A Note on Church-Turing Thesis for the Foundation of Computation Course.

Chung-Chih Li*
School of Information Technology
Illinois State University
Normal, IL 61790, USA

April 24, 2008

One of the most astonishing achievements in the 20^{th} century is the invention of computers. The highest honor for a computer scientist who has made significant contributions to the field is to receive the Turing Award, which is recognized as the Nobel Prize in computer science. The award is named after Alan Turing in order to honor him for his tremendous contributions in the field of computer science ranged from Artificial Intelligence, Cryptography, to the foundation of computer science.

On the top of the list of Turing's contributions is his imaginary mechanical computing device now called Turing Machine. Turing used his imaginary machines to successfully formalize the nature of computing in an intuitive way people had been looking for but failed to see it for more than two millennia. Not that people before Turing didn't know what computing was, but that whenever people proposed a formal device to carry the concept of computing, there's always a doubt about how comprehensive the device could be, and there's always a desire to push the limit farther. Turing cleaned up the doubt and drew a clear line about the limitation of computing and showed that any reasonable notions of computation are all the same. For example, the formalism proposed by Alonzo Church called λ-calculus bears the same power of computing as the Turing machine does. Likewise, Kurt Gödel and Steven Kleene's recursive functions describe the same class of functions that can be computed by the Turing Machines. Gödel was convinced that the system of λ-calculus was equivalent to his recursive functions, but when Church further claimed the thesis (the thesis we will discuss in a moment), Gödel turned skeptical and refused to accept it until Turing came along with his work. Alan Turing, then was a student in Princeton under Church's supervision[1], came out with his proposal of computing machines to characterize all computable

*e-mail: cli2@ilstu.edu

[1]John von Neumann encouraged Turing to come to the US to complete his doctoral degree. Since Princeton University rejected Turing's scholarship, not many people could recognize important thing as Johnny did after all, von Neumann used his own money to support Turing's study. After WWII, von Neumann actually constructed the very first modern computer with the structure known as the von Neumann machine, which is still the underlaying principle of today's (and tomorrow's, I believe) computers. Von Neumann apparently was inspired by Turing's work. In fact, the von Neumann machine to the most is just a vacuum-tube version of the Universal Turing Machine. It took another decade for Marvin Minsky to come out with his Universal Register Machine (URM), which is an abstract version of von Neumann machine, and Minsky proved that URM is equivalent to the Universal Turing Machine. Since

real numbers and Church presented it to Gödel. After he carefully reviewed Turing's work, Gödel changed his mind and accepted the thesis. (Gödel was not a guy who would change mind easily. He in fact worried about that there may be a possible flaw in Turing arguments. Gödel was paranoid. Remember?) Although Turing himself did not explicitly claim the thesis, his work helps the thesis to be comprehend with a more intuitive notion, we now call the claim *Church-Turing Thesis.*

A remark by Ludwig Wittgenstein is pretty precise: Turing Machines are humans who calculate. By "Calculate" we mean "follow the rule to reach the conclusion". If the rule is fixed and the information is comprehensive, we should have any discrepancy in our conclusion. I will skip the formal definition of Turing machines, which is rather standard and can be found in every textbook of the theory of computation. However, only with a few exceptions, the Church-Turing Thesis are always stated in an unsatisfactory way, and that may lead to a very wrong understanding. For example, let me just quote a paragraph from Ray Kurzwell's popular book, "The age of Spiritual Machines", page 268:

> "The [Church-Turing] thesis states that all problems that a human being can solve can be reduced to a set of algorithms, supporting the idea that machine intelligence and human intelligence are essentially equivalent."

The first time I read the passage, my reaction was: Oh my God!! What on earth this guy could read the thesis this way? To clean up this misconception, I compose this brief note.

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

This is the **only** right version of Church-Turing Thesis. Note that, there is no proof for Church-Turing Thesis. The thesis is more like an empirical statement. (The problem is, "all formalisms", products of human minds, apparently are not effectively enumerable. However, with many heavy 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

then, no one could push the concept of computation any farther. There have been a few effort motivated by real needs and applications to try to lift the concept of computation into another level. For example, quantum computing, higher order computation, biocomputation, and so on. However, the foundational obstacle for further pursue those directions to a sound computability theory is the lack of an equivalent Church-Turing thesis.

algorithm and claim your victory (of course, your algorithm has to be correct). In other words, "a program" and "a Turing machine" are considered equivalent.

Applications of Church-Turing Thesis

Terminologies for a bit theoretical stuff: 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 detailed 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 the 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:

$$0, 1, 2, 3, \ldots.$$

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 if 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 have to argue that the Ackermann function grows much faster than any primitive recursion functions. We skip all details here.

