Intuition on λ abstraction & application

Let's pretend we have $\mathbb{N}, +$

1) $((\lambda x.\, x+5)\ 10) = 10+5 = 15$

2) $(\lambda f.\, f\ 7)\ (\lambda x.\, x+5) =$
$= (\lambda x.\, x+5)\ 7 = 7+5 = 12$

3) $(\lambda f.\, f\ (f\ 7))\ (\lambda x.\, x+5) =$
$(\lambda x.\, x+5)\ ((\lambda x.\, x+5)\ 7) =$
$(\lambda x.\, x+5)\ 12 = 17$

4) $((\lambda x.\, (\lambda y.\, x))\ 5)\ 7 =$
$(\lambda y.\, 5)\ 7 = 5$

Intuition 1.

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

variable renaming should be allowed

"rule α" / α-conversion

Intuition 2

$((\lambda x.\, t)\ e) = t\{e/x\}$

substitution of e for x in t

"rule β"

Free vs bound variables

$$\sum_{i} (i+j) \left[=_\alpha \sum_{k} (k+j) \right]$$

bound i, free j

$$\int_a^b f(x,y)\,dx \qquad \forall x.\ p(x,y)$$

x: bound, y: free; x: bound, y: free

$$\begin{aligned}
&free(x) = \{x\} \\
&free(te) = free(t) \cup free(e) \\
&free(\lambda x.t) = free(t) \setminus \{x\}
\end{aligned}$$

Substitution $t\{e/x\}$

$$\begin{aligned}
&x\{e/x\} = e \\
&y\{e/x\} = y \quad \text{if } y \neq x \\
&(t_1\ t_2)\{e/x\} = (t_1\{e/x\}\ \ t_2\{e/x\})
\end{aligned}$$

$$(\lambda y.t)\{e/x\} = \lambda z.(t\{z/y\}\{e/x\})$$

where $z = x_i$ with

$$i = \min\{i \mid x_i \notin free(\lambda y.t) \cup free(e) \cup \{x\}\}$$

$$\lambda x.y \ =_\alpha \ \lambda z.y$$

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

$\lambda x.z$ $\qquad$ $(\lambda z.z)$ $\quad \lambda z.z$

We must rename z

α - conversion

$$\frac{}{(\lambda x.t) =_\alpha (\lambda y. t\{y/x\})} \text{ if } y \notin free(t)$$

$$\frac{t =_\alpha t'}{\lambda x.t =_\alpha \lambda x.t'} \qquad \frac{t =_\alpha t'}{(te) =_\alpha (t'e)} \qquad \frac{e =_\alpha e'}{(te) =_\alpha (te')}$$

$=_\alpha$ closed under contexts

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

α must be an equivalence

β - rule $\quad \to_\beta$ β - reduction

$$\frac{}{(\lambda x.t)e \longrightarrow_\beta t\{e/x\}}$$

β closed under contexts

$=_\beta$ β - conversion

the smallest equivalence containing $\to_\beta$, i.e.,

$$(=_\beta) = \left((\to_\beta) \cup (\underset{\beta}{\longleftarrow}) \right)^*$$

Optional rule : η rule

$$(\lambda x. t\, x) \longrightarrow_\eta t \quad \text{if } x \notin free(t)$$

Th: λ calculus is "Turing complete",
i.e. it can perform all the computations $\mathbb{N}^k \to \mathbb{N}$ that we can perform in a general programming language (e.g. Java)

We present this result by showing how to encode the most common programming tools in λ,
e.g.: data types (booleans, $\mathbb{N}$, ...)

Church encoding of data types
WARNING: do not memorize this!
We will see a more general result later on.

