Lecture 9-10 **13-14.2.2025**

Today's agenda:

Church numerals, successor function, primitive recursive functions

Church numerals

$0 = \lambda f.\lambda y.y$
$1= \lambda f.\lambda y.\ f\ y$
$2= \lambda f.\lambda y.\ f\ (f\ y)$
$n= \lambda f.\lambda y.\ f\ (f....(f\ y))...)$ n times

n f y => f and y are extracting the body of n and then one more f is being applied increasing the value by 1.
This technique is called NFI (killing the lambda term)

Successor is defined as: succ = $S = \lambda n.\ \lambda f.\lambda y.\ f\ (n\ f\ \ y)$

Now we want to design the successor function. How did we get the term for successor function?

Suppose that it is correct. We shall work out for S0 and S1 to get the insight.

$S0 = (\lambda n.\lambda f.\lambda y.\ f(n\ f\ y))(\lambda f.\lambda y.y)$
$\quad = (\lambda n.\lambda f.\lambda y.\ f(n\ f\ y))(\lambda u.\lambda v.v)$ [by renaming]
$\quad =\lambda f.\lambda y.f\ ((\lambda u.\lambda v.v)f\ y)$
$\quad =\lambda f.\lambda y.f\ (\ (\lambda u.\lambda v.v)\ f)\ y)$ [by left associative]
$\quad =\lambda f.\lambda y.f\ ((\lambda v.v)y)$
$\quad =\lambda f.\lambda y.f\ y$ corresponds to numeral 1

Since n is an argument of S so we add λn.

Now succ of n is n' = n+1; now n' is represented as $\lambda f.\lambda y.$ (something)

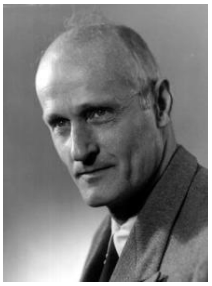
Where the (something) is < f (the body of n) >. Let us call it f M. when M is applied to n we get the body of n. So n must occur in M. Now n would be replaced by the actual parameter n.

But the actual parameter contains λf and λy. So in order to get rid of them, supply the corresponding parameters. Thus in doing so, we get only the body of n.

$S1 = (\lambda n.\ \lambda f.\lambda y.\ f(n\ f\ y))(\ \lambda f.\lambda y.\ fy)$
$\quad = \lambda f.\lambda y.f((\lambda f.\lambda y.fy)fy))$
$\quad = \lambda f.\lambda y.f\ (\ (\lambda y.fy)y)$
$\quad = \lambda f.\lambda y.f\ (fy)$

However, designing the lambda term for predecessor function is hard (invented by Kleene during a visit to a dentist!).

The implementation of the predecessor function is involved. In fact, Church thought for a long time that it might not be possible, until his student Kleene found it. In fact, there is a legend that Kleene conceived the idea while visiting his dentist, which is why the trick is called the wisdom tooth trick.



Stephen Kleene (1909-1994) Church's student and a pioneer of LC. He invented regular expressions. Kleene star (Kleene closure) is named after him. His works helped to provide the foundations of theoretical computer science.

primitive recursive functions:

add(m, 0) = m   add(m, n+1) = succ(add(m,n))
multiply(m,1) = m   multiply(m,n+1) = add(m, multiply(m,n))
exponent(m,0) = 1   exponent(m,n+1) = multiply(m, exponent(m,n))

Successor function: succ = λn. λf.λy. f(n f y)

generalize the successor function to adding m. so we have plus = λm. λn.λf. λy. m f(n f y)

Recall the factorial function: f = IF (iszero n) S0 n* f (pred n)

now we have a pure lambda term for f. the components are (i) IF (ii) 0 (iii) S (iv) iszero n (v) * (vi) pred

everything can be done using pure lambda calculus

Design of iszero function: iszero = λn. n (λx. false) true

iszero = λn. n (λx. false) true   true = λf. λy. f   false = λf. λy. y

~~λn.~~n
[λf. λy. y…( ) ……true…..]   beta reduction true or false
= (λy. y) true
= true

iszero 0 = (λn. n (λx. false) true) 0
= (λn. n (λx. false) true) λf. λy. y
= ((λf. λy. y) (λx. false)) true
= (λy. y) true
= true

iszero 1 = (λn. n (λx. false) true) 1
= (λn. n (λx. false) true) (λf.λy.fy)
= (λf. λy. f y) (λx. false) true
= ((λf. λy. f y) (λx. false))true
= (λy. ((λx. false) y)) true
= (λy. false) true
= false

End of lecture