Lambda Calculus

History

- Introduced by Alonzo Church in 1930s as a way of formalizing the concept of effective computability.
- Any computation solved by a Turing machine can be expressed in lambda(λ) calculus.
- Universal in the sense that any **computable** function can be expressed and evaluated using this formalism.
- Who uses λ calculus as computational model?
- Purely functional languages.

Syntax

Exp $\longrightarrow$ constant|variable|Exp Exp|λ variable.Exp

The only keywords used in the language are λ and dot.

Examples:

1. 10
2. X
3. X 6
4. λy.y 2
5. (λx.x) (λy.y)
6. λx.(+ x 5)

Simplifications

- λ calculus treats functions “anonymously” i.e. without giving them explicit names.
- λ calculus only uses functions of a single input. If a function requires 2 or more inputs they can be reworked into an equivalent function that accepts a single input that accepts a single input and as output returns another function, that in turn accepts a single input. For example

    Sum=x+y

In lambda calculus it will be

    λx.λy.(+ x y)

Free Variables

- The **free variables** of an expression are those variables not bound by a lambda abstraction.
- An occurrence of a variable in an expression is free if it refers to a variable introduced (by a λ-abstraction) outside of the expression.
- Example:
  - λx.x : x is bound and **not a free variable**
  - x: x is free in expression
  - λx.x y: x is bound and y is a free variable
  - (λx.x) (λy.y x): x is bound in the first expression from the left . y is bound in the second expression but x is free in the second expression.
    - Please notice the important fact that x in the second expression is totally independent of the x in the first expression.

Examples

1. λx.λy.xyz
2. (λx.λy.((λz.x)(λy.z)))
3. (λz.zz)z
4. (λx.λy.λz.(zy(λw.x)))
5. (λx.w(λw.(y(λz.(f (λf .f))))))

1. z is free and x and y are not free
2. z is free and x is not
3. z outside is free
4. No free variables found
5. w,y,f (outside the last lambda) are free but f in (f (λf .f) is not free

Substitution rules

- Based on an operation on syntax of λ expression
- Rule 1: x [M/x]=M
  - x[M/x] is an expression
  - Substitute all appearances of x with M we get M
- Rule 2: c [M/x]=c
  - c is a constant.
  - Since there are no expressions with x so you will get a constant
- Rule 3: y [M/x]=y
  - x and y are different variables
  - y is an expression
  - Substitute all x with M but since x and y are different and y does not contain x we get the result y

Substitution rules

- Rule 4: (E1 E2) [M/x]=(E1[M/x])(E2[M/x])
  - E1 and E2 are expressions. Substitute all occurrences of x with M in E1 and then in E2
- Rule 5: (λx.E) [M/x] = (λx.E)
  - E is an expression
  - x is bound to the lambda expression

Substitution only replaces free occurrences of the variables

Substitution rules

- Rule 6: (λy.E) [M/x] x and y are different variables
  - =(λy.(E[M/x])) if y is not free in M
  - =(λz.(E[z/y]) [M/x]) where z is not free in E or M

  Let M have a y which is a free variable then it will be a problem as there is local (bounded) y.

  Let E be (+ y x) and M be (*y 3), the above expression will be

  (λy.(+ y x)) [(* y 3)/x]= (λy.(+ y (*y 3)))

  because y is a bound variable it will be problematic. This is called name capture. To solve this problem replace y with another variable z in the lambda function. So the lambda expression becomes (λz.(+ z x)) [(* y 3)/ x]=(λz.(+ z (* y 3))

Conversion Rules

- α conversion:

$$(\lambda x.E) \underset{\alpha}{\Leftrightarrow} (\lambda y.E[y/x])$$

where y is not free in E.

Intuitively it does not matter what we call the local variables till we use the variables names consistently.

Example:

In scheme

$$(\text{lambda } (x)\ (+\ x\ 3)) \underset{\alpha}{\Leftrightarrow} (\text{lambda } (y)\ (+\ y\ 3))$$

Conversion Rules

- β conversion:
  $(\lambda x.E)\ M \underset{\beta}{\Leftrightarrow} E[M/x]$
  - Application of M to (λx.E)
  - Example:
    1. $(\lambda x.(+\ x\ x))\ (+\ 3\ 4) \underset{\beta}{\Leftrightarrow} (+\ (+\ 3\ 4)\ (+\ 3\ 4))$
    2. $(\lambda x.(+\ x\ 10))\ 6 \underset{\beta}{\Leftrightarrow} (+\ 6\ 10)$

Conversion Rules

- η conversion (Eta conversion):

  (λx.E x) ⇔ E where x is not free in E

  - λx is a function and x is a parameter and it passes x directly to something else
  - If two functions are the same iff they give the same results for all arguments.
  - Example:
    - (define (f x) (g x))

Conversion rules

- δ conversion:
  - Gives meaning to constant operations (+,-,* etc)
  - (+ 1 3) $\underset{\delta}{\Leftrightarrow}$ 4
  - if true E1 E2 ⇔ E1

β Reduction

- It is a β conversion in => direction.
- $(\lambda x.E)\ M \underset{\beta}{=>} E[M/x]$

δ Reduction

- It is a δ conversion in => direction.

