

A Brief Note on Church-Turing Thesis and R.E. Sets

Chung-Chih Li

licc@hal.lamar.edu

Dept. of Computer Science, Lamar University
Beaumont, Texas, USA

March 18, 2004

Church-Turing Thesis: All formalisms for computable functions are equivalent.

This is the **only** right version of Church-Turing Thesis. However, with tons of philosophical debates, we have the following variation; I call it Enhanced Church-Turing Thesis.

Enhanced Church-Turing Thesis: All informal notions of computation can be implemented on Turing Machines.

People who believe in the Enhanced Church-Turing Thesis assume that the origin of every formalism for computable functions must be some sort of informal notions. Since all formalisms we ever have turn out to be equivalent, it must be the case that their origins are also equivalent. I hardly find myself at a position to believe in this, since I do not see sufficient reasons to further believe that nothing will lose when we try to formalize our informal notions of computation in symbols. We speak of those we can speak of, and pass a lot more in silence.

A mild variation is that “all algorithms are computable by Turing machines”. This is fine with me, since an algorithm in our computer science discipline is “formal” enough to me. So, from now on, if one asks you to prove something that is computable, don’t be shy to give him/her an algorithm and claim your victory, provided that the algorithm is correct. In other words, “a program” and “a Turing machine” are considered equivalent.

Terminologies —

Given an *acceptable programming system* φ , let φ_i denote the function computed by i^{th} φ -program. By Church-Turing thesis, we can leave out detail construction between an acceptable programming system and the formalism for Turing machines, and we are free to switch the meaning of φ_i from the function computed by i^{th} φ -program to the function computed by i^{th} Turing machine. Also, borrowing from the idea of Gödel’s numbers (that the entire universe can be coded by natural numbers), we consider each natural number a Turing machine under a fixed explanation. Thus, if

we accept the bold idea (as I do) that computability cannot go beyond Turing machine, then it is trivial to claim that all (partial) computable functions are enumerable, where we simply write a program to enumerate all natural numbers:

$$0, 1, 2, 3, \dots$$

Unlike many textbooks suggest, we don't have to be bothered by an arbitrary number n that cannot be interpreted as a set of legal instructions for a Turing machine, since we can simply consider n as a broken Turing machine that won't compute anything meaningful. Think of broken C++ programs; no C++ compiler has any problems with them. Clearly, for every $i \in \mathbf{N}$, φ_i computes some function, and the function may not be total. We use $f(x) \uparrow$ to denote that f is undefined on x , and use $\varphi_i(x) \uparrow$ to denote that the φ -program i on input x runs forever. Also, we use $\varphi_i(x) \downarrow$ to denote that the φ -program i on input x will stop and output something eventually; and that something is the value of $\varphi_i(x)$. Moreover, we use $\varphi_i(x) \uparrow_s$ to denote that the φ -program i on input x does not stop in s steps. Similarly, $\varphi_i(x) \downarrow_s$ denotes that the φ -program i on input x does stop in s steps. Apparently, both $\varphi_i(x) \uparrow_s$ and $\varphi_i(x) \downarrow_s$ are decidable.

- **Recursive functions**

A function, f , is said to be recursive if f is total and there is a φ -program for it.

- **Partial Recursive functions**

A function, f , is said to be partial recursive if there is a φ -program for it.

Theorem 1 *There is a total function that is not recursive.*

Proof: Define f as follows: for every $x \in \mathbf{N}$,

$$f(x) = \begin{cases} \varphi_x(x) + 1 & \text{if } \varphi_x(x) \downarrow; \\ 0 & \text{if } \varphi_x(x) \uparrow. \end{cases}$$

It is clear that f is total. We shall prove that there is no φ -program for f . By contradiction, suppose $\varphi_e = f$. By the definition of f , $\varphi_e(x) \downarrow$ for every x . Thus, $\varphi_e(e) = \varphi_e(e) + 1$. This is a contradiction. Therefore, f is not recursive. □

Note that, all primitive recursive functions are recursive, but the set of all primitive recursive functions does not include all recursive functions. Ackermann function is one of the first two non-primitive recursive functions discovered in 1928 by Ackermann (the other one is Sudan's function discovered independently in 1927). To prove that the Ackermann function is computable is easy with today's tools – Turing machines and Church-Turing thesis. By using the technique of double mathematical inductions, that the Ackermann function is recursive can be easily proven also. However, to prove that the Ackermann function is not primitive is a bit involved. We argue that the Ackermann function grows much faster than any primitive recursion functions. We skip all details here.

Let $\langle \cdot, \cdot \rangle : \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$, be a standard pairing function. That is, $\langle \cdot, \cdot \rangle$ is recursive and bijective. In fact, there is an efficient program to compute $\langle \cdot, \cdot \rangle$, but efficiency is not our concern at this

moment. Also, there are two recursive functions π_1 and π_2 such that, for all $x, y \in \mathbf{N}$, we have $\pi_1(\langle x, y \rangle) = x$ and $\pi_2(\langle x, y \rangle) = y$. I leave you to justify the statements above as exercise.

