Computation; Notes

July 4, 2020

Contents

1 Computable Functions

1.1 Basic Concepts

§ **Partial Functions** A *partial function* generalizes the usual definition of function, the idea being that this kind of function is potentially not defined on the entire domain. Formally:

Definition 1.1. A *partial function* f from X to Y (written as $f : X \nrightarrow Y$) is a triple (g, X, Y) such that $X' \subseteq X$ and $g : X' \to Y$ is a function. Furthermore:

- The *domain* of f is denoted by $\mathsf{Dom}(f)$ and is equal to X';

- If $\mathsf{Dom}(f) = X$ then f is a *total function*[1];

- If $x \in (X \setminus \mathsf{Dom}\, f)$ then $f(x)$ is said to be *undefined*, denoted $f(x) = -$, on the other hand, if $x \in \mathsf{Dom}\, f$ then we write $f(x) = y$ with $y = g(x)$ and say that f is *defined* at x.

Henceforth the word "function" will always mean "partial function." As an example, consider the (partial) function:

$$f : \mathbb{N}_0 \nrightarrow \mathbb{N}_0$$
$$n \mapsto \sqrt{n}.$$

If $n \in \mathbb{N}_0$ is not a perfect square, then $f(n)$ is undefined.

§ **Lambda Notation** We will often use Alonzo Church's *lambda notation*. Given a mathematical expression $a(x_1, \ldots, x_n)$ the function $f : \mathbb{N}_0^n \to \mathbb{N}_0$ that maps $(x_1, \ldots, x_n) \mapsto a(x_1, \ldots, x_n)$ may be denoted by $\lambda_{x_1, \ldots, x_n}.f(x_1, \ldots, x_n)$.

1.2 The URM

§ **Informal Discussion** An *algorithm* is a finite sequence of discrete mechanical instructions. A numerical function is *effectively computable* (or simply *computable*) if an algorithm exists that can be used to calculate the value of the function for any given input from its domain.

§ **The Unlimited Register Machine** The *unlimited register machine* has an infinite number of *registers* labelled $R_1, R_2, \ldots$, each containing a natural number, if R_i is a register then r_i is the number it contains. It can be represented as follows

R_1	R_2	R_3	R_4	R_5	R_6	R_7	$\cdots$
r_1	r_2	r_3	r_4	r_5	r_6	r_7	$\cdots$

---

[1]Total functions and usual functions are equivalent.

The contents of the registers determine its *state* or *configuration*, which might be altered by the URM in response to certain *instructions*.

§ URM Programs

Name of Instruction	Instruction	URM response
Zero	$\mathtt{Z}(n)$	$r_n \leftarrow 0$
Successor	$\mathtt{S}(n)$	$r_n \leftarrow r_n + 1$
Transfer	$\mathtt{T}(m,n)$	$r_n \leftarrow r_m$
Jump	$\mathtt{J}(m,n,q)$	if $r_m = r_n$ then jump to q-th instruction; otherwise proceed to next instruction.

Without exception, the parameters of these instructions are elements of $\mathbb{N}_1$.

Definition 1.2. An *URM program* is a finite sequence of URM instructions. The number of instructions of a program is denoted by $\#P$.

Given a program $P = (I_1, \ldots, I_n)$ the URM always starts by executing I_1, the execution flow then proceeds incrementally unless a jump instruction is performed. The machine's response to each instruction is described in the table above.

Definition 1.3. An URM program P *computes* the function $f : \mathbb{N}_0^n \nrightarrow \mathbb{N}_0$ if for every $a_1, \ldots, a_n, b \in \mathbb{N}_0$ then:

$$P(a_1, \ldots, a_n) \downarrow b \Leftrightarrow (a_1, \ldots, a_n) \in \mathsf{Dom}\, f \wedge f(a_1, \ldots, a_n) = b,$$

and

$$P(a_1, \ldots, a_n) \uparrow \Leftrightarrow f(a_1, \ldots, a_n) = -$$

The class of URM-computable functions is denoted by $\mathcal{C}$ and by $\mathcal{C}_n$ the class of n-ary computable functions.

