lambda x \lambda y \lambda z.xz(yz))\ KI$.

$(\lambda y \lambda z.Kz(yz))\ I$

$\lambda z.Kz(Iz)$

CBN: $\lambda z.z$ ← $\lambda z.Kz(Iz)$ → CBV: $\lambda z.Kzz \to \lambda z.z$

$KAB = A$

CBN or CBV.

Class exercise KII.

K, and KI $\lambda x \lambda y.x$ choose to $\lambda x \lambda y.x$ choose second

$True = K \quad \lambda x \lambda y.x$

$False = KI \quad \lambda x \lambda y.y$.

$IFElse = \lambda c \lambda a \lambda b.\ c\ a\ b$.

$IF\ Else\ True\ A\ B \to$ $True\ A\ B \to A$

$IF\ Else\ False\ A\ B \to$ $KI\ A\ B \to B$.

— philosophically, this is the behaviour that we want from true/false, from a programmatic / algorithmic perspective.

now we assign meaning to terms.

AND = $\lambda a \lambda b.$ ifelse a b false -

if a returns b else return false

— short circuited booleans.

OR $\lambda a \lambda b.$ ifelse a T b -

NOT $\lambda a.$ ifelse a F T.

Church Numerals.

$\emptyset$ is $\lambda f \lambda x. x$

1 is $\lambda f. \lambda x. f x$

2 is $\lambda f \lambda x. f(f x)$

3 is $\lambda f \lambda x\ f(f(f x))$, etc... — admissible w/r to λ-calculus.

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

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

$=_\alpha\ \lambda m \lambda n \lambda f.\ m\ (n f)$.

---

expt $\lambda m \lambda n . (n\ m)$.

plus $\emptyset$ 1 :

$(\lambda m \lambda n \lambda f \lambda x.\ m\ f\ (n f x))\quad \emptyset\ 1$

$\lambda f \lambda x.\ \emptyset\ f\ (1\ f\ x)$

$\lambda f \lambda x.\ \emptyset\ f\ (f x) = (\lambda x \lambda y. y)\ f\ (f x)$

$= \lambda f \lambda x.\ \underline{f x}. = 1$

Times $\emptyset$ 2.

$(\lambda m \lambda n. \lambda f.\ m (n f))\ \emptyset\ 2$

$\lambda f.\ \emptyset\ (2 f)$

$= \lambda f \lambda x.\ \emptyset\ (2 f)\ x.$

$\lambda x \lambda y. y = \lambda f \lambda x. x.$

---

set theory.

$\{\}$ 0

$\{\{\}\}$ 1

$\{\{\}, \{\{\}\}\}$ 2 .

decreasing numbers.

η-conversion.

$\lambda x.\ m x =_\eta m$ - up to syntax.

what does "3" mean?

expt $m^n = \lambda m \lambda n.\ n\ m$. one flaw.

$m^0 \neq 1$. $(\lambda f \lambda x.\ x)\ m \to \lambda x.\ x$.

$\lambda m \lambda n.$ if (iszero n) $(\lambda f \lambda x.\ f\ x)$ (else) $(n\ m)$ 3:

$\to^{k}_{k \geq 1}$ ← homework.

$2^3 = 8$: $(3\ 2)$

$(\lambda f \lambda x.\ f\ (f\ (f\ x))\ 2 \to \lambda x.\ 2\ (2\ (2\ x))$

$2\ (2\ (\lambda z.\ x\ (x\ z)))$

$2\ (\lambda z.\ x\ (x\ z))$

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

$(x\ (x\ u))$

$\lambda u.\ (x\ (x\ (x\ (x\ u))))$

4

4'

2 4'

$(\lambda f\ \lambda x.\ f\ (f\ x))\ 4'$

α-conv. $(\lambda f \lambda x.\ f\ (f\ x))\ x$

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

$\lambda u.\ 4'\ (4, u)$

$(x\ (x\ (x\ (x\ u))))$

$\to \lambda u.\ x\ (x\ (x\ (x\ (x\ (x\ (x\ (x\ u)))))))$

$=_2\ 8.$

PRED (-1) $= \lambda n \lambda f \lambda x.\ n\ (\lambda g.\lambda h.\ h\ (g\ f))\ (\lambda u.x)\ (\lambda u.u)$

Subtract $= \lambda m \lambda n.\ n$ PRED m

3-2 $(\lambda f \lambda x.\ f\ (f\ (f\ x)))$ PRED m -

PRED (PRED (PRED m))

PRED two.

$\lambda f \lambda x.$ Two $\overbrace{(\lambda g \lambda h.\ h\ (g\ f))\ (\lambda u.x)\ (\lambda u.u)}^{G}$

$(G\ (G\ (\lambda u.x\ 1)\ \cdot\ (\lambda u.u)$

$G\ (\lambda h.\ h\ x)\ (\lambda u.u)$

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

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

DATA Structures:

- pairs (a, b) $a, (b, c)$ linked list $a, (b, (c, Nil))$

if c a else b.