The advantage of having $\langle \cdot, \cdot \rangle$, π_1 and π_2 is that, for example, we can express the following function

$$f(n) = \pi_1(n) + \pi_2(n)$$

in a bit clearer way:

$$f(\langle x, y \rangle) = x + y.$$

Also, the computability of any function does not change if we use $\langle \cdot, \cdot \rangle$, π_1 and π_2 in the function definition.

Given a function f , the domain of f is the set $\{n \mid f(n) \text{ is defined}\}$.

- **Recursive sets**

A set, S , is said to be recursive if there is recursive characteristic function for S .

- **Recursively Enumerable sets**

A set, S , is said to be recursively enumerable if there is a recursive function f such that,

$$S = \{f(n) \mid n \in \mathbf{N}\}.$$

The following statements are equivalent.

1. S is recursively enumerable.
2. S is a set such that, there is a partial recursive function f such that, for every $n \in \mathbf{N}$,

$$n \in S \iff f(n) = 1.$$
3. S is the domain of some partial recursive function f .
4. S is the set of solutions to some Diophantine equation.

To prove that 1. 2. and 3. are equivalent is easy. I leave it as an exercise. 4. is a celebrated theorem proven in 1970 due to Matiyasevitch. The theorem implies a negative answer to Hilbert's 10th problem and perfectly unifies arithmetic and computability theory. Unfortunately, the proof is way beyond the scope of this class.

- **Halting problem:** *Given any φ -program i , does the program stop on taking itself as the input.*

We restate this problem in terms of set as follows.

$$K = \{i \mid i \in \mathbf{N} \text{ and } \varphi_i(i) \downarrow\}.$$

Thus, to answer the halting problem with regard to program i , we check if $i \in K$. K will be our first example of recursively enumerable sets (Theorem 10) that are not recursive (Theorem 2). In fact, K is such a powerful set in a sense that every r.e. set can be reduced to K . We will introduce the concept of reducibility later (in the class, if I don't have time to write it up here).

Theorem 2 K is not recursive.

Proof: By contradiction, suppose there is a recursive characteristic function χ for K . That is, for every $x \in \mathbf{N}$, $\chi(x) = 1$ iff $\varphi_x(x) \downarrow$ and $\chi(x) = 0$ iff $\varphi_x(x) \uparrow$. Define f as follows: for every $x \in \mathbf{N}$

$$f(x) = \begin{cases} 0 & \text{if } \chi(x) = 0; \\ \uparrow & \text{if } \chi(x) = 1. \end{cases}$$

Since χ is recursive, there is a φ -program for χ , and hence f is computable and there is a φ -program for it. Note that, for \uparrow in the definition of f , we can simply let our program go to an infinite loop. Let e be a φ -program for f . Then, what will be the result of running φ -program e on e ? If $\varphi_e(e) = 0$, that means $\chi(e) = 0$; but $\chi(e) = 0$ means $\varphi_e(e) \uparrow$. A contradiction! On the other hand, suppose $\varphi_e(e) \uparrow$ in case that $\chi(e) = 1$; but if $\chi(e) = 1$, that means $\varphi_e(e) \downarrow$. A contradiction too! Therefore, the assumption that there is a recursive characteristic function χ for K is impossible. \square

Using similar arguments, one can easily prove the following theorems.

Theorem 3 $\{i \mid \varphi_i(0) \downarrow\}$ is not recursive.

Theorem 4 $\{i \mid \varphi_i(0) = 0\}$ is not recursive.

Theorem 5 $\{i \mid \varphi_i(0) \leq \varphi_i(1)\}$ is not recursive.

Theorem 6 $\{i \mid \varphi_i \text{ is total}\}$ is not recursive.

Theorem 7 $\{i \mid \exists x \varphi_i(x) \downarrow\}$ is not recursive.

Don't be surprised we can easily give a nonrecursive set. And don't be sorry either; we have many useful sets they are all recursive; such as regular languages and context-free languages we have learned. The technique for proving theorems 2 to 7 are all the same: **diagonalization**, in other words: **making trouble**.

Define $E = \{\langle i, j \rangle \mid \varphi_i = \varphi_j\}$. That is, E is the set of all pairs of φ -programs that compute the same function. Let \overline{E} be the complement of E . We will prove that E and \overline{E} are not recursive.

Theorem 8 E is not recursive.

Proof: Same technique: diagonalization. By contradiction, suppose there is a recursive characteristic function χ for E . Let a be a φ -program that outputs 0 on every input, i.e., $\forall x \varphi_a(x) = 0$. Construct φ -program e as follows: for every $x \in \mathbf{N}$

$$\varphi_e(x) = \begin{cases} 0 & \text{if } \chi(\langle a, e \rangle) = 0; \\ 1 & \text{if } \chi(\langle a, e \rangle) = 1. \end{cases}$$

Since χ is recursive, so is φ_e . It is clear that χ can't correctly give a right answer on $\langle a, e \rangle$. (Why? Complete the argument. I need a complete argument in the final!) \square

Theorem 9 \overline{E} is not recursive.