Booleans

- Introduction / Construction
  T, F true, false constants
- Elimination / Destruction
  if b then t else e
- Laws
  if T then t else e = t
  if F then t else e = e
- A solution
  $T = \lambda x.(\lambda y.\, x)$
  $F = \lambda x.(\lambda y.\, y)$
  if b then t else e = (bt)e

Notation :

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

$\rightsquigarrow \lambda x\, y\, z.\, t$

$(((t\ e)\ f)\ g)$

$\rightsquigarrow (t\ e\ f\ g)$

$T = \lambda x y.\ x \qquad F = \lambda x y.\ y$

if b then t else e $= b\ t\ e$

if T then t else e $= T\ t\ e = t$

if F then t else e $= F\ t\ e = e$

$Not = \lambda b.$ if b then F else T

$Not' = \lambda b\, x\, y.\ b\ y\ x$

$And = \lambda a b.$ if a then b else F

$Or = \lambda a b.$ if a then T else b

Pairs

- Intro $(C\ x\ y)$
- Elim π_1, π_2 projections
- Laws :

  $\pi_1\ (C\ x\ y) = x$

  $\pi_2\ (C\ x\ y) = y$
- A solution

  $C = \lambda x y.(\lambda k.\ k\ x\ y)$

  $\pi_1 = \lambda p.\ p\ (\lambda a b.\ a)$

  $\pi_2 = \lambda p.\ p\ (\lambda a b.\ b)$

$C = \lambda xy.(\lambda k.\, k\, x\, y)$

$\pi_1 = \lambda p.\, p\,(\lambda ab.\, a)$

$\pi_2 = \lambda p.\, p\,(\lambda ab.\, b)$

Check:

$\pi_1\,(C\, x\, y) =$

$\pi_1\,(\lambda k.\, k\, x\, y) =$

$(\lambda k.\, k\, x\, y)(\lambda ab.\, a) =$

$(\lambda ab.\, a)\, x\, y = x$ ✓

$\pi_2\,(C\, x\, y) =$

$\pi_2\,(\lambda k.\, k\, x\, y) =$

$(\lambda k.\, k\, x\, y)(\lambda ab.\, b) =$

$(\lambda ab.\, b)\, x\, y = y$ ✓

Naturals

- Intro 0, $S\,n$ (zero, successor)
- Elim iter $n\ f\ x$

  $(\cong \underbrace{f(f \cdots (f\, x))}_{n \text{ times}})$
- Laws

  iter $0\ f\ x = x$

  iter $(S\,n)\ f\ x = f(\text{iter}\ n\ f\ x)$
- A solution

  $0 = \lambda f x.\, x$

  $S = \lambda n.\, \lambda f x.\, f(n\, f\, x)$

  iter $n\ f\ x = n\, f\, x$
- Check

  iter $0\ f\ x = 0\, f\, x =$

  $(\lambda f x.\, x)\, f\, x = x$ ✓

  iter $(S\,n)\ f\ x = (S\,n)\, f\, x =$

  $(\lambda f x.\, f(n\, f\, x))\, f\, x =$

  $f(n\, f\, x) = f(\text{iter}\ n\ f\ x)$ ✓

$\underbrace{S(S(S \cdots}_{n} O) = \lambda f x. \underbrace{f(f \cdots f}_{n} x)$

$Even\ n \begin{cases} T & \text{if } n \text{ is even} \\ F & \text{o.w.} \end{cases}$

$Even = \lambda n.\ n\ Not\ T$

$IsZero\ n \begin{cases} T & \text{if } n = 0 \\ F & \text{o.w.} \end{cases}$

$IsZero = \lambda n.\ n\ (\lambda a.\ F)\ T$

$Add = \lambda n\ m.\ n\ S\ m$

$Mul = \lambda n\ m.\ n\ (Add\ m)\ O$

Tricky: predecessor:

$P\ O = O$

$P\ (S\ n) = n$

$g(a, b) = (b, b+1)$

$(0,0) \mapsto (0,1) \mapsto (1,2) \mapsto (2,3) \cdots$

$g^n(0,0) = (P\ n, n)$

$P = \lambda n.\ \pi_1\ (n\ G\ (C\ 0\ 0))$

$G = \lambda p.\ ((\pi_2\ p)\ (S\ (\pi_2\ p)))$

Ex: write the factorial

$n\ f\ x \approx$ for loop in programming lang.

for i := 1 to n do

...

We still need a "while" loop

```
x := 0;
while n > 1 do
    x := x + 1;
    if even n then
        n := n / 2;
    else n := 3 · n + 1;
return x;
```

↑ Collatz function

We currently do not know whether the program halts $\forall n \in \mathbb{N}$

We need general recursion to implement this in λ calculus