(a, (left, right) tree

pair a b

cons $= \lambda a \lambda b.\ \lambda c.\ c\ a\ b$

1st car $= \lambda p.\ p\ T$

2nd cdr $= \lambda p.\ p\ F.$

cons 2 3 $= (\lambda c.\ c\ 2\ 3)$

car $(\lambda c.\ c\ 2\ 3)$

$(\lambda c.\ c\ 2\ 3)\ T \to T\ 2\ 3 \to 2$

CDR $(\lambda c.\ c\ 2\ 3) \to (\lambda c.\ c\ 2\ 3)\ F \to F\ 2\ 3 \to 3.$

con 2 (con 3 (cons 5 Nil))

Nil $= \lambda x \lambda y.\ y$ = zero = False.

ISNil

want: ISNil $\lambda x \lambda y.\ y \to T$

ISNil $(\lambda c.\ c\ 2\ 3) \to F.$

ISNil $= \lambda p.\ p\ (\lambda x \lambda y \lambda z.\ \text{False})\ T$

$\lambda c.\ c\ \underline{2}\ \underline{3}\ (\quad)\ \underline{T}$

Isnil (λc. c 2 3) →
(λc. c 2 3) (λx λy λz. False) T
(λx λy λz. False) 2 3 T → False.

Isnil Nil = F ____ T → T.

idea is consistent with Abstract data type -
Data & Interface (λc. c 2 3) provides interface -

(cons (2 (con 3 Nil)).
CAR (CDR λc. c 2 (λd. d 3 Nil)
CAR λd. d 3 Nil (λd. d 3 Nil) T
→ (3). T 3 Nil → 3 -

Repetition / Recursion. ↳ illegal

need term with following behavior :

Fix m = m (Fix m)
= m (m (Fix m))
→ (m (m (m (Fix m))))
. . . .

Fix = λm (λx. m (x x)) (λx. m (x x))

fixpt of a function f(x) = x as a fixpoint. but x is itself a function.

Confirm Fix m = m (Fix m).

Fix m = (λx. m (x x)) (λx. m (x x))
→ m ((λx. m (x x)) (λx. m (x x)))
= m Fix m.

Sum of a linked list: Sum (n) → if (isnil n) zero
else car (n) + Sum (cdr n).

Sum = Fix λf λn. ifelse (isnil n) zero (plus (car n) (f (cdr n)))
m

f is name of function, not defining something in terms of itself.

Sum (cons 2 (cons 3 Nil))
n

Fix m (2, 3, Nil)
= m (Fix m) (2, 3, Nil)
= (λn. ifelse ...) (2, 3, Nil)

; felse (isnil (u, 3, nil)) zero (plus 2 . Fixm (3, nil)).

Fixm (3, Nil) -
= m (Fixm) (3, nil)
= λt λn . . .
ifelse (isnil (3 nil)) zero (pluss (Fixm nil)).

Note: can't use call-by value.

(Fixm) Nil.
= m (Fixm) Nil
λcλn → ifelse (isnil nil) zero (plus ...)
↓ zero. car nil → (λx.y)
λp p T) = (λt λx . y) T

Fix = Y — Y combinator.

Divide n m = Fix λd. λm λn. if (LT m n) 0
plus 1 (d (m−n) n)

LT (less than)
is λm λn. iszero (sub m n) and not(iszero (sub n m)).

Applicative order considerations

CBV.

If else c a b → want to ignore reduction of a, b,

If_else = λc λa λb. c a b I.

if-else False λx.1/0 , λx. 2.

False λx. 1/0 λx. 2 I
λx. 2 I → 2.

do not eval until applied to an argument.

Applicative order Fix:

$Z = \lambda m. ((\lambda x. m (\lambda y. x x y)) (\lambda x. m (\lambda y. x x y)))$

$Z m = (m (Z m))$
$m (\lambda y (\lambda x. m (\lambda y. x x y)) (\lambda x. m (\lambda y. x x y)) y)$
$=_\eta$ m (Fix m) — requires eta-equivalence.

Static Scoping.

```
int y = 1
int f(int x){return x+y}
int g(){
  int y = 2;
  return f(0)
}
```

first cover let.

$let\ x = v\ in\ B \equiv (\lambda x.B)\ v.$

A;B
A then B $(\lambda x.B)\ A.$ first do A then do B. in CBV.

$$let\ y = 1\ in\ let\ f = \lambda x.\ x+y\ in\ let\ y = 2\ in\ (f\ 0)$$

$$(\lambda y.\ (\lambda f.\ (\lambda y.\ f\ 0)\ 2)\ (\lambda x.\ x+y))\ 1$$

- So the scoping rules of λ-calculus gives static scoping.

Undecidability of the Halting problem:

There's a λ-term HALT:

given P, Q, if $(P\ Q) \longrightarrow_\beta$ normal form

then (HALT P Q) → True

if (P Q) has no normal form

(HALT P Q) → False -

$$let\ Q = \lambda p.\ if\ (HALT\ p\ p)\ ((\lambda x.xx)(\lambda x.xx))\ else\ (\lambda x.x).$$

? Does (Q Q) halt (have normal form)?

if yes - then 0

if no then yes.

contradicts assumption that HALT exists.

