CMPT 409: Theoretical Computer Science	**Summer 2017**

Lecture 4: Computability (12 - 16 June)

Lecturer: Ternovska, Eugenia *Scribe: Lily Li*

4.1 Functions

Let ∞ denote *undefined*.

Definition 4.1 *A* ***partial function*** *is a funtion* $f : (\mathbb{N} \cup \{\infty\})^n \to \mathbb{N}\{\infty\}$ *such that* $f(c_1, ..., c_n) = \infty$ *if some* $c_i = \infty$.

The ***domain*** *of function* f *is* $Dom(f) = \{\bar{x} \in \mathbb{N}^n : f(\bar{x}) \neq \infty\}$ *where* $\bar{x} = (x_1, ..., x_n)$. *If* $Dom(f) = \mathbb{N}^n$ *then* f *is* ***total***.

4.2 Register Machines

A register machine RM is a finite set of registers, $R_1, ..., R_m$, each storing some natural number. Program $P = \langle c_0, ..., c_{h-1} \rangle$ where each c_i is a command. The list of allowed commands are:

1. z_i: zero register R_i.
2. S_i: increment register R_i.
3. $J_{i,j,k}$: jump to command c_k if $R_i = R_j$.

The **state** of a RM is $\langle k, R_1, ..., R_m \rangle$ where k is the index of the next command to execute and R_i the content of register i.

The map $\mathsf{Next}_P(s)$ returns the state resulting from the execution of one step of the program P in state s. State s_i is **halting** if $k = h$. Then $\mathsf{Next}_P(s) = s$. For current state $s = \langle k, R_1, ..., R_m \rangle$, $\mathsf{Next}_P(s)$ is defined as:

1. If $k < h$ and $c_k = s_i$ then the new state is $\langle k + 1, R_1, ..., R_i + 1, ..., R_m \rangle$.
2. If $k < h$ and $c_k = z_i$ then the new state is $\langle k + 1, R_1, ..., R_{i-1}, 0, ..., R_m \rangle$.
3. If $k < h$ and $c_k = J_{i,j,k'}$ then the new state is $\langle k', R_1, ..., R_m \rangle$ if $R_i = R_j$ and $\langle k + 1, R_1, ..., R_m \rangle$ if $R_i \neq R_j$ and $k < h$.

The computation of P is a sequence (finite or infinite) of states $s_0, s_1, ...$ such that $s_{i+1} = \mathsf{Next}_P(s_i)$ when $i \geq h$ — ending with the halting state. A program computes a function $f(a_1, ..., a_n)$ if $s_0 = \langle 0, a_1, ..., a_n, 0, ..., 0 \rangle$ and when started from s_0, P halts with $R_1 = f(a_1, ..., a_n)$. If P fails to halt, then $f(a_1, ..., a_n) = \infty$.

Conversely, for each P and $n \geq 0$, there is an n-ary function f_P computable by P. We say that such an f is *computable* if some RM computes it.

Example 4.2 *Consider how we would implement the function $f(x) = x - 1$ using a RM. First observe that we are dealing with natural numbers we need to assume that $x - 1 \geq 0$. We begin with $s_0 = \langle 0, x, 0 \rangle$. At a high level, the program should set R_3 (currently zero) to 1, then simultaneously increment R_1 and R_3 until the latter is equal to x. Formally the computation can be preformed using the following sequence — with s_i the halting state:*

$$c_0 = s_3, c_1 = J_{2,3,5}, c_2 = s_1, c_3 = s_3, c_4 = J_{1,1,1}$$

It is a good exercise to consider how you would implement $f(x, y) = x \cdot y$ as a RM.

Definition 4.3 *A function f defined from g and h by **primitive recursion** if and only if*

$$f(\bar{x}, 0) = g(\bar{x})$$
$$f(\bar{x}, y + 1) = h(\bar{x}, y, f(\bar{x}, y))$$

where $\bar{x} = x_1, ..., x_n$. $n = 0$ is allowed. Here you want to think of g as the base case and h as the recursive step.

Example 4.4 *Lets consider how to define addition using primitive recursion: we define $f_+(x, y)$ as follows: let $f_+(x, 0) = g(x) = x$ and $f_+(x, y + 1) = h(x, y, f_+(x, y)) = f(x, y) + 1$. Notice that g is essentially the identity and h is the successor function on the third argument.*

Next we use $f_+(x, y)$ to define $f_\cdot(x, y)$. The base case is $f_\cdot(x, 0) = g(x) = 0$ (see below). Let $f_\cdot(x, y + 1) = h(x, y, f_\cdot(x, y)) = x + f_\cdot(x, y)$. In particular $h(x, y, z) = f_+(I_{3,1}, I_{3,3})$ where $z = f_\cdot(x, y)$.

Finally we define exponentiation using $f_\cdot(x, y)$. Let

You should think of primitive recursion as those functions are bounded by construct. This is not true of functions in general, so RM is a strict subset of TM.

Definition 4.5 *f is defined from g and $h_1, ..., h_m$ by **composition** if an only if*

$$f(\bar{x}) = g\left(h_1(\bar{x}), ..., h_m(\bar{x})\right)$$

where $f, h_1, ..., h_m$ are h-ary and g is m-ary.

Definition 4.6 *f is **primitive recursive** if and only f can be obtained from the initial functions: $Z : 0, S : s(x) = x + 1, I_{n,i}(x_1, ..., x_n) = x_i$ where $i \leq i \leq n$ (projection).*

Example 4.7 *Consider using the above definition of primitive recursion to define the constant function $z_1(x) = 0$. Let $z_1(0) = Z = g$ (Z as in the definition). $z_1(y + 1) = z_1(y) = h(y, z_1(y))$ where $h = I_{1,1}$.*

Generally for the constant function $k_{n,i}$,

$$k_{n,i}(x_1, ..., x_n) = i$$

We can also define $k_{n,i}$ using recursion! In particular our base case is $z_1(x)$. $k_{n,i} = s(s(\cdots s(z_1(I_{i,1}))\cdot)$ where the successor function s (again defined above) is applied i times.

Theorem 4.8 *Ackerman's function is not primitive recursive.*

Proof:

∎