Theorem 10 K is recursively enumerable.

Proof: Here we consider a recursively enumerable set as a set its elements can be enumerated by a recursive function f . Let a be a φ -program that outputs 0 on every input. Clearly, $a \in K$. Define f as follows: for every $i, s \in \mathbf{N}$,

$$f(\langle i, s \rangle) = \begin{cases} i & \text{if } \varphi_i(i) \downarrow_s; \\ a & \text{if } \varphi_i(i) \uparrow_s. \end{cases}$$

Since whether or not the computation of $\varphi_i(i)$ halts in s steps is recursively decidable, it follows that f is recursive. Therefore, if $i \in K$, $\varphi_i(i)$ must halt in some finitely many steps, say s steps. Let $\langle i, s \rangle = n$. We have $f(n) = i$. On the other hand, if $i \notin K$, $\varphi_i(i)$ will not halt no matter how many steps we allow it to compute. Thus, for every $s \in \mathbf{N}$, $f(\langle i, s \rangle) = a \neq i$. \square

Theorem 11 Let A be a set. If A and \overline{A} are both recursively enumerable, then A is recursive.

I leave the proof as an exercise.

Theorem 12 \overline{K} is not recursively enumerable.

With Theorem 11, we can argue that a set A is not recursively enumerable if we can show that \overline{A} is recursively enumerable but not recursive. Thus, we have Theorem 12 immediately. However, if we cannot prove \overline{A} is recursively enumerable, Theorem 11 cannot help us to prove that A is not recursively enumerable. Sets E and \overline{E} are such examples.

Theorem 13 E is not recursively enumerable.

Proof: By contradiction, suppose there is a recursive function f such that, $E = \{f(n) \mid n \in \mathbf{N}\}$. That is, $\varphi_i = \varphi_j$ iff $\exists n \in \mathbf{N} f(n) = \langle i, j \rangle$. Let e be a φ -program that diverges everywhere, i.e.,

$$\varphi_e(x) = \uparrow \quad \text{for every } x.$$

Define φ_i as follows: for every $x \in \mathbf{N}$,

$$\varphi_i(x) = \begin{cases} 0 & \text{if } \exists n \in \mathbf{N} f(n) = \langle i, e \rangle; \\ \uparrow & \text{if no such } n \text{ exists.} \end{cases} \quad (1)$$

Note that, this construction is legitimate since if the n in (1) does not exist, the search will go forever and the φ -program i will never stop as we want on all inputs. But, what is $\varphi_i(x)$, really? We have two cases.

- Case 1: $\forall n f(n) \neq \langle i, e \rangle$. If this is the case, then $\varphi_i(x) \uparrow$ for every x . Thus, $\varphi_i = \varphi_e$. By the assumption, there is an n such that $f(n) = \langle i, e \rangle$. A contradiction arrives.
- Case 2: $\exists n f(n) = \langle i, e \rangle$. If this is the case, then $\varphi_i(x) = 0$ for every x . Thus, $\varphi_i \neq \varphi_e$. By the assumption, there is no n such that $f(n) = \langle i, e \rangle$. A contradiction arrives.

In both cases, we have a contradiction. Therefore, the recursive function f that enumerates \overline{E} does not exist. □

Theorem 14 \overline{E} is not recursively enumerable.

Proof: By contradiction, suppose there is a recursive function f such that, $\overline{E} = \{f(n) \mid n \in \mathbf{N}\}$. That is, $\varphi_i \neq \varphi_j$ iff $\exists n \in \mathbf{N} f(n) = \langle i, j \rangle$. Let a be a φ -program that outputs 0 everywhere, i.e.,

$$\varphi_a(x) = 0 \quad \text{for every } x.$$

Define φ_i as follows: for every $x \in \mathbf{N}$,

$$\varphi_i(x) = \begin{cases} 0 & \text{if } \exists n \in \mathbf{N} f(n) = \langle i, a \rangle; \\ \uparrow & \text{if no such } n \text{ exists.} \end{cases} \quad (2)$$

We have two cases.

- Case 1: $\forall n f(n) \neq \langle i, a \rangle$. If this is the case, then $\varphi_i(x) \uparrow$ for every x . Thus, $\varphi_i \neq \varphi_a$. By the assumption, there is an n such that $f(n) = \langle i, a \rangle$. A contradiction arrives.
- Case 2: $\exists n f(n) = \langle i, a \rangle$. If this is the case, then $\varphi_i(x) = 0$ for every x . Thus, $\varphi_i = \varphi_a$. By the assumption, there is no n such that $f(n) = \langle i, a \rangle$. A contradiction arrives.

In both cases, we have a contradiction. Therefore, the recursive function f that enumerate \overline{E} does not exist. □