

Computation; Notes

July 21, 2020

Contents

1	Computable Functions	2
1.1	Basic Concepts	2
1.2	The URM	2
1.3	Primitive and Partial Recursive Functions	5
1.4	Bounded and Unbounded Quantification	6
1.5	Alternative Models of Computability	6

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 $f : X \rightharpoonup Y$) is a triple (g, X', Y) such that $X' \subseteq X$ and $g : X' \rightarrow Y$ is a function. Furthermore:

- The *domain* of f is denoted by $\text{Dom}(f)$ and is equal to X' ;
- If $\text{Dom}(f) = X$ then f is a *total function*¹;
- If $x \in (X \setminus \text{Dom } f)$ then $f(x)$ is said to be *undefined*, denoted $f(x) = -$, on the other hand, if $x \in \text{Dom } f$ 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:

$$\begin{aligned} f : \mathbb{N}_0 &\rightharpoonup \mathbb{N}_0 \\ n &\mapsto \sqrt{n}. \end{aligned}$$

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 $a(x_1, \dots, x_n)$ the function $f : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$ that maps $(x_1, \dots, x_n) \mapsto a(x_1, \dots, x_n)$ may be denoted by $\lambda_{x_1, \dots, x_n}.f(x_1, \dots, x_n)$.

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, \dots , 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	\dots
r_1	r_2	r_3	r_4	r_5	r_6	r_7	\dots

¹Total 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	$Z(n)$	$r_n \leftarrow 0$
Successor	$S(n)$	$r_n \leftarrow r_n + 1$
Transfer	$T(m, n)$	$r_n \leftarrow r_m$
Jump	$J(m, n, q)$	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 $P = (I_1, \dots, I_n)$ 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 $f : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$ if for every $a_1, \dots, a_n, b \in \mathbb{N}_0$ then:

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

and

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

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 $x_1, \dots, x_n \in \mathbb{N}_0$, is defined by:

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

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

Definition 1.5. Let $M(x_1, \dots, x_n)$ 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:

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

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 $J(m, n, q) \in P$ it holds that $q \leq \#P + 1$.

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

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

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

- $P_1(a_1, a_2, a_3, \dots) \downarrow$ iff $P_2(a_1, a_2, a_3, \dots) \downarrow$;
- when it is the case that both computations $P_1(a_1, a_2, a_3, \dots)$ and $P_2(a_1, a_2, a_3, \dots)$ 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 $l \in \{1, \dots, \#P\}$, $(P; Q)[l] = P[l]$,
- For every $k \in \{1, \dots, \#Q\}$:

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

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

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

$$(T(i_1, 1), \dots, T(i_n, n), Z(n+1), \dots, Z(\rho(P))); P; (T(1, j)),$$

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

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

1. **CallP**,
2. **CallP** $[i_1, \dots, i_n \rightarrow j]$,

where $n, j \geq 1$ and $i_1, \dots, i_n > n$ 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: $\text{zero} = \lambda_x \cdot 0$,
2. the *successor* function: $\text{suc} = \lambda_x \cdot x + 1$,
3. for each $n \in \mathbb{N}_1$ and $i \in \{1, \dots, n\}$, the *projection* function: $U_i^n = \lambda_{x_1, \dots, x_n} \cdot x_i$.

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

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

$$\begin{aligned} g : \mathbb{N}_0^n &\rightarrow \mathbb{N}_0^k \\ (x_1, \dots, x_n) &\mapsto g(x_1, \dots, x_n) \\ &\simeq (g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n)). \end{aligned}$$

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

$$\begin{aligned} f \circ g : \mathbb{N}_0^n &\rightarrow \mathbb{N}_0 \\ (x_1, \dots, x_n) &\mapsto (f \circ g)(x_1, \dots, x_n) \simeq f(g(x_1, \dots, x_n)) \\ &\simeq f(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n)). \end{aligned}$$

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

Theorem 1.3. *Let $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$ and $g_i : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$, for each $i \in \{1, \dots, k\}$, are computable functions, then the function $h = f \circ (g_1, \dots, g_k) : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$ defined, for any $(x_1, \dots, x_n) \in \mathbb{N}_0^n$, by $h(x_1, \dots, x_n) \simeq f(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$, 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 $\mathbf{k}^{(n)}$ the n -ary constant function define as follows:

$$\begin{aligned} \mathbf{k}^{(n)} : \mathbb{N}_0^n &\rightarrow \mathbb{N}_0 \\ (x_1, \dots, x_n) &\mapsto \mathbf{k}^{(n)}(x_1, \dots, x_n) = k \end{aligned}$$

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

1.4 Bounded and Unbounded Quantification

1.5 Alternative Models of Computability

§ Turing Machines A *Turing Machine* M is an abstract device which “performs” operations on a tape of infinite length in both directions. This tape is divided in individual squares along its length. At any given moment each square contains a single symbol from a fixed and finite set called the *alphabet of* $M = \{s_0, \dots, s_n\}$. We assume that s_0 is the *blank* symbol β used to denote an empty square. The machine M has a *reading head* which, at any given moment, scans and reads a single square of the tape. This machine is capable of performing the following three kinds of operations on the tape:

1. Erase the symbol of the square being scanned and write one of the symbols in the alphabet;
2. Move the reading head one square to the right of the square being scanned;
3. Move the reading head one square to the left of the square being scanned.

At any given moment, the machine M is in one of a finite number of *states*, represented by q_1, q_2, \dots, q_m . The execution of an operation may cause the state of M to change.

The operation to be performed by M is determined by its *specification*, denoted by Q , and its current state. The set Q is finite and its elements are quadruples, each of which is of the form (q_i, s_j, s_k, q_l) , or (q_i, s_j, R, q_l) , or (q_i, s_l, L, q_l) , with $i, l \in \{1, \dots, m\}$ and $j, k \in \{1, \dots, m\}$.

A quadruple of the form (q_i, s_j, α, q_l) (where $\alpha \in \{s_k, R, L\}$) in Q specifies the action to be performed by M when it is in state q_i and reading the symbol s_j , as follows:

1. Execute the following operation on the tape:
 - If $\alpha = s_k$, erase s_j and write s_k in the square being scanned;
 - If $\alpha = R$, move the reading head one square to the right;
 - If $\alpha = L$, move the reading head one square to the left.
2. Change into state q_l .

In order for a machine M to begin a computation an initial state is required and its reading head must be positioned over a single square of a given tape. Then M starts performing the actions determined by its specification, as described above. The computation terminates if M is in a state q_p and reading a symbol s_r such that there is no quadruple in Q which starts with $q_p s_r$. Unless stated otherwise, given a Turing machine M and an infinite tape, M starts its computation in the state q_1 and with its reading head placed over the leftmost non-blank square of the received tape.

Example 1.2. Let M be a Turing machine with alphabet $\{\beta, a, b\}$. This machine may be in only one of two states q_1 and q_2 . Let Q be the following specification of M :

$$q_1 a b q_2$$

$$q_1 b a q_2$$

$$q_2 a R q_1$$

$$q_2 b R q_1$$

Definition 1.16. A function $f : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$ is *Turing-computable* if there exists a Turing machine that computes f .

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