COMPUTABILITY THEORY AND RECURSIVELY ENUMERABLE SETS

JOSHUA LENERS

Abstract. An algorithm is function from ω to ω defined by a finite set of instructions to transform a given input x to the desired output y. However, simply using a finite number of instructions does not guarantee that a given algorithm will take a finite amount of time. An algorithm is intuitively computable if input x halts, and yields correct output y in a finite amount of time. We introduce the concepts of recursive functions, Turing machines, and partial recursive functions to formally define a computable function. We then introduce (partial) recursive functions and show some basic results about partial recursive functions. We introduce the idea of unsolvable problems, and the reducibility of problems.

Contents

1. Basic Definitions

Definition 1.1. A function is called *primitive recursive* if it is one of (I-III) or is the application of (IV-V) to primitive recursive functions:
(I) The *successor function*, $x \mapsto x+1$.
(II) The *constant functions*, $(x_1, \cdots, x_n) \mapsto m$ where m, $n \geq 0$.
(III) The *identity functions* also called *projections*, $(x_1, \cdots, x_n) \mapsto x_i$ with $1 \leq i \leq n$.
(IV) The composition operation, If $g_1, \cdots, g_m, h$ are primitive recursive, then $f(x_1, \cdots, x_n) = h(g_1(x_1, \cdots, x_n), \cdots, g_m(x_1, \cdots, x_n)$ is also primitive recursive.
(V) The primitive recursion operation, if g, h are primitive recursive with $n \geq 1$ with $f(0, x_2, \cdots, x_n) = g(x_2, \cdots, x_n)$ and
$f(x_1+1, x_2, \cdots, x_n) = h(x_1, f(x_1, \cdots, x_n), \cdots, x_n)$ is also primitive recursive
Thus a function can be defined by a sequence $f_1, \cdots, f_k = f$ where each f_i is is an initial function from (I-III), or the application of IV or V to functions f_j with $j < i$.

Example 1.2. The function for adding two numbers $f(x, y) = x + y$ already has a fairly complicated definition:
$f_1 = x \mapsto x + 1$

$f_x \mapsto x$
$f_3 = (x_1, x_2, x_3) \mapsto x_2$
$f_4 = f_1 \circ f_3$
$f_5(0, x_2) = f_2(x_2)$
$f_5(x_1 + 1, x_2) = f_4(x_1, f_5(x_1, x_2), x_2)$

Notations 1.3. The expression $\mu x P$ where x is a variable and P is a predicate means the least x that satisfies predicate P.
The expression $f(x) \downarrow$ means that f is defined at x. Similarly, $f(x) \downarrow = y$ means f is defined at x and the value is y. $f(x) \uparrow$ means that $f(x)$ is not defined (does not converge).

Since we can define each primitive recursive function with a finite string of symbols, we can obtain an enumeration for them. Let f_n be the n^{th} such function in this enumeration. Under this enumeration n is the *index* of f_n. The function corresponding to $d(x) = f_x(x) + 1$ is clearly intuitively computable. We find the x^{th} primitive recursive function, apply it to x and add one to the output. However, d is not primitive recursive since for all x $d \neq f_x$. This also shows that the listing function $\phi(x, y) = f_x(y)$ is not primitive recursive since we could use it to calculate the function d giving a contradiction. This diagonalization argument affects our characterization because we are only considering total computable functions. In order to consider partial functions we must expand our schemata:

Definition 1.4. A function is partial recursive if it can be defined with respect to (I)-(V) along with an additional rule:
(VI) The unbounded search operation, if ψ is a partial recursive function of $n + 1$ variables and $\phi(x_1, \cdots, x_n) = \mu y[\psi(x_1, \cdots, x_n, y) \downarrow = 0] \& (\forall z \leq y)[\psi(x_1, \cdots, x_n, z) \downarrow]]$ then ϕ is a partial recursive function of n variables.

Remark 1.5. A partial recursive function that is total is called recursive. A primitive recursive function is recursive (since it is total) but a recursive function is not necessarily primitive recursive.

Another characterization of (partial) computable functions is thanks to Alan Turing.

Definition 1.6. A *Turing machine* consists of an infinitely long tape of cells containing zeroes and ones, initially with finitely many ones, and a reading head with finitely many internal states. Based on internal state q_i, the machine reads the symbol at position j, performs some action on that symbol (changing it or keeping it the same), moves either right or left along the tape, and sets its internal state to some q_k.
We can enumerate Turing machines based on the finite strings defining their internal states. We call the machine indexed by e the Turing program P_e.
Then let $\phi_e^{(n)}$ denote the partial function of n variables computed by P_e.