Let $\langle \cdot, \cdot \rangle : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$, 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 10^{th} 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 introduced 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. □

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 $\{i \mid \varphi_i$ *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!) □

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 $s \in \mathbf{N}$, $f(\langle i, s\rangle) = a \neq i$. □

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 not 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 $x \in \mathbf{N}$,

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

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 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 $x \in \mathbf{N}$,

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

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. □

A nice theorem:

Now, let me provide a proof for one of my favorite theorems. I think my version had simplified the original proof a little. Church-Turing Thesis is implicitly used everywhere.

Theorem 15 Kreisel-Lacombe-Shoenfield Theorem *(1957)* $\mathbf{F}$ *is an effective operation total on* $\mathcal{R}$ *if and only if* $\mathbf{F}$ *is a recursive functional restricted on* $\mathcal{R}$ *and which is total on* $\mathcal{R}$

Proof:

($\Leftarrow$) Straightforward. Given $\mathbf{F}$ as required, we can compute $\psi(y)$ for any $y \in N$ as following,

> Enumerate $\varphi_y(0), \varphi_y(1), \ldots,$ and feed into $\mathbf{F}$ until $\mathbf{F}$ outputs an answer. Use this answer as the value of $\psi(y)$.

($\Rightarrow$) Given ψ which determines an effective operation $\mathbf{F}$, and which is total on $\mathcal{R}$. We want to construct a functional $\mathbf{G}$ as required.

Main idea: By the totality of ψ on $\mathcal{R}$, if $f \in \mathcal{R}$ then $\mathbf{F}(f) \downarrow$, and by the compactness and monotonicity of ψ, there is a finite initial segment $\tilde{f}$ of f, such that $\mathbf{F}(\tilde{f}) \downarrow = \mathbf{F}(f) \downarrow$, and $\mathbf{F}$ converges to $\mathbf{F}(f)$ on all possible extension of $\tilde{f}$. If we are able to decide the minimal length of the initial segment of f on which $\mathbf{F}$ converges, then we can collect the infomation about f until the required segment appears, and then we can find an index of a recursive function which contains the segment [2] and apply ψ to this index. Finally, we will use the value from ψ as the value of $\mathbf{G}$ on f.

We will use the Recursion Theorem to find out the minimal length of the initial segment we need for every recursive function f.

Define a function h with two variables, such that for all x and y,

$$\varphi_{h(x,y)} = \begin{cases} \emptyset & \text{if } \psi(y) \uparrow, \\ \varphi_y & \text{if } \psi(y) \downarrow \wedge \psi(x) \uparrow, \\ \varphi_y & \text{if } \psi(y) \downarrow \neq \psi(x) \downarrow, \\ \tilde{f}^0 & \text{if } \psi(y) \downarrow = \psi(x) \downarrow, \wedge \psi(x) \text{ needs } \omega \text{ steps}, \\ & \quad \wedge \{0, \ldots, \omega\} \subset \text{ domain}(\varphi_y), \wedge \exists \tilde{f} [\varphi_y^{[\omega]} \subset \tilde{f} \wedge \psi(y) \neq \mathbf{F}(\tilde{f}^0)], \\ \varphi_y^{[\omega]} & \text{o/w, where } \psi(x) \text{ needs } \omega \text{ steps.} \end{cases}$$

Claim: h is recursive. We skip the proof for this claim. (Hint: $\varphi_y(x) = \tilde{f}^0(x)$ if $x \leq \omega$.)

[2] We may use the zero extension of $\tilde{f}$, but we are not necessary to do so since $\mathbf{F}(\tilde{f})$ will converge even $\tilde{f} \notin \mathcal{R}$.

do $i = 0, 1, \ldots$ until stop;
  let μ^0 be an index of $\lambda x \begin{cases} f(x) & \text{if } x \leq i; \\ 0 & \text{o/w.} \end{cases}$
  if $\psi(n(\mu^0)) \downarrow$ exactly in i steps,
    then output($\psi(\mu^0)$), stop;
    else next i;
next i;

Figure 1: Program g

By the Recursion Theorem, there exists $n \in \mathcal{R}$, such that

$$\varphi_{n(y)} = \varphi_{h(n(y),y)} = \begin{cases} \emptyset & \text{if} \quad \psi(y) \uparrow, & \text{case i} \\ \varphi_y & \text{if} \quad \psi(y) \downarrow \wedge \psi(n(y)) \uparrow, & \text{case ii} \\ \varphi_y & \text{if} \quad \psi(y) \downarrow \neq \psi(n(y)) \downarrow, & \text{case iii} \\ \tilde{f}^0 & \text{if} \quad \psi(y) \downarrow = \psi(n(y)) \downarrow, \wedge \psi(n(y)) \text{ needs } \omega \text{ steps}, & \\ & \quad \wedge \{0, \ldots, \omega\} \subset \text{ domain}(\varphi_y), & \\ & \quad \wedge \exists \tilde{f}[\varphi_y^{[\omega]} \subset \tilde{f} \wedge \psi(y) \neq \mathbf{F}(\tilde{f}^0)], & \text{case iv} \\ \varphi_y^{[\omega]} & \text{o/w, where } \psi(n(y)) \text{ needs } \omega \text{ steps.} & \text{case v} \end{cases}$$

Claim: If $\varphi_y \in \mathcal{R}$, then $\varphi_{n(y)} = \varphi_{h(n(y),y)} = \varphi_y^{[\omega]}$.