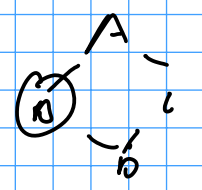
the existence of HALT leads to a contradiction.

Admissibility w/r to λ-calculus is important.

Church-Rosser

Statelessness. (f 0) (f 0) - print

affects optimization.



f(n) + f(n)

let x = f(n) → print once versus print twice.
x + x

- Seperate computation from ib.

x = x+1. not let x = (.

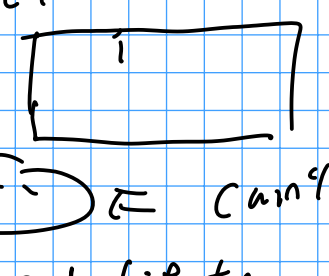
↓

but can this still be admissible. $\lambda_{\underline{b}}$. $\underline{\lambda_b}$

let x = 1 in

Should be ← let x = x+1

new memory location.



x has new value.

but can we reuse old mem. location?

mutation ends lifetime.

can't use x again.

perfectly fine.

```
for (int i=0; i<n; i++)
;
```

typed λ-calculus.

Simply typed λ-calculus.

types written a → b. a, b can be replaced by any type.

λx. M : a → b → c ≡ a → (b → c)

→ associates to the right.

$\underline{I} = \lambda x.x : a \to a$.

$k = \lambda x \lambda y.x : a \to b \to a$.

type derivation ⊢ term : type.

$$x : a \ldots \vdash x : a.$$

$$\frac{x : a \ldots \vdash M : b}{\ldots \vdash \lambda x.M : a \to b}\ abs \qquad \frac{\ldots \vdash M : a \to b \quad \ldots \vdash N : a}{\ldots \vdash (M\,N) : b}$$

S I k

$(\lambda x \lambda y \lambda z.xz(yz))\ I\ k$

$\lambda z.\ I\,z(k\,z)$ -

$\lambda z.\ z\,(\lambda y.y)$ -

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

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

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

$$\frac{y:a\to b \vdash y:a\to b \qquad x:a \vdash x:a}{\text{app.}}$$

$$\frac{x:a,\ y:a\to b \vdash yx : b}{\text{abs}}$$

$$\frac{x:a \vdash \lambda y.\, yx : (a\to b)\to b}{\text{abs}}$$

$$\lambda x \lambda y.\, yx : a \to (a\to b)\to b$$

$$\frac{x:a \vdash x:a}{\lambda x.x : a\to a.}$$

$$\frac{x:a,\ y:b \vdash y:z}{x:a \vdash \lambda y.\, y : b\to b}\ \text{abs}$$

$$\lambda x \lambda y.\, y : a\to b\to b$$

c = modus ponens.

type inference.

$\lambda x \lambda y.\, yx :$ $A \to B$.

$A \to B \to z$

but $B = (A \to C)$

$A \to (A\to C) \to C$.

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

$A \to B \to C \to D$.

$A = (C \to E \to D.)$

$B \to (C \to E)$

$(C \to E \to D) \to (C \to E) \to C \to D$.

propositional tautology:

← Curry–Howard isomorphism.

type Soundness theorem

if $\Gamma \vdash S : A$ is a valid type derivation

and $S =_\beta t$, then $\Gamma \vdash t : A$ is also a valid type derivation.

type safety. — (Subject Reduction).

$$\frac{x:a\to b\to c \vdash x:a\to(b\to d) \qquad z:a \vdash z:a}{x:a\to b\to c,\ z:a \vdash xz : b\to c} \qquad \frac{y:a\to b \vdash y:a\to b \qquad z:a \vdash z:a}{y:a\to b,\ z:a \vdash yz : b}$$

$$\frac{x:a\to b\to c,\ y:a\to b,\ z:a \vdash (xz)(yz) : c}{\lambda x \lambda y \lambda z : x\ z\ (y\ z) \ :\ (a\to b\to c)\to(a\to b)\to a\to c}$$

problem with types

$K : \lambda x \lambda y.\, x : a\to b\to a$

KI $\lambda x \lambda y.\, y : a\to b\to b$

can't call both "bool"

unless both becomes $a\to a\to a$

Can't have nested pairs.

(cons 1, (cons 2, 5))

hd ↗ ↗ tl

⟵ allow another kind of term: (a, b)

$\frac{\vdash a : t \quad \vdash b : s}{\vdash (a, b) : (t * s)}$

* means ∧ and.

a*b → a

a*b → b.

```
let f(n):
  if (n>0): return 1
  else: return "abc"
```

✗ not typable.

— pairs must become primitive.

Strong Normalization: every term has a β-normal form (unique).

— Fix must be imported

Fix m = m(Fix m) means Fix has type $(a \to a) \to a$.

— not a tautology.

Disjoint type $a \mid b$

$\frac{\vdash x : a}{\vdash \text{il}\, x : a \mid b} \qquad \frac{\vdash x : b}{\vdash \text{inr} : a \mid b}$.

type Bntree = Nil | Node of nt * Bntree * Bntree -

Nil: Bntree

Node(1, Nil, Nil): Bntree.

Quantification

Subtyping → compromises static typing.