Definition 1.4. Let P be an URM program and $n \in \mathbb{N}_1$. The unique n-ary function that P computes is denoted by $f_P^{(n)}$, which, given any $x_1, \ldots, x_n \in \mathbb{N}_0$, is defined by:

$$f_P^{(n)}(x_1, \ldots, x_n)) = \begin{cases} - & \text{if } P(x_1, \ldots, x_n) \uparrow \\ y & \text{if } P(x_1, \ldots, x_n) \downarrow y \end{cases}.$$

Example 1.1. Let $Q = (\mathtt{Z}(2), \mathtt{J}(1,2,6), \mathtt{S}(2), \mathtt{J}(1,1,2), \mathtt{S}(3), \mathtt{T}(2,1))$. The binary function computed by Q is $f_Q^{(2)}(x, y) = x$.

Definition 1.5. Let $M(x_1, \ldots, x_n)$ be an n-ary predicate. The characteristic function of the predicate M is the function $C_M : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$ defined by:

$$C_M(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } M(x_1, \ldots, x_n) \\ 0 & \text{if } \neg M(x_1, \ldots, x_n) \end{cases}.$$

The predicate M is *decidable* if its characteristic function is computable and *undecidable* when it is not.

§ Building programs out of other programs

Definition 1.6. A program P is in *standard form* if, for every jump instruction $\mathtt{J}(m,n,q) \in P$ it holds that $q \leq \#P + 1$.

Definition 1.7. Let $P = (I_1, \ldots, I_n)$ be an URM program. We denote by P^* the program $(I'_1, \ldots, I'_n)$ constructed as follows:

$$I'_i = \begin{cases} \mathtt{J}(m,n,\#P+1) & \text{if } I_i = \mathtt{J}(m,n,k), \text{ with } k > \#P + 1 \\ I_i & \text{otherwise} \end{cases} .$$

Definition 1.8. Two programs P_1 and P_2 are *(strongly) equivalent* if, for any initial configuration $(a_1, a_2, a_3, \ldots)$ it holds that:

- $P_1(a_1, a_2, a_3, \ldots) \downarrow$ iff $P_2(a_1, a_2, a_3, \ldots) \downarrow$;

- when it is the case that both computations $P_1(a_1, a_2, a_3, \ldots)$ and $P_2(a_1, a_2, a_3, \ldots)$ stop, the final configurations of both machines are equal.

Theorem 1.1. *Let P be a URM program. The program P^* is in standard form and is equivalent to P.*

Definition 1.9. Let P and Q be URM programs. The *concatenation* of P and Q, denoted by $P; Q$, is the URM program defined as follows:

- $\#(P;Q) = \#P + \#Q$,

- For every $l \in \{1, \ldots, \#P\}, (P;Q)[l] = P'[l]$,

- For every $k \in \{1, \ldots, \#Q\}$:

$$(P;Q)[\#P + k] = \begin{cases} Q[k] & \text{if } Q[k] \text{ is not a jump instruction} \\ \mathtt{J}(m,n,r+\#P) & \text{if } Q[k] = \mathtt{J}(m,n,r) \end{cases} .$$

Definition 1.10. Let P be an URM program. If $\{R_{v_1}, \ldots, R_{v_n}\}$ is the set of registers mentioned in program P, we denote by $\rho(P)$ the number $\max\{v_1, \ldots, v_n\}$.

Definition 1.11. Let $n, j \geq 1$ and $i_1, \ldots, i_n > n$ and P be an URM program. We denote by $P[i_1, \ldots, i_n \to j]$ the following URM program:

$$(\mathtt{T}(i_1,1), \ldots, \mathtt{T}(i_n,n), \mathtt{Z}(n+1), \ldots, \mathtt{Z}(\rho(P))); P; (\mathtt{T}(1,j)),$$

where the sequence of instructions $\mathtt{Z}(n+1), \ldots, \mathtt{Z}(\rho(P))$ only occurs if $\rho(P) > n$.

