[**Lambda Calculus**](http://en.wikipedia.org/wiki/Lambda_calculus)

**T = cons | T T | variable | λ v.e**

**T** –“application” **v**- variable, **e**-term, **λ v.e** – “lambda experssion” or “abstract”

**Terms examples:**

**z**

**xyz** [means **(xy)z** , ie. “left associative”]

**x(yz)**

**binds very strongly**

**λ x. xyx means (λ x. ((xy)x))**

**(λ x. xy)x**

* All functions takes only one variable
* Don’t put parenthetic if they not necessary

**“Sugar”**

**λ f . λ x(f x) (f x) -curry function**

**λ f x . λ x(f x) (f x) sugar for**

**FV** - “free variable ”

FV: term -> {variable}

FV const = Ø

FV v = {v}

FV (e1e2) = FV e1 U FVe2

FV (λ v . e) = (FV e) \ {v}

Lc – is a logic, not a programming language, eg. algebra = calculus

**“reduction”**

🡪β – reduction

* (λ v. e1) e2 --> e1 [v <- e2]

substitute e2 for (free) occurrences of v in e1

* const [v <- e] = const
* v [v <- e] = e
* u [v <- e] = u (u != v)
* (e1 e2 ) [v <- e3] = (e1 [v <- e3] e2 [v <- e3]
* (λ v e1) [v <- e2] = λ v e1
* (λ v1 e1) [v2 <- e2] = λ v1 (e1[v2 <-e2] **where v1 != v2 and v1 != FV (e2)**

**Abbreviations**

zero = λ f x. x

succ = λ n f x. n f (f x)

one = succ zero --> λ f x. f x

plus = ...

times = ...

succ zero = (λ n f x. n f (f x)) (λ f x. x)

--> (λ f x. n f (f x)) (n <-λ f x. x)

= λ f x. λ f x. x f (f x))

--> λ f x.( λ x. x) [f <- f] (f x)

= λ f x.( λ x. x) (f x)

--> λ f x.x [ x <-f x]

= λ f x. f x

succ (succ zero) ->? λ f x. f (f x)

|  |  |
| --- | --- |
| (λ x. (λ y . xy))y  --> (λ y . xy)) [x <- y]  = λ y | λ x. (λ z. x z) y  --> (λ z. x z) [x <-z]  = λ z. yz |

€ FV (.)