lambda calculus

Jianguo Lu

History

- A formal system designed to investigate function definition, function application, and recursion.
- Introduced by Alonzo Church in 1930s.
- The calculus can be used to cleanly define what a computable function is.
- Lambda calculus influenced Lisp.
- It is the smallest universal programming language
  - Smallest: simple syntax and transformation rule
  - Universal: any computable function can be expressed and evaluated using this formalism.
    - Equivalent to Turing machine (Church-Turing Thesis)

syntax

- Syntax
  - variable

    <expr> ::= <variable>
  - lambda abstraction

    <expr> ::= λ <variable>. <expr>
  - lambda application

    <expr> ::= <expr> <expr>
- Example

  lambda abstraction: λx. x*x
  f(x)=x*x

  lambda application: (λx. x*x) 3
  f(3)

- Whether the following are λ expressions?

  x

  λx.x

  xy

  x.x

  (λx.x)xy

  λx.xx

Correspondence to java program

```
int  f ( int  x) { return x - 2;}
```

```
    λ        x  .          - x  2
```

```
int  f ( int  x, int y ) { return x - y;}
```

```
    λ    x  . λ    y.   - x  y
```

```
int  f ( int  x, int y ) { return x - y;}  f(2,3)
```

```
    (λ    x  . λ    y.   - x  y) 2 3
```

- Two notational conventions
  - The scope of λ extends as far as possible to the right
    - λ x. - x 2 is λ x . (- x 2)
    - λx. xx is λx.(xx)
  - Association to the left
    - (λx. λy. – x y) 2 3 is ((λx. λy. – x y) 2) 3
- How to reduce (calculate) f(2, 3),

  or (λx. λy. – x y) 2 3 ?
- Alpha, beta, eta conversions

β reduction

- β Rule

  (λx. E) E′ ➔ E[E′/x]

  (λx.x)3
  ➔x[3/x]
  =3

  (λx. x*x) 3
  ➔(x*x)[3/x]
  ➔ 3*3

  (λx. λy. x-y) 5 2
  ➔(λy. 2-y) 5
  ➔2 -5
  Notice the left association convention
  (λx. λy. x-y) 5 2
  ➔(λy. 5-y) 2
  ➔ 5 -2

  (λx.xx)(λy. y)
  ➔ (xx)[λy.y/x]
  = (λy.y)(λy.y)
  ➔y[λy.y/y]
  = λy.y

  (λx.λy.yx) 3 (λx.x)
  ➔(λy.yx) [3/x] (λx.x)
  ➔(λy.y3) (λx.x)
  ➔(λx.x)3
  ➔3

  (λx.λy.xy) y
  ➔(λy.xy) [y/x]
  ➔(λy.yy) ?

α conversion

α *conversions allow us to rename bound variables.*

A bound name x in the lambda abstraction (λx.e) may be substituted by any other name y, as long as there are *no free occurrences of y in e:*

λx. λy.xy ➔ λx. λz.xz

λx. λy.xy➔ λz. λy.zy

(λx. λy. x y) y

→(λ x. λz . x z) y *α conversion*

→(λ z . x z) [y/x] *β reduction*

→ (λ z . y z)

= y *η reduction*

η reduction

λ x . f x→f

η reductions allow one to remove "redundant lambdas".

Suppose that f is a *closed expression* (i.e., there are no free variables in f).

Then:

(λ x . f x) y → f y *β reduction*

So, (λ x . f x) behaves the same as f !

η reduction says, *whenever x does not occur free in f*, we can rewrite (λ x . f x) as f.