Definition 1.12. A *generalized URM program* Q is a finite sequence of *generalized URM instructions* $(I_1, \ldots, I_k)$, with $k \geq 1$, where each each of this sequence's elements is either a standard URM instruction or one of the following two (called P-calling instructions):

1. $\mathtt{CallP}$,

2. $\mathtt{CallP}[i_1, \ldots, i_n \to j]$,

where $n, j \geq 1$ and $i_1, \ldots, i_n > n$ and P is an URM program which does not contain instructions that call program Q or instructions that call other programs that call program Q.

1.3 Primitive and Partial Recursive Functions

Definition 1.13. The following functions are called *basic functions*:

1. The *zero* functions: $zero : \lambda_x \cdot 0$,

2. the *successor* function: $suc = \lambda_x \cdot x + 1$,

3. for each $n \in \mathbb{N}_1$ and $i \in \{1, \ldots, n\}$, the *projection* function: $U_i^n = \lambda_{x_1, \ldots, x_n} \cdot x_i$.

Theorem 1.2. *The basic functions are URM-computable.*

Definition 1.14. Let $f : \mathbb{N}_0^k \twoheadrightarrow \mathbb{N}_0$ and $g_i : \mathbb{N}_0^n \twoheadrightarrow \mathbb{N}_0$, for each $i \in \{1, \ldots, k\}$. Define $g = (g_1, \ldots, g_k)$ as

$$g : \mathbb{N}_0^n \twoheadrightarrow \mathbb{N}_0^k$$
$$(x_1, \ldots, x_n) \mapsto g(x_1, \ldots, x_n)$$
$$\simeq (g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n)).$$

The composition of f and $g = (g_1, \ldots, g_k)$, denoted $f \circ g$, is the following function:

$$f \circ g : \mathbb{N}_0^n \twoheadrightarrow \mathbb{N}_0$$
$$(x_1, \ldots, x_n) \mapsto (f \circ g)(x_1, \ldots, x_n) \simeq f(g(x_1, \ldots, x_n))$$
$$\simeq f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n)).$$

Both of these functions are defined at a point $(x_1, \ldots, x_n)$ if and only if $g_1, \ldots, g_n$ are defined at $(x_1, \ldots, x_n)$.

Theorem 1.3. *Let $f : \mathbb{N}_0^k \twoheadrightarrow \mathbb{N}_0$ and $g_i : \mathbb{N}_0^n \twoheadrightarrow \mathbb{N}_0$, for each $i \in \{1, \ldots, k\}$, are computable functions, then the function $h = f \circ (g_1, \ldots, g_k) : \mathbb{N}_0^n \twoheadrightarrow \mathbb{N}0$ defined, for any $(x_1, \ldots, x_n) \in \mathbb{N}_0^n$, by $h(x_1, \ldots, x_n) \simeq f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$, is also computable. In other words, composition preserves computability.*

Definition 1.15. Let $k \in \mathbb{N}_0$ and $n \in \mathbb{N}_1$. We denote by $k^{(n)}$ the n-ary constant function define as follows:

$$k^{(n)} : \mathbb{N}_0^n \twoheadrightarrow \mathbb{N}_0$$
$$(x_1, \ldots, x_n) \mapsto k^{(n)}(x_1, \ldots, x_n) = k$$

Theorem 1.4. *For every $k \in \mathbb{N}_0$ and every $n \in \mathbb{N}_1$ the function $k^{(n)}$ is obtained by composition of basic functions.*

1.4 Turing Machines

Definition 1.16. A *Turing Machine M* is a tuple $(Q, \Gamma, \beta, \Sigma, \delta, q_0, F)$.

Definition 1.17. A function $f : \mathbb{N}_0^n \twoheadrightarrow \mathbb{N}_0$ is *Turing-computable* if there exists a Turing machine that computes f.

The class of Turing-computable functions is denoted by $\mathcal{T}$.