Remarks 1.7. The idea that partial recursive functions (including recursive functions) are the same functions as those which are effectively computable is commonly referred to as Church-Turing Thesis. It is clear that partial recursive functions are intuitively computable, so the Church-Turing Thesis asserts the converse: intuitively computable functions are partial recursive. However, since we are not able

to formally define effective computability except in an intuitive sense, it is impossible to prove Church-Turing Thesis. However, no effectively computable function has been found that cannot be represented by a Turing machine, and hence a partial recursive function.

When we say something is computable, or effectively computable, we are not implying that something could be feasibly calculated. In fact there are a number of partially recursive (and thus by Church-Turing Thesis, effectively computable) functions that would take longer than the estimated lifetime of the universe to complete on today's computers.

2. Basic Results about Partial Recursive Functions

Lemma 2.1 (Padding Lemma)**.** *Each partial recursive function ϕ_x has $\aleph_0$ indices, and furthermore for each x we can effectively find an infinite set A_x of indices for the same partial function (i.e. $\phi_y = \phi_x$ for all $y \in A_x$).*

Proof. Consider the Turing program P_x associated with ϕ_x, consisting of states $\{q_0, \cdots, q_n\}$ we can add an arbitrary number of "idle" steps (that is steps which do not change the tape in any way) as $q_{n+1}, \cdots, q_{n+i}$. Since we have a new set of quintuples, Turing program enumerates differently, thus giving us an infinite number of indices for the Turing program P_x. □

Theorem 2.2 (Normal Form Theorem)**.** *There exists a predicate $T(e, x, y)$ (called the Kleene T-predicate) and a function U(y) which are primitive recursive such that $\phi_e(x) = U(\mu y T(e, x, y))$.*

Theorem 2.3 (Enumeration Theorem)**.** *There is a partial recursive function of 2 variables $\phi_z^{(2)}(e, x)$ such that $\phi_z^{(2)}(e, x) = \phi_e(x)$.*

Proof. Simply let $\phi_z^{(2)}(e, x) = U(\mu y T(e, x, y))$. □

Remark 2.4. The existence of the function $U(\mu y T(e, x, y))$ asserts the existence of a Universal Turing Machine: a Turing Machine which takes two arguments e and x, and produces output $U(y)$, for any P_e that produces a result when applied to x.

Theorem 2.5 (Parameter Theorem)**.** *For every $m, n \geq 1$ there exists an injective recursive function s_n^m such that for all $x, y_1, \cdots, y_m$*
$\phi^{(n)}_{s_n^m(x,y_1,\cdots,y_m)} = (z_1 \cdots z_n) \mapsto \phi_z^{(m+n)}(y_1, \cdots, y_m, z_1, \cdots, z_n)$

Definition 2.6. Let $\langle x, y\rangle$ be the image of (x, y) under the map $\frac{1}{2}(x^2 + 2xy + y^2 + 3x + y)$ which is bijective. Let $\pi_1(\langle x, y\rangle) = x$ and let $\pi_2(\langle x, y\rangle) = y$. Define $\langle x, y, z\rangle = \langle\langle x, y\rangle, z\rangle$, and similarly for larger tuples.

Definition 2.7. We write $\phi_{e,s}(x) = y$ if $x, y, e < s$ and y is the output of $\phi_e(x)$ in fewer than s steps of the Turing program P_e. If such a y exists we say that $\phi_{e,s}$ converges and write $\phi_{e,s} \downarrow$.

Theorem 2.8. *The sets $\{\langle e, x, s\rangle : \phi_{e,s}(x) \downarrow\}$ and $\{\langle e, x, y, s\rangle : \phi_{e,s}(x) = y$ are recursive.*

Proof. Extract e x and s (if applicable y) and compute P_e applied to x until an output is found or s steps have been completed. □

3. Recursively Enumerable Sets and Unsolvable problems

Definition 3.1. A set A (subset of ω) is recursive if the characteristic function χ_A is recursive.

Definition 3.2. A set A is recursively enumerable if A is the domain of some p.r. function.
Let the eth r.e. set be denoted by:
$W_e = \text{domainof}\phi_e = \{x : \phi_e(x) \downarrow\} = \{x : (\exists y)T(e, x, y)\}$

Definition 3.3. Let $K = \{x : \phi_x(x)\text{converges}\} = \{x : x \in W_e\}$.