More example

- λ x y .E is a shorthand notation for λx. λy .E

(λ x y . x y) (λ x . x y) (λ a b . a b)

→(λ x z . x z) (λ x . x y) (λ a b . a b) *α conversion*

→(λ z . (λ x . x y) z) (λ a b . a b) *β reduction*

→(λ x . x y) (λ a b . a b) *β reduction*

→(λ a b . a b) y *β reduction*

→(λ b . y b) *β reduction*

→y *η reduction*

There can be more than one path of reductions

$(\lambda\, x\, y\, .\, x\, y)\ (\lambda\, x\, .\, x\, y)\ (\lambda\, a\, b\, .\, a\, b)$

$\rightarrow(\lambda\, x\, z\, .\, x\, z)\ (\lambda\, x\, .\, x\, y)\ (\lambda\, a\, b\, .\, a\, b)$ *α conversion*

$\rightarrow(\lambda\, z\, .\, (\lambda\, x\, .\, x\, y)\ z)\ (\lambda\, a\, b\, .\, a\, b)$ *β reduction*

$\rightarrow(\lambda\, z\, .\, z\, y)\ (\lambda\, a\, b\, .\, a\, b)$ *β reduction*

$\rightarrow(\lambda\, a\, b\, .\, a\, b)\ y$ *β reduction*

$\rightarrow(\lambda\, b\, .\, y\, b)$ *β reduction*

$\rightarrow y$ *η reduction*

Normal form

A lambda expression is in normal form *if it can no longer be reduced by beta or eta reduction rules.*

Not all lambda expressions have normal forms!

$\Omega = (\lambda x . x x) (\lambda x . x x)$

$\rightarrow$ $[(\lambda x . x x) / x] (x x)$

$=$ $(\lambda x . x x) (\lambda x . x x)$ *β reduction*

$\rightarrow$ $(\lambda x . x x) (\lambda x . x x)$ *β reduction*

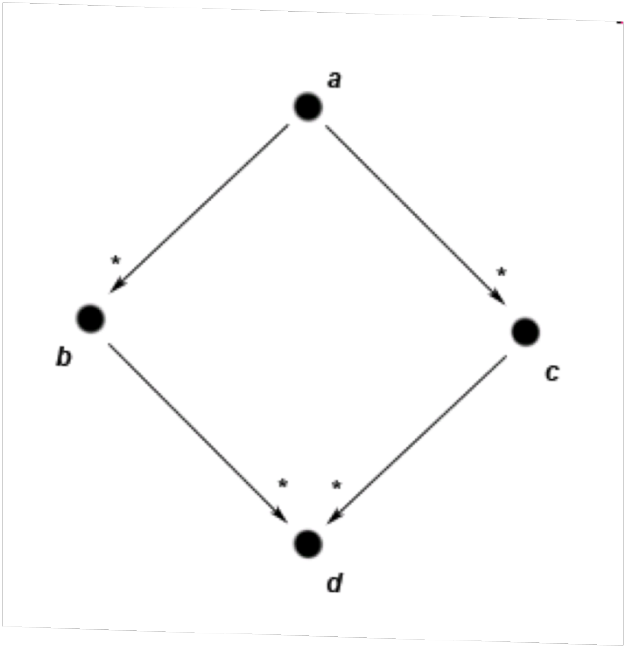
$\rightarrow$ $(\lambda x . x x) (\lambda x . x x)$ *β reduction*

$\rightarrow$ ...

Reduction of a lambda expression to a normal form is analogous to a *Turing machine halting* or a *program terminating.*

Church-Rosser Theorem

- There is only one normal form for any Lambda term, if it exists
- if term *a* can be reduced to both *b* and *c*, then there must be a further term *d* to which both *b* and *c* can be reduced.
  - d is possibly equal to either *b* or *c*



Arithmetics

- Church Numbers

  0 λsz.z

  1 λ sz.s(z)

  2 λsz.s(s(z))

  3 λsz.s(s(s(z)))

  …

- Successor function

  S λwyx.y(wyx)

  S0= (λwyx.y(wyx)) (λsz.z)

  =λ yx.y((λ sz.z)yx)

  =λ yx.y(x)

  =1

  S1= 2

  …

- Plus

  + λmn.mSn

  + 2 3 = (λ mn.mSn) 2 3

  = 2 S 3

  = λsz.s(s(z)) S 3

  =S(S(3))

  …

  =5