Lecture 6 **6.2.2025**

Today's agenda:
Computation with pure LC
Reductions in LC

Computation with pure LC : example [α-axiom, β-axiom]

Eg: S = (λxyz. ((x z) (y z))) I = (λx.x) Compute SII

For the sake of illustration, let us remove some parentheses s.t . the meaning remains same

S = λxyz. (x z) (y z)

SII = (λxyz. (x z) (y z)) (λx.x) (λx.x)

= ((λxyz. (x z) (y z)) (λx.x)) (λx.x) SII = (SI)I since fgh = (fg) h

We want to have unique names: no repetition of names, so use α-axiom

$=_\beta$ ((λxyz. (x z) (y z)) (λu.u)) (λx.x) [portion in red above where x is replaced with u]

Now we rename the x on the rightmost side to get

$=_\beta$ ((λxyz.(x z) (y z)) (λu.u)) (λv.v) form M N P

use β-axiom: replacing x by λu.u

$=_\beta$ ((λyz. ((λu.u) z) (y z)) (λv.v) A

$=_\beta$ ((λyz. (z (y z))) (λv.v) B

Now we have to apply, so y would be replaced by λv.v

$=_\beta$ λz. (z ((λv.v) z)) F

$=_\beta$ λz. z z E which is in normal form--cannot be reduced any further.

Normal form means simplest form

Eg, (2+1) * (3+2) + (5+2)*2
=(3)*(5) + (7)*2
=15+14
=29—normal form

An abstraction is always in a normal form, if the body is in normal form. We need to apply beta rules when a term is of the form M N.

Reductions *Term rewriting is another way of doing computation.*

There are two types of reduction of lambda terms: CBN (call-by-name), CBV(call-by-value).

The terms would be of the form (M N)

CBN: take N as it is and replace x in M as above.

CBV: simplify N as much as possible (normal form) and then the resultant term would be applied for M.

Example:

Eg1, (λx. x + x) ((λ y. y) 5) of the form M (N P)

CBN: = (λ y. y)5) + ((λ y. y)5) = 5+5 = 10 replace x with (N P) as it is

CBV: = (λ x. x + x) 5 = 5+5 = 10. Simplify (N P) to say v and then replace x with v

CBV semantics says that N would be reduced to a value before calling the function.

In CBN: N is evaluated only when needed. This is called lazy rewriting. (call by need)

Haskell uses CBN, and lazy rewriting is an important feature in Haskell.

CBN is guaranteed to reach a normal form, if one exists;

CBV may get stuck, forever evaluating an argument that would never be used.

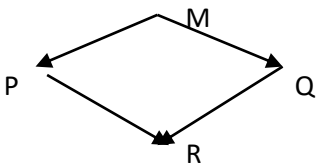
Eg2, (λx. z + z) ((λ y. y) 5) there is no free occurrence of x; so the answer would be z+z.
But CBV would evaluate (N P)—which as we can see is not required in this case.

What if (N P) is non-terminating? CBV would get stuck although there exists a normal form.

Church-Rosser Theorem: For all pure lambda terms M, P, Q, if M reduces to P (M =>* P) and M reduces to Q (M =>* Q) then there must exist a term R such that P=>*R and Q=>*R. M =>* P means zero or more alpha renaming and beta-reductions.

The theorem says that order of reduction does not influence the end result. The end result is in normal form, if one exists.

The above property is also called diamond property or confluence diagram.



Exercise: (i) Give another way of computing SII. (ii) Give all possible ways of computing SII. (iii) Draw the confluence diagram.

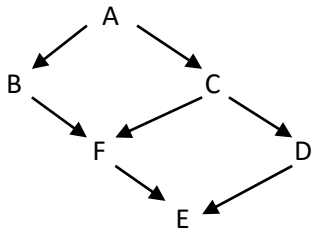
Another way of computing SII:

$=_\beta$ (($\lambda yz.$ (($\lambda u.u$) z) (y z)) ($\lambda v.v$)) A

$=_\beta$ $\lambda z.$ (($\lambda u.u$) z) (($\lambda v.v$) z) C

$=_\beta$ $\lambda z.$ ($\lambda u.u$) z z D

$=_\beta$ $\lambda z.$ z z E u is replaced by z z



End of lecture