Proposition 3.4. *K is recursively enumerable.*

Proof. K is the domain of the following partial recursive function:

$$\psi(x) = \begin{cases} x, & \text{if } \phi_x(x) \downarrow \\ \text{undefined} & \text{if otherwise} \end{cases}$$

□

Proposition 3.5. *K is not recursive.*

Proof. Recall K recursive implies χ_K the characteristic function of K is recursive. Then the function

$$f(x) = \begin{cases} \phi_x(x) + 1, & \text{if } x \in K \\ 0 & \text{if } x \notin K \end{cases}$$

would be recursive. However $f \neq \phi_x$ for any x. □

Remark 3.6. This is an example of an unsolvable problem, we have proved that there is no algorithm that computes whether or not a Turing program P_e converges given e. This is closely related to the so-called "halting problem" and as we will show later, it is in fact equivalent.

Definition 3.7. $K_0 = \{\langle x, y\rangle : x \in W_y\}$.

This is the definition of the halting problem. For any program y, given input x does the program halt? If we define recursive characteristic function for this set, we could design an algorithm to answer this problem. (Namely, compute $\chi_{K_0}(\langle x, y\rangle)$.)

Corollary 3.8. *K_0 is not recursive. (i.e. The halting problem is not computable)*

Proof. $x \in K$ iff $\langle x, x\rangle \in K_0$. Finding a recursive characteristic function for K_0 would provide us one for K, which is impossible. □

Definition 3.9. A set A is an index set if for all x and y ($x \in A$ & $\phi_x = \phi_y$) then $y \in A$. In other words, A is the $\aleph_0$ collection of functions that compute ϕ_x.

4. Reducibility

We can see that knowing the answer to some problems can give us the answer to other problems. For instance the fact that K was not recursive, told us a lot about the properties of K_0. This leads to a question of equivalence. The most general way to define an equivalence of problems is to say: "Two problems are equivalent if answering the question 'Is $x \in A$?' tells us whether or not $y \in B$."

Definition 4.1. A set (subset of ω) A is many-one reducible to B ($A \leq_m B$) if there is a recursive function f such that $f(A) \subseteq B$ and $f(\bar{A}) \subseteq \bar{B}$. Put simply $x \in A$ iff $f(x) \in B$.
If f is injective we say that it is one-reducible ($\leq_1$).
If we have $A \leq_m B$ and $B \leq_m A$ we say $A \equiv_m B$. We can define this similarly for one-reducibility.

Definition 4.2. The m-degree (similarly one-degree) of A is the set $\{B : A \equiv_m B\}$

Proposition 4.3. $K \equiv_1 K_0$

Proof. We have shown already that $x \in K$ iff $\langle x, x\rangle \in K_0$, since $\langle ., .\rangle$ is injective we have that $K \leq_1 K_0$. Consider the function

$$\psi(x,y) = \begin{cases} 1 & \text{if } \phi_{\pi_1(x)}(\pi_2(x)) \downarrow \\ \text{undefined} & \text{if otherwise} \end{cases}$$

By the parameter theorem we have a recursive function f such that $\psi(x,y) = \phi_{f(x)}(y)$ for all a, y.

Suppose $x \in K_0$, and $x = \langle s, t\rangle$. Then $\phi_s(t) \downarrow$, which implies that $\phi_{f(x)}(y)$ converges for all y, and in particular $\phi_{f(x)}(f(x))$ converges. Thus we have $x \in K_0 \to f(x) \in K$ by a recursive function.

Suppose that $x \notin K_0$, and $x = \langle s, t\rangle$. Then $\phi_s(t) \uparrow$ so $\phi_{f(x)}(y)$ diverges for all y, and so $\phi_{f(x)}(f(x))$ diverges. Thus we have $x \notin K_0 \to f(x) \notin K$ □

Remark 4.4. These definitions of equivalence are useful, but still to strong to capture the true idea of reducibility, which we will call Turing reducibility ($\leq_T$). We can similarly define Turing equivalence and Turing degree. We will define Turing Reducibility first using Oracle Turing Machines, and then using the formalizations of partial recursive functions.

Definition 4.5. An Oracle Turing machine, is a Turing machine as described before with an additional tape A called the oracle, the squares on this tape are determined by the function χ_A. While it is operating the Turing machine reads from both machines at x in order to determine its actions.

Definition 4.6. We can treat a Oracle Turing Machine as a partially recursive function, for the set A as having the additional basic function χ_A added to (I)-(VI). We say that this function is Turing computable in A.

If we can use A to solve B, we know that we have Turing reducibility, in the thus if we have the proper oracle tape, we can decide any problem in its Turing Degree.

References

[1] R. I. Soare. Recursively Enumerable Sets and Degrees. Springer-Verlag. 1987.