Lecture 8 **10.2.2025**

Today's agenda:

Recursion in LC

The Y combinator denoted by Y

$Y = \lambda t. (\lambda x. t\ (x\ x))\ (\lambda x. t\ (x\ x))$ compare any subterm with $sa = \lambda x.\ x\ x$

Here sa is modified as: $sa' = (\lambda x. \underline{t}\ (x\ x))$ the t is introduced

Now we do sa' sa' and ensure that it is a closed term. So we obtain

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

Recursive function: eg factorial function—say f

f n = if n=0 then 1 else n* f (n-1)

We will show later how the other symbols can be expressed in LC.

--we need a lambda term for zero. Then we have the successor function.
--successor function: S0 = 1, SS0 =2, and so on.
--predecessor function: pred 1 = 0, pred 2 = 1, and so on, undefined for 0.
thus, 1 = S0, n-1 = pred n;
*,- are primitive recursive functions that can be obtained using 0, Successor function in LC. IF can be defined in LC.

For equality '=' we need to define the function 'iszero' where iszero 0 is true and false otherwise.

Thus rewriting the above as f n = IF (iszero n) S0 n* f (pred n)

Recursive function: eg factorial function—say f

f n = if n=0 then 1 else n* f (n-1) f n = IF (iszero n) S0 n* f (pred n)

So let G = λf. λn. if n=0 then 1 else n* f (n-1), (non recursive)

G = λf. λn IF (iszero n) S0 n* f (pred n) so f = Y G

Now Y G = G (Y G) by definition, let n=1

Y G 1 = G (Y G) 1

= (G (YG)) 1 by left associativity how to apply recursion.

= ((λf. λn. if n=0 then 1 else n* f (n-1)) (YG)) 1

= (λn. if n=0 then 1 else n*(YG)(n-1))1
= if 1=0 then 1 else 1*(YG)(1-1)
= 1*(YG)(1-1)
= 1*G(YG)(0)
=1*[(lf. λn. if n=0 then 1 else n* f (n-1)) (YG))] (0)
= 1* (λn. if n=0 then 1 else n*(YG)(n-1))0
= 1* (if 0=0 then 1 else 0*(YG)0-1
=1*1 = 1

In this case why did the recursion terminate? It is because we have included a base case of the recursion.

Note: The Y combinator is based on CBN. If we want to use CBV, we need a different combinator.

The specialty of the Y combinator:

Fixed point = fix point It means that for a function f and some argument x we have f x = x;

For HOF, x is a function, so f p = p (p is called the fixed point of f). p can be a function itself, but is called fixed point of f.

fixed point generator will take f, and return its fixed point

A fixed point generator is a function that generates a fixed point for f;

So g f = p, (1) also fp = p (2)

substitute g f for p in (2)

f (gf) = gf or gf = f (gf) or

Yt = t (Yt)

Thus Y is called the fixed point combinator. the Y used in recursion is a fixed-point generator. Special property of Y is that it will always return a fixed point.

Another example: plus (+)

--we need a lambda term for zero. Then we have the successor function S

--successor function: S0 = 1, SS0 =2, and so on.

--predecessor function: pred 1 = 0, pred 2 = 1, and so on, undefined for 0.

thus, 1 = S0, n-1 = pred n, n+1 = succ n

x + y = y if x = 0 otherwise (pred x) + (succ y)

add = λx.λy. if x=0 y (add (pred x) (succy)) which is recursive

Claim: plus = Y M where M = λadd. λx.λy. if x=0 y (add (pred x) (succy))

Exercise: verify the above claim by working out an example.

End of lecture