When used left to right, the β-conversion, δ-conversion and η-conversion rules are called β-reduction, δ-reduction and η-reduction, respectively, and the arrow is written as =>

Example of β reduction

(λx.+ x x)((λy.* y 2) 3)

- There are 2 β conversions possible
  - Apply to first expression (x expression)
  - Apply to expression of y.
- First β conversion:

(λx.+ x x)((λy.* y 2) 3) $=>_\beta$ (λx.+ x x)(* 3 2) $=>_\beta$ (+ (* 3 2) (* 3 2)) $=>_\delta$ (+ 6 (* 3 2)) $=>_\delta$ (+ 6 6) $=>_\delta$ 12

- Second β conversion:

(λx.+ x x)((λy.* y 2) 3) $=>_\beta$ (+ ((λy.* y 2) 3) ((λy.* y 2) 3)) $=>_\beta$ (+ (* 3 2) ((λy.* y 2) 3)) $=>_\beta$ (+ (* 3 2) (* 3 2)) $=>_\delta$ (+ 6 (* 3 2)) $=>_\delta$ (+ 6 6) $=>_\delta$ 12

Normal Form

- An expression to which reduction cannot be applied is said to be in normal form.
- It is not always possible to reach to a normal form.
- Example:
  - $(\lambda x.x\ x)\ (\lambda x.x\ x) \Rightarrow_{\beta} (\lambda x.x\ x)\ (\lambda x.x\ x)$
  - Each x is applied with $(\lambda x.x\ x)$, hence there is no normal form.

- Does it matter which order of β conversion part of the expression is chosen?
- ((λy.3)((λx.x x) (λx.x x))
  - Outermost β reduction
    - $=>_{\beta}$ 3
  - What about the innermost β reduction?
    - $=>_{\beta}$ ((λy.3)((λx.x x) (λx.x x))
    - It will never end.

Thus, β reduction order does matter!!

Orders of evaluation:

- Applicative order: Always choose the leftmost innermost redex to reduce.
  - i.e. all the arguments are evaluated when the procedure is applied
  - Example:
    - $(\lambda x.+\ x\ 1)((\lambda y.y)\ 3) \Rightarrow_{\beta} (\lambda x.+\ x\ 1)\ 3$
- Normal order: Always choose the leftmost, outermost redex to reduce.
  - That is, whenever possible the arguments are substituted into the body of an abstraction before the arguments are reduced
  - Example:
    - $(\lambda x.+\ x\ 1)((\lambda y.y)\ 3) \Rightarrow_{\beta} (+\ ((\lambda y.y)\ 3)\ 1)$

Which is better?

- Most programming languages use applicative order for evaluation.
- But applicative order is not a normalizing strategy.
- Normal order may take more steps to complete but may sometimes be more efficient than applicative order as normal order may terminate where as applicative order does not.
- Example:
  - $((\lambda x.2)((\lambda x.x\ x)\ (\lambda x.x\ x)))$
  - The above expression will not terminate with applicative order but will terminate with normal order

Church Rosser Theorems

- The Church-Rosser theorem says that if two terms are convertible, then there is a term to which they both reduce.
  - If E1 ⇔ E2 , then there exists an expression E such that E1 => E and E2 => E
- No expression can be can be reduced to two distinct normal forms.
  - Start with some expression choose 1 set of reductions to make and then choose other set of expressions, one cannot end up with two different normal forms.
- If E1 reduces to E2, where E2 is in normal form and if there is any reduction from E1 to E2, then there is a normal order reduction.
  - That is, normal order evaluation is most likely to terminate.

Recursion in lambda calculus

- Y combinator is used to express recursion in the λ-calculus.
  - Built-in function
- Properties:
  - Y f=f(Y f)
    - It is also called fixed point combinator.
    - Given a function f, Y returns the “fixed point” of f
    - The fixed point of a function: f is a value of x such that f(x)=x.
- How to represent Y as an ordinary λ expression?
  - Y = (λh. (λx. h (x x)) (λx. h (x x)))

Derivation of Yf=f(yf) where Y is represented as ordinary lambda expression.

$Y = (\lambda h.\ (\lambda x.\ h\ (x\ x))\ (\lambda x.\ h\ (x\ x)))$

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

$\quad =>_{\beta} (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$

$\quad =>_{\beta} f\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x)) = f\ (Y\ f)$