Proof of the claim:

1. Case i fails because $\psi(y) \downarrow$.
2. Case ii fails, otherwise $\varphi_{n(y)} = \varphi_y, then \psi(n(y)) \downarrow = \psi(y) \downarrow$.
3. Case iii fails, otherwise, if $\psi(y) \downarrow \neq \psi(n(y)) \downarrow, then \varphi_{n(y)} = \varphi_y and \psi(y) \downarrow = \psi(n(y)) \downarrow$, and hence a contradiction.
4. Case iv fails. By means of contradiction, suppose case iv holds and there is y, $\varphi_y \in \mathcal{R}$. That is, $\varphi_{n(y)} = \tilde{f}^0$ and $\psi(n(y)) \downarrow = \psi(y) \downarrow$, where $\tilde{f}$ is an extension of $\varphi_y^{[\omega]}$ such that $\mathbf{F}(\tilde{f}^0) \downarrow \neq \psi(y) \downarrow$. That implies $\psi(n(y)) \downarrow \neq \psi(y) \downarrow$ and leads a contradiction. The failure of this case means all possible zero extensions of $\varphi_y^{[\omega]}$ won't change the $\mathbf{F}$'s output which has stuck with the value equal to $\psi(f)$.
5. Case v holds, that means given y if $\varphi_y \in \mathcal{R}$, $\psi(n(y))$ must converge in ω steps and $\varphi_{n(y)} = \varphi_y^{[\omega]}$. Therefore, $\psi(n(y)) = \psi(y) =$ the value of ψ on any possible zero extension of $\varphi_y^{[\omega]}$.

Construct G: Given f in term of its graph. Consider the program g in Figure 1.[3] We will use the program g to search the initial segment of f we need, and let the program terminate in the loop when $i = k$, and μ_k^0 be the value of μ^0.

[3] I have another versions. I think both are valid.

Question: It is trivial to prove that such ω exists, but the question is: must k be large enough to reach ω? In another words, is it possible that there is a k less than ω and k happens to satisfy the condition to get rid of the loop, but $\psi(y) \neq \psi(\mu_k^0)$?

The answer to the question is the answer to that: does this functional **G** we have constructed and the functional determined by ψ coincide on $\mathcal{R}$?

Claim: $k = \omega$.

Proof of the claim: By way of contradiction, suppose $k < \omega$ and $\psi(y) \neq \psi(\mu_k^0)$. We take $\bar{y}$ as the original y, and take μ_k^0 as y and k as ω in the definition of $\varphi_{h(n(y),y)}$. Clearly, $\varphi_y \in \mathcal{R}$ but case iv will hold, because there is an extension $\tilde{f}$ of $\varphi_y^{[\omega]}$ such that $\varphi_y^{[\omega]} \subset \tilde{f} \subset \varphi_{\bar{f}}$.

Exercise: Define a recursive function h, for all $a \in \mathbf{N}$, which outputs a program for an effective operation total on $\mathcal{R}$.

$$\varphi_{h(a)} = \lambda y \begin{cases} \uparrow & \text{if } \exists x(0 \leq x \leq a \ \& \ \varphi_y(x) \uparrow), \\ \max\{\varphi_y(x) | 0 \leq x \leq a\}. \end{cases}$$

Is there an a that is so large such that there is a recursive function f, and $\varphi_{h(a)}(n(\hat{f^{[i]}})) \downarrow$ exactly in i steps, and $i < a$. Where $\hat{f^{[i]}}$ is an index of the zero extension of $f^{[i]}$.

From the above proof, the answer is NO. Once a is fixed, for all $f \in \mathcal{R}$ and for all i, $\varphi_{h(a)}(n(\hat{f^{[i]}}))$ will not converge in a steps. □

Version 2:

do $i = 0, 1, \ldots$ until stop;

  do $j = 0, \ldots, i$

    if $\psi(n(j))$ not $\downarrow$ in i steps, go to next j;
    let ω be the number of steps $\psi(n(j)) \downarrow$ needed;
    if exists k, $0 \leq k \leq \omega$ such that $\varphi_j^{[i]}(k) \uparrow$ or $\varphi_j(k) \neq f(k)$, go to next j;
    let μ^0 be an index of $\lambda x \begin{cases} \varphi_j(x) & \text{if } x \leq i; \\ 0 & \text{o/w.} \end{cases}$
    output($\psi(\mu^0)$), stop;

  next j;

next i;