Lecture 12-13 **20-21.2.2025**

Today's agenda:

Primitive recursive functions
Design of arithmetic operations
--

Basic functions:
$N = \{ 0, 1, \cdots, \}$

(a) For any $k \geq 0, zero_k(n_1 \cdots n_k) = 0$ for all $n_1 \cdots n_k \in N$ [k-ary zero function]
(b) For any $k \geq j > 0$, $id_{\{k,j\}}(n_1 \cdots n_k) = n_j$ for all $n_1 \cdots n_k \in N$ [k-ary identity function]
(c) $succ(n) = n + 1$ for all $n \in N$

Complex functions:

(1) Let $k, l \geq 0$, $g : N^k \mapsto N$ (g: k-ary), $h_1, \cdots, h_k$ be l-ary function. Composition of g with h is a l-ary function f: $f(n_1, \cdots, n_l) = g\big(h_1(n_1, \cdots, n_l), \cdots, h_k(n_1, \cdots, n_l)\big)$
(2) let $k \geq 0$, g: k-ary, h: k+2 ary, f: k+1 ary recursively defined
$f(n_1, \cdots, n_k, 0) = g(n_1, \cdots, n_k)$
$f(n_1, \cdots, n_k, m + 1) = h\big(n_1, \cdots, n_k, m, f(n_1, \cdots, n_k, m)\big)$ for all $n_1 \cdots n_k \in N$

Primitive recursive functions are all basic functions and all functions that can be obtained by any number of successive applications of composition and recursion.

Examples:

1. plus2(n) = n + 2
   in (1) let l=1, k=1, g = h_1 = succ
   plus2(n) = succ (succ(n))

2. plus(n,m) = n + m
   in (2), k=1, $g = id_{\{1,1\}}$
   $plus(n, 0) = id_{\{1,1\}}(n) = n$
   $plus(n, m + 1) = h(n, m, plus(n, m))$
   $h(n, m, plus) = succ\left(id_{\{3,3\}}(m, n, plus)\right)$
   $plus(n, m + 1) = succ\big(plus(n, m)\big)$

3. $mult(m, n) = m \cdot n$
   $mult(m, 0) = zero(m) = 0$
   $mult(m, n + 1) = h(m, n, mult(m, n))$
   $h = plus\left(m, id_{\{3,3\}}(m, n, mult)\right)$
   $mult(m, n + 1) = plus\big(m, mult(m, n)\big)$

4. $\exp(m, 0) = succ\ (zero(m)) = 1$
   $\exp(m, n + 1) = mult(m, \exp(m, n))$

Arithmetic

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

$S = \lambda n.\ \lambda f.\lambda y.\ f\ (n\ f\ y)$ successor function

plus = $\lambda m.\ \lambda n.\ \lambda f.\ \lambda y.\ m\ f\ (n\ f\ y)$

multiply = $\lambda m.\ \lambda n.\ \lambda f.\ \lambda y.\ n\ (m\ \ f\ )\ y$

exponent = $\lambda m.\ \lambda n.\ \ n\ m$ (to encode m^n) Note: there is no $\lambda f.\ \lambda y$

Consider any primitive recursive function encoding given above. Can we design a TM for the function, i.e. a TM encoding? Yes. Can we give an encoding in pure LC? Yes. From this we can say that LC and TM are equivalent.

To put it simply, whatever can be done using one of them can also be done using the other. Moreover, if something cannot be done in one of them, then it cannot be done in the other.

Given the above encodings for numerals, how to prove 1+1 = 2?

Method1: LHS: use successor of 1 and verify that the result is the encoding of 2 (RHS).

Method2: LHS: use plus and verify that the result is the encoding of 2 (RHS).

--

Design of the plus function:

$S = \lambda n.\ \lambda f.\lambda y.\ f\ (n\ f\ y)$ successor function let plus = $\lambda m.\ \lambda n.\ \lambda f.\lambda y.\ M\ N$
now, plus(m,n) will have (m+n)-f s. we know how to extract n-f s. so N = (n f y).
if we have a placeholder to replace it with these n-f s, then we are done.
now, $m = \lambda f.\lambda y.\ \ f(f\ldots.(fy))\ldots)$ m times, after consuming m and n in plus we have:
$\lambda f.\lambda y.\ M\ N$

Since m has to occur in (M N), so let M = m; now we get $\lambda f.\lambda y.\ \ \lambda f.\lambda y.\ \ f(f\ldots.(fy))\ldots)\ \ N$

We need to get rid of the inner λf and λy. We need two arguments: now we have one –i.e., N. so introduce the argument f to consume λf. N will consume λy. We now have (m+n)-number of f s. thus M = m f and so we have plus = $\lambda m.\ \lambda n.\ \lambda f.\lambda y.\ m\ f\ (n\ f\ y)$

End of lecture