Notes on computability and recursion

Benjamin C. Jantzen

April 23, 2013

Note: most material taken directly from *Computability* by N. J. Cutland.

1 The informal notion of following an algorithm

Definition An *algorithm* or *effective procedure* is "... a mechanical rule, or automatic method ... for performing some mathematical operation."

Performance of an algorithm requires no creativity and only the intelligence necessary to follow (or embody) a finite set of directions.

If there is an algorithm for computing a function, we say that function is *effectively calculable*, *algorithmically computable*, or just *computable*.

Example of an effective procedure:
Multiplying two numbers as you are taught in grade school.

Example of a procedure that is not effective:
Decide whether sentence q is a consequence of a set of sentences P by deriving every possible consequence of P.

2 The formal notion of "computability"

2.1 Turing computable

Turing idealized human computation as a succession of very simple operations on a tape of unlimited length:

1. writing or erasing a single symbol.
2. transferring attention from one part of the tape to another.

He then formalized the idea of a machine that carries out computations in this way. (draw Turing machine, showing tape, read head, and internal state indicator)

Any machine M is completely specified by finite set of quadruples: $q_i s_j s_k q_l$. The quadruples are interpreted as follows. When in state q_i and scanning the symbol s_j:

1. Operate on the tape thus:
   (a) if $s_k \neq s_j$ is a symbol, erase s_j and write s_k
   (b) if $s_j = R$, move one square to the right
   (c) if $s_j = L$ move one square to the left
2. Change into state q_l. There is at most one quadruplet beginning with $q_i s_j$. To begin a computation, M must be set in a prescribed initial state. Computation ends when no instruction is available (when there is no instruction that says what to do in the current internal state and reading the current symbol).

Definition The *partial function computed by M* is defined by:

$$f(x) = \begin{cases} \text{the number of occurrences of the symbol 1 on the final tape} & \text{if the computation halts} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Definition A partial function is *Turing-computable* if there is a Turing machine that computes it. The set of all Turing-computable functions is denoted $\mathcal{TC}$.

2.2 The Unlimited Register Machine (URM)

Consists of an infinite number of registers, $R_1, R_2, R_3, \ldots$ each of which contains a natural number. The number in register R_n is denoted r_n. Programs consist of finite sets of instructions, each of which may alter the contents of the registers. Instructions are of four kinds:

1. Zero instructions: $Z(n)$ sets the contents of R_n to 0.
2. Successor instructions: $S(n)$ sets contents of R_n to $r_n + 1$.
3. Transfer instructions: For each $m = 1, 2, 3, \ldots$ and $n = 1, 2, 3, \ldots$ there is an instruction $T(m, n)$ that replaces the number in R_n with the number r_m from R_m.
4. Jump instructions: For each $m = 1, 2, 3, \ldots$, $n = 1, 2, 3, \ldots$, and $q = 1, 2, 3, \ldots$ there is an instruction $J(m, n, q)$ that behaves as follows. If $r_m = r_n$ then proceed to the qth instruction in P. If $r_m \neq r_n$, then go to the next instruction in P.

A computation begins from a prescribed initial configuration, then follows the instructions sequentially. The computation halts when there is no next instruction.

Definition Let f be a partial function from $\mathbb{N}^n$ to $\mathbb{N}$.

1. Suppose that P is a program, and let $a_1, a_2, \ldots, a_n, b \in \mathbb{N}$.
   (a) The computation $P(a_1, a_2, \ldots, a_n)$ *converges to* b if $P(a_1, a_2, \ldots, a_n) \downarrow$ and in the final configuration b is in R_1. We write this $P(a_1, \ldots, a_n) \downarrow b$.
   (b) *P URM-computes f* if, for every $a_1, \ldots, a_n, b$, $P(a_1, \ldots, a_n) \downarrow b$ iff $(a_1, \ldots, a_n, b) \in \text{Dom}(f)$ and $f(a_1, \ldots, a_n) = b$.
2. The function f is *URM-computable* is there is a program that URM-computes f.

2.2.1 Example: Addition

Let M be the URM with instructions: $I_1 = J(2,3,5), I_2 = S(1), I_3 = S(3), I_4 = J(1,1,1), I_5 = Z(2), I_7 = Z(3)$ and let the initial state of the machine be given by $r_1 = a, r_2 = b, r_3 = r_4 = \cdots$ where a and b are non-negative integers. After a finite number of steps, the program halts with $a + b$ in R_1 and 0 everywhere else.

3 The notion of recursive function

3.1 Definitions

Definition The class $\mathcal{R}$ of *partial recursive functions* is the smallest class of partial functions that contains the basic functions:

1. the zero function, $\mathbf{0}$ ($\mathbf{0}(x) = 0$)
2. the successor function, $S(x) = x + 1$
3. for each $n \geq 1$ and $1 \leq i \leq n$, the projection function, $U_i^n(x_1, x_2, \ldots, x_n) = x_i$

and is closed under the operations of substitution, recursion, and minimalisation.

Definition New functions may defined from old by *recursion*. Suppose that $f(\mathbf{x})$ and $g(\mathbf{x}, y, z)$ are functions. Then we can define $h(x, y)$ by:

(i) $h(\mathbf{x}, 0) = f(\mathbf{x})$

(ii) $h(\mathbf{x}, y + 1) = g(\mathbf{x}, y, h(\mathbf{x}, y))$

Definition The *minimilsation operator*, μ, is defined for any function $f(\mathbf{x}, y)$ by:

$$\mu y(f(\mathbf{x}, y) = 0) = \begin{cases} \text{the least y such that} f(\mathbf{x}, y) = 0 & \text{if such exists} \\ \text{undefined} & \text{if there is no such } y \end{cases}$$

3.2 Relation to computability

$\mathcal{R} = \mathcal{TC} = \mathcal{C}$

4 Recursive and recursively enumerable sets

Definition Let A be a subset of $\mathbb{N}$. The *characteristic function* of A is the function C_A given by:

$$c_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

Then A is said to be *recursive* if c_A is computable, or equivalently if "$x \in A$' is a decidable predicate.

Example: The even numbers.

Definition Let A be a subset of $\mathbb{N}$. Then A is recursively enumerable if the function f given by

$$f(x) = \begin{cases} 1 & \text{if } x \in A \\ \text{undefined} & \text{if } x \notin A \end{cases}$$

is computable (or equivalently if the predicate "$x \in A$' is partially decidable).

What's the difference? If a set if recursive, there is an effective procedure that will terminate in finite time and will tell you whether any given natural number is in the set or not. If it is merely recursively enumerable but not recursive, there is no such procedure. However, there is a way to effectively compute a (possibly endless) list